**ACADEMIA DE TRANSPORTURI,
INFORMATICĂ ŞI COMUNICAŢII**

Zgureanu Aureliu

CRIPTAREA
ŞI
SECURITATEA INFORMAŢIEI

Note de curs

CHIŞINĂU 2013

Notele de Curs la disciplina „Criptarea şi securitatea informaţiei” a fost examinat şi aprobat la şedinţa catedrei „Matematică şi Informatică”, proces verbal nr. 3 din 11.11.2013, şi la şedinţa Comisiei metodice şi de calitate a FEI, proces verbal nr. 1 din 02.12.2013.

Cuprins

Introducere

Una dintre caracteristicile societăţii moderne o reprezintă Informatizarea. Tehnologii informaţionale noi sunt permanent implementate în diverse domenii ale activităţii umane. Prin utilizarea calculatoarelor şi a software-ului respectiv sunt dirijate procese complexe din cele mai diverse domenii de activitate. Calculatoarele stau la baza mulţimilor de sisteme de prelucrare a informaţiei, care înfăptuiesc păstrarea, prelucrarea informaţiei, distribuirea ei către utilizator, realizând astfel tehnologii informaţionale moderne.

Odată cu dezvoltarea mecanismelor, metodelor şi formelor de automatizare a proceselor de prelucrare a informaţiei creşte şi dependenţa societăţii de gradul de securitate a proceselor de gestionare a informaţiei, realizat prin intermediul diverselor tehnologii informaţionale aplicate, de care depinde bunăstarea, sau uneori şi viaţa multor oameni.

În acest context un specialist modern din domeniul Tehnologiilor Informaţionale este obligat să posede cunoştinţe şi aptitudini de asigurare a securităţii informaţiei în toate fazele de dezvoltare şi de funcţionare a sistemelor informaţionale.

Soluţionarea problemelor legate de securitatea informaţiei constituie obiectul de studiu al ***Criptografiei***, care este o ramură a matematicii moderne, ce se ocupă de elaborarea metodelor matematice capabile să asigure confidenţialitatea, autentificarea şi non-repudierea mesajelor, precum şi integritatea datelor vehiculate. Criptografia este un set de standarde şi protocoale pentru codificarea datelor şi mesajelor, astfel încât acestea să poată fi stocate şi transmise mai sigur. Ea stă la baza multor servicii şi mecanisme de securitate, folosind metode matematice pentru transformarea datelor, în intenţia de a ascunde conţinutul lor sau de a le proteja împotriva modificării. Criptografia ne ajută să avem comunicaţii mai sigure, chiar şi atunci când mediul de transmitere (de exemplu, Internetul) nu este de încredere. Ea poate fi utilizată pentru a contribui la asigurarea integrităţii datelor, precum şi la menţinerea lor în calitate de date secrete, ne permite să verificăm originea datelor şi a mesajelor prin utilizarea semnăturilor digitale şi a certificatelor.

Unul dintre instrumentele principale ale criptografiei este sistemul de criptare, la baza căruia se află algoritmul de criptare.

Scopul acestui curs este de familiariza viitorul specialist din domeniul Ttehnologiilor Informaţionale cu noţiunile fundamentale ale criptografiei şi cu metodele moderne criptare ca instrument indispensabil al securităţii informaţiilor.

Tema 1. Noţiuni de bază ale Criptografiei.

Criptografia reprezintă o ramură a matematicii care se ocupă cu securizarea informaţiei, precum şi cu autentificarea şi restricţionarea accesului într-un sistem informatic. În realizarea acestora se utilizează în mare parte metode matematice, profitând de unele probleme cu complexitate de rezolvare suficient de înaltă. Termenul „*criptografie*" este compus din cuvintele de origine greacă *κρυπτός*, kryptós (ascuns) şi γράφειν, gráfein (a scrie). Criptografia urmăreşte următoarele obiective:

1. *Confidenţialitatea* (*privacy*) – proprietatea de a păstra secretul informaţiei, pentru ca aceasta să fie folosită numai de persoanele autorizate.
2. *Integritatea datelor* – proprietatea de a evita orice modificare (inserare, ştergere, substituţie) neautorizată a informaţiei.
3. *Autentificare* – proprietatea de a identifica o entitate conform anumitor standarde. Este compusă din:
    a. autentificarea unei entităţi;
    b. autentificarea sursei informaţiei;
4. *Non-repudierea* – proprietatea care previne negarea unor evenimente anterioare.

Celelalte obiective legate de securitatea informaţiei (*autentificarea mesajelor*, *semnături*, *autorizare*, *validare*, *controlul accesului*, *certificare*, *timestamping*, *confirmarea recepţiei*, *anonimitate*, *revocare*) pot fi derivate din aceste patru.

Împreună cu Criptografia se dezvoltă *Criptanaliza* – (din greacă, *kryptós*, „ascuns", şi *analýein*, „a dezlega") este studiul metodelor de obţinere a înţelesului informaţiilor criptate, fără a avea acces la informaţia secretă necesară în mod normal pentru aceasta. De regulă, aceasta implică găsirea unei chei secrete.

Criptografia şi Criptanaliza împreună constituie *Criptologia* (din greacă, *kryptós*, „ascuns", şi λόγος, „cuvânt") – ştiinţa care se ocupă cu metodele de criptare şi decriptare.

În continuare sunt date noţiunile fundamenatle cu care se operază în croptologie.

O mulţime nevidă $\boldsymbol{T}$ se numeşte *alfabet*.

Elementele alfabetului ***T*** se numesc *litere*. Una şi aceeaşi literă poate intra într-un cuvânt de mai multe ori.

O consecutivitate finită de elemente din alfabetul ***T*** se numeşte *cuvânt*.

Numărul de elemente ale alfabetului se numeşte *lungimea alfabetului*.

Un cuvânt ce nu conţine nici o literă se numeşte *cuvânt nul*.

Lungimea cuvântului, notată cu ***w***, este numărul de litere în acest cuvânt, unde fiecare literă se consideră de câte ori se întâlneşte în el.

Vom nota cu T^* mulţimea tuturor cuvintelor alfabetului ***T***.

Submulţimile mulţimii T^* le vom numi *limbaje* (formale) peste ***T***.

Un mesaj în forma sa originară se numeşte *text clar* (uneori *text în clar*, în engleză *plaintext*) şi îl vom nota cu ***pt*** sau cu ***m*** (de la „*message*" - mesajul).

Rescrierea textului clar, folosind o metodă cunoscută numai de expeditor (eventual şi de destinatar), se numeşte *criptare* (sau *cifrare*) a mesajului.

Text criptat sau *text cifrat* (în engleză *ciphertext*) se numeşte textul obţinut în rezultatul operaţiei de criptare a textului plan. Textul criptat îl vom nota cu ***ct*** sau cu ***c*** (de la „*cipher*" - cifrul). Textul cifrat se mai numeşte *criptogramă*.

Procesul retransformării criptogramei în textul original este numit *decriptare*.

Un *canal* este o cale pentru fluxul de informaţii.

Destinatarul primeşte printr-un canal textul criptat şi îl decriptează, ştiind metoda folosită pentru criptare, obţinând mesajul iniţial. În literatura de specialitate expeditorul de obicei este numit ***Alice*** iar destinatarul este numit ***Bob***. Deci, Alice şi Bob trebuie să stabilească în prealabil detaliile modalităţii de criptare şi de decriptare. Aşadar, *criptarea* este o metodă de camuflare a *textului clar* în aşa fel încât substanţa să nu sufere modificări semantice.

Criptarea se foloseşte pentru a fi siguri că informaţia este inaccesibilă oricărei persoane care nu deţine instrumentul necesar decriptării, chiar dacă oricine poate vedea datele în formă criptografică. Oricum nu va înţelege nimic, care să conducă spre descifrarea textului original.

Persoana care interceptează criptograma şi încearcă să obţină textul clar aplicând diverse metode, însă fără a avea cheia de decriptare, este numită *criptanalist*.

Sistemul care realizează operaţiile de criptare şi decriptare se numeşte *sistem de criptare* (sau *sistem criptografic*, sau *criptosistem*).

În criptografia modernă un ***sistem de criptare*** este definit ca o structură cu cinci componente (***P***, ***C***, ***K***, ***E***, ***D***):

- $\boldsymbol{P} = \{pt \;/\; pt \in \boldsymbol{T}^*\}$ – spaţiul (mulţimea) textelor în clar, scrise pentru un alfabet nevid ***T*** (în mod obişnuit $\boldsymbol{T}=\{0,1\}$);
- ***K*** – spaţiul (mulţimea) cheilor de criptare k, $k \in \boldsymbol{K}$;
- Familia funcţiilor de criptare dependentă de chei şi de un algoritm de criptare ***E***

  $\boldsymbol{E}_k : \boldsymbol{P} \rightarrow \boldsymbol{C}$, $\boldsymbol{E}_k = \{e_k \;/\; e_k(pt) = ct$ şi e_k este injectivă$\}$;
- Familia funcţiilor de decriptare dependentă de chei şi de un algoritm de decriptare ***D***

  $\boldsymbol{D}_k : \boldsymbol{C} \rightarrow \boldsymbol{P}$, $\boldsymbol{D}_k = \{ d_k \;/\; d_k(e_k(pt)) = pt$ pentru orice $pt \in \boldsymbol{P}\}$;
- ***C*** spaţiul (mulţimea) mesajelor cu text criptat unde:

  $\boldsymbol{C} = \{ct \;/$ există $k \in \boldsymbol{K}, a \in \boldsymbol{P}, ct = \boldsymbol{E}_k(a)\}$.

Schema aplicării unui sistem de criptare este prezentată în Figura 1.1.

Pentru ca un sistem de criptare să fie considerat bun, el trebuie să îndeplinească trei criterii (enunţate de Francis Bacon în sec. XVII):

1. Fiind date e_k şi $pt \in \boldsymbol{P}$ să fie uşor de calculat $e_k(pt)$.
2. Fiind date d_k şi $ct \in \boldsymbol{C}$ să fie uşor de determinat $d_k(ct)$.
3. Să fie imposibil de determinat pt din ct, fără a cunoaşte d_k.

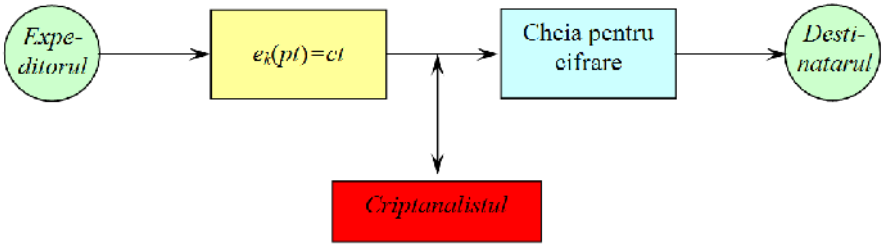


Figura 1.1. *Schema aplicării unui sistem de criptare*

Criteriile 1 şi 2 implică că pentru utilizatorii legali sistemul de criptare nu trebuie să fie prea complicat (se presupune că utilizatorii au un timp acceptabil pentru calcule). În criteriul 3 „imposibilitatea” e înlocuită în prezent cu „dificultatea de a calcula”. Se presupune că un interceptor de asemenea are acces la tehnica de calcul. Ultimul criteriu defineşte – sub o

formă vagă – ideea de ”securitate” a sistemului. La aceste criterii, Bacon adăuga şi o a patra regulă:

4. Textul criptat trebuie să fie un text banal, fără suspiciuni.

Această condiţie nu mai poate fi considerată importantă şi este utiliză astăzi doar de un subdomeniu strict al criptografiei, numit *steganografie* – ştiinţa despre transmiterea secretă a informaţiei prin păstrarea secretului a însuşi faptului transmiterii acestei informaţii.

Metodele de criptare pot fi divizate pe categorii în felul următor:

a) *În funcţie de tipul operaţiilor folosite*:
   - Bazate pe substituţii
   - Bazate pe transpuneri

b) *În funcţie de tipul de chei folosite*:
   - Sisteme Simetrice (single-key, secret-key, private-key)
   - Sisteme Asimetrice (two-key, public-key)

c) *Metoda prin care datele sunt procesate*:
   - Cu cifruri bloc
   - Cu cifruri fluide (flux, şir, “stream”)

Cifrurile clasice foloseau substituţia sau transpoziţia. Printre metodele moderne de criptare deosebim două direcţii principale: *sistemele cu cheie secretă* (sau *sisteme simetrice*) în care e_k este bijectivă şi *sisteme cu chei publice* (sau *sisteme asimetrice*) – sistemele în care e_k nu este bijectivă.

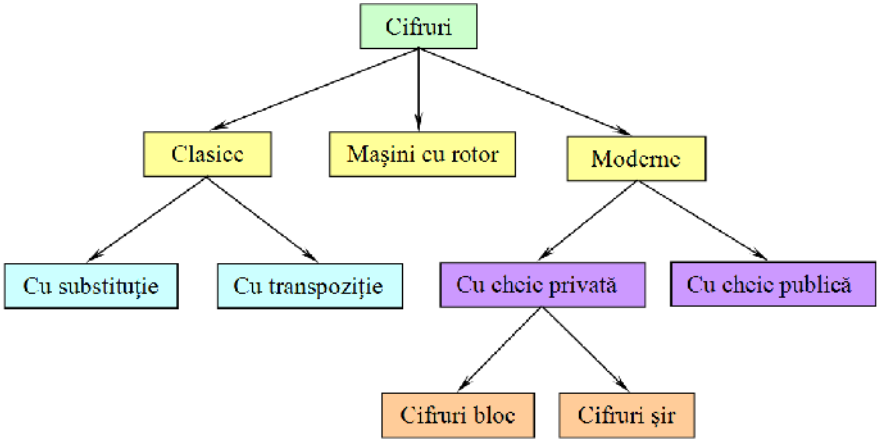


Figura 1.2. *Clasificarea cifrurilor*

Există două tipuri de sisteme simetrice: sisteme care se bazează pe algoritmi de *tip bloc* şi sisteme care se bazează algoritmi de *tip şir* (sau *flux*, în engleză *stream cipher*). Algoritmii de tip bloc acţionează asupra blocurilor de text clar şi text cifrat. Algoritmii de tip şir se aplică şirurilor de text clar şi text cifrat, la nivel de bit sau octet. În Figura 1.2 sunt prezentate tipurile de cifruri utilizate în trecut sau prezent.

Algoritmii moderni de tip bloc criptează mesajul în blocuri de 64 – 265 biţi. Pentru acesta se aplică o funcţie matematică între un bloc de biţi ai mesajului în clar şi cheie (care poate varia ca mărime), rezultând acelaşi număr de biţi pentru mesajul criptat. Funcţia de criptare este realizată astfel încât să îndeplinească următoarele cerinţe:

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

Cifrurile şir (sau cifruri *fluide*) la fel formează o clasă importantă de algoritmi de criptare. Ceea ce le caracterizează şi le diferenţiază faţă de cifrurile bloc este faptul că cifrurile şir procesează informaţia în unităţi oricât de mici, chiar bit cu bit, aplicând funcţia XOR între biţii cheii şi biţii de cifrat, iar funcţia de criptare se poate modifica în cursul criptării. Cifrurile şir sunt algoritmi cu memorie, în sensul că procesul de criptare nu depinde doar de cheie şi de textul clar, ci şi de starea curentă. În cazul în care probabilitatea erorilor de transmisie este mare, folosirea cifrurilor şir este avantajoasă deoarece au proprietatea de a nu propaga erorile. Ele se folosesc şi în cazurile în care datele trebuie procesate una câte una, datorită lipsei de spaţiu de memorie.

Cifruri *asimetrice* utilizeaza o pereche de chei: o *cheie publică* și o *cheie privată*. Un utilizator care deține o astfel de pereche își publică o cheie (cheia publică) astfel încat oricine dorește să o poata folosi pentru a îi transmite un mesaj criptat. Numai deținătorul cheii secrete (private) este cel care poate decripta mesajul astfel criptat.

Cele două chei sunt legate matematic, însă cheia privată nu poate fi obținută din cheia publică. În caz contrar, orcine ar putea decripta mesajele

destinate unui alt utilizator, fiindcă oricine are acces la cheia publică a acestuia. O analogie foarte potrivită pentru proces este folosirea cutiei poştale. Oricine poate pune în cutia poştală a cuiva un plic, dar la plic nu are acces decât posesorul cheii de la cutia poştală.

Cripografia asimetrică se mai numeşte criptografie cu chei publice şi e compusă din două mari ramuri:

- *Criptarea cu cheie publică* – un mesaj criptat cu o cheie publică nu poate fi decodificat decât folosind cheia privată corespunzătoare. Metoda este folosită pentru a asigura confidenţialitatea.
- *Semnături digitale* – un mesaj semnat cu cheia privată a emiţătorului poate fi verificat de către oricine, prin acces la cheia publică corespunzatoare, astfel asigurându-se autenticitatea mesajului.

Tema 2. Cifruri clasice. Cifruri de substituţie

Încă foarte demult, circa 4000 de ani în urmă, în oraşul Menet Khufu de pe malul Nilului un scrib cu experienţă a desenat ieroglife care relatau viaţa stăpânului său, devenind astfel cel care a pus bazele istoriei criptografiei. Sistemul său nu este un sistem al scrierii secrete în sens contemporan. Pentru aceasta el nu a folosit un cifru complet. Această înscriere, făcută circa în 1900 î.Hr. pe mormântul lui Khnumphotep, numai alocuri consta din simboluri ieroglifice neobişnuite în locul unora uzuale la acel moment. Marea lor parte se întâlneşte în ultimele douăzeci de coloane în care sunt enumerate monumentele create de către Khnumphotep în timpul serviciului său la faraonul Amenemhet II. Aceste înscrieri au fost făcute mai degrabă pentru a da o importanţă textului şi nu pentru a împiedica citirea lui. Astfel scribul nu a aplicat scrierea secretă dar, fără îndoială, a aplicat unul dintre elementele de bază ale criptării – transformarea deliberată a celor scrise.

Astfel, adăugarea la textele de acest fel a elementelor de secret a dat naştere criptografiei. Este adevărat, acest fapt semăna mai mult cu un joc, deoarece avea scopul de a amâna dezlegarea ghicitorii pentru un interval de timp scurt. Deci şi criptanaliza lui consta numai în rezolvarea problemei. Aşadar putem afirma că criptanaliza Egiptului antic, spre deosebire de cea contemporană, foarte serioasă, era numai o quasi-ştiinţă. Însă orice lucru măreţ are un început modest. Hieroglifele Egiptului antic conţineau, într-o formă departe de cea impecabilă, două dintre elementele de bază ale criptografiei – secretul şi transformarea textului. Astfel s-a născut criptologia.

În primii 3000 de ani dezvoltarea ei nu a fost una continuă. În unele locuri criptologia se năştea şi murea odată cu civilizaţia ce i-a dat naştere. În altele ea a rezistat pătrunzând în literatură pentru ca generaţiile următoare să poată urca spre nivele mai înalte ale criptologiei. Înaintarea spre aceste nivele era lentă şi anevoioasă. Mai multe erau pierdute decât păstrate. Cunoştinţele acumulate au căpătat amploare numai la începutul Renaşterii europene.

Criptografia clasică este criptografia dinaintea calculatorului, de unde şi denumirea de criptografie pre-computaţională. În criptografia clasică, algoritmii erau bazaţi pe caracter şi constau dintr-o serie de transformări elementare (substituţii, transpoziţii) ale caracterelor textului clar. Unii

algoritmi aplicau aceste transformări în mod repetat, îmbunătăţind în acest mod securitatea algoritmului. În criptografia modernă bazată pe calculator (criptografie computaţională), lucrurile s-au complicat, dar multe dintre ideile criptografiei clasice au rămas nemodificate.

Criptografia clasică se încadrează în clasa criptografiei cu chei simetrice.

Cifrul de substituţie (***substitution cipher***) este cifrul bloc la care fiecare caracter sau grup de caractere ale textului clar ***m*** este substituit cu un alt caracter sau grup de caractere în textul cifrat ***c***, descifrarea făcându-se prin aplicarea substituţiei inverse asupra textului cifrat. În criptografia clasică există patru tipuri de cifruri de substituţie. Deosebim cifruri cu substituţie monoalfabetică şi polialfabetică.

Cifruri de substituţie monoalfabetică (*monoalphabetic ciphers*) sunt cifrurile în care fiecare caracter al textului în clar ***m*** este înlocuit cu un caracter corespunzător în textul cifrat ***c***. Mai jos sunt prezentate câteva dintre cele mai cunoscute cifruri de substituţie monoalfabetică:

Cifrul lui *Cesar* (sau *Cezar*). În acest cifru fiecare literă a textului clar este înlocuită cu o nouă literă obţinută printr-o deplasare alfabetică. Cheia secretă ***k***, care este aceeaşi la criptare cât şi la decriptare, constă în numărul care indică deplasarea alfabetică, adică $\boldsymbol{k} \in \{1, 2, 3, \ldots, n-1\}$, unde n este lungimea alfabetului. Criptarea şi decriptarea mesajului pentru cifrul Cezar poate fi definită de formulele

$$\boldsymbol{c} = e_k(\boldsymbol{x}) = \boldsymbol{x} + \boldsymbol{k} \pmod{n},$$
$$\boldsymbol{m} = d_k(\boldsymbol{y}) = \boldsymbol{y} - \boldsymbol{k} \pmod{n},$$

unde ***x*** este reprezentarea numerică a caracterului respectiv al textului clar. Funcţia numită *Modulo* (a mod b) returnează restul împărţirii numărului întreg a la numărul întreg b. Această metodă de criptare este numită aşa după Iulius Cezar, care o folosea pentru a comunica cu generalii săi, folosind cheia $\boldsymbol{k} = 3$ (Tabelul 2.1). Pasul de criptare al cifrului lui Cezar este de obicei încorporat în scheme mai complexe precum *Cifrul Vigenère* (vezi mai jos).

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	0	1	2
A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
D	*E*	*F*	*G*	*H*	*I*	*J*	*K*	*L*	*M*	*N*	*O*	*P*	*Q*	*R*	*S*	*T*	*U*	*V*	*W*	*X*	*Y*	*Z*	*A*	*B*	*C*

Tabelul 2.1. *Exemplu pentru un cifru Cezar cu cheia* ***k***=3

De exemplu, pentru $\boldsymbol{k} = 3$ avem

$$e_k(\mathrm{S}) = 18 + 3 \pmod{26} = 21 = \mathrm{V}$$
$$d_k(\mathrm{V}) = 21 - 3 \pmod{26} = 18 = \mathrm{S}$$

Exemplu*:* Textul clar

„EXEMPLIFICARE CEZAR”,

prin aplicarea cheii $\boldsymbol{k} = 3$ se transformă în textul criptat

„*HAHPSOLILFDUH FHCDU*”.

Cifrul lui Cezar este foarte uşor de spart, deci este un cifru foarte slab. Astfel, un criptanalist poate obţine textul clar prin încercarea celor 25 de chei. Nu se ştie cât de util era cifrul Cezar în timpul când era folosit de către cel de la care îi provine numele, dar este probabil ca el să fie destul de sigur, atât timp cât numai câţiva dintre inamicii lui Cezar erau în stare să scrie şi să citească, dar mai ales să cunoască concepte de criptanaliză.

Cifrul *afin* este o generalizare a cifrului Cezar.

Cheia

$$\boldsymbol{k} = \{(a, b) \mid a, b \in \mathrm{Z}_{26} = \{0, 1, 2, \ldots, 25\}, \mathrm{cmmdc}(a, 26) = 1\},$$

iar funcţiile de criptare şi decriptare (pentru o cheie $\boldsymbol{k} = (a, b)$) sunt

$$e_k(\boldsymbol{x}) = a\boldsymbol{x} + b \pmod{26},$$
$$d_k(\boldsymbol{y}) = a^{-1}\boldsymbol{y} + a^{-1}(26 - b) \pmod{26}.$$

Condiţia ca a să fie prim cu 26 asigură existenţa lui a^{-1} în Z_{26}.

De exemplu, pentru $a = 7$, $b = 16$ funcţia de criptare este $e_k(\boldsymbol{x}) = 7\boldsymbol{x} + 16$, care poate fi reprezentată cu Tabelul 2.2:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
16	23	4	11	18	25	6	13	20	1	8	15	22	3	10	17	24	5	12	19	0	7	14	21	2	9
A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
Q	X	E	L	S	Z	G	N	U	B	I	P	W	D	K	R	Y	F	M	T	A	H	O	V	C	J

Tabelul 2.2. *Exemplu pentru cifrul afin cu cheia* $\boldsymbol{k} = (7, 16)$

Astfel, textul clar *LA MULTI ANI* se criptează în *PQ WAPTU QDU*.

Deoarece $7^{-1} = 15 \pmod{26}$, decriptarea se realizează matematic folosind funcţia

$$d_k(\boldsymbol{y}) = 15\boldsymbol{y} + 15(26-15) \pmod{26} = 15\boldsymbol{y} + 15 \cdot 11 \pmod{26} = 15\boldsymbol{y} + 9 \pmod{26}$$

(sau practic, inversând cele două linii ale tabelului de mai sus).

Condiţia $\mathrm{cmmdc}(a, 26) = 1$ asigură de asemenea injectivitatea funcţiei e_k. De exemplu, pentru $e_k(\boldsymbol{x}) = 10\boldsymbol{x} + 1$, *A* şi *N* se transformă ambele în *B*, iar *O* nu apare ca imagine în alfabetul substituţiei.

Orice cheie ***k*** a cifrului afin este determinată complet de valorile întregi (*a*, *b*) pentru care cmmdc(*a*, 26) = 1. Sunt posibile 12 valori pentru *a*: 1, 3, 5, 7, 9, 11, 15, 19, 21, 23, 25 şi 26 valori pentru *b*, care se iau independent de *a*, cu o singură excepţie a = 1, b = 0 (care se exclude deoarece nu conduce la nici o criptare). Aşadar mulţimea cheilor în acest caz este alcătuită din

$$12 \cdot 26 - 1 = 311$$

chei diferite, număr suficient de pentru atacul prin forţa brută.

Cifrul *Polibios*. Cifrul Cezar nu este cel mai vechi algoritm de criptare. Se pare că primul astfel de algoritm a fost utilizat de Polybios (istoric grec mort cu 30 ani înaintea naşterii lui Cezar). Iniţial acesta a fost doar un sistem maritim de semnalizare cu torţe iar ulterior i s-a dat o semnificaţie criptografică.

În cifrul Polibios pentru fiecare alfabet se construieşte un careu aparte de cifrare, de cele mai dese ori cu numărul de coloane şi linii egal (însă nu e o condiţie necesară). Dimensiunile careului depind de lungimea *n* a alfabetului. Pentru a crea careul se iau două numere întregi, produsul cărora e cel mai aproape de *n*. Liniile şi coloanele se numerotează. După aceasta literele alfabetului se înscriu în acest careu în ordinea apariţiei. Dacă nu sunt suficiente celule pentru literele alfabetului se pot înscrie într-o celulă 2 litere (de frecvenţa cât mai redusă).

Pentru alfabetul latin, putem avea careuri Polibios 5×5, după cum este reprezentat în Tabelul 2.3:

	a	b	c	d	e
a	A	B	C	D	E
b	F	G	H	I/J	K
c	L	M	N	O	P
d	Q	R	S	T	U
e	V	W	X	Y	Z

	1	2	3	4	5
1	A	B	C	D	E
2	F	G	H	I/J	K
3	L	M	N	O	P
4	Q	R	S	T	U
5	V	W	X	Y	Z

	a	b	c	d	e
a	A	B	C	D	E
b	F	G	H	I	J
c	K	L	M	N	O
d	P	R	S	T	U
e	V	W	X	Y	Z

Tabelul 2.3. *Exemple de tabele ale cifrului Polibios*

Observaţie: în al treilea careu a fost omisă litera Q care este una cu frecvenţă redusă.

În operaţia de criptare, fiecare caracter ***m*** va fi reprezentat printr-o pereche de litere (x, y), unde $x, y \in \{A, B, C, D, E\}$ (unde ABCDE este cheia cifrului) sau $x, y \in \{1, 2, 3, 4, 5\}$ (cheia este 12345) care dau linia, respectiv coloana pe care se află ***M***.

Astfel, textul clar VENI VIDI VICI este criptat în

55 15 33 24 55 24 14 24 55 24 13 24.

Deci sistemul de criptare Polybios este o substituţie monoalfabetică cu alfabetul

W ={AA, AB, AC, . . . , EE} sau W ={11, 12, 13, . . . , 55}

de 25 caractere.

Sunt diverse versiuni ale sistemului Polybios. Astfel, dacă se folosesc drept coordonate cifrele 1, 2, 3, 4, 5 în loc de A, B, C, D, E, sistemul a fost folosit în penitenciarele ruseşti şi de către prizonierii americani din Vietnam. Este foarte simplu de învăţat şi poate fi aplicat folosind diverse semne drept coordonate-chei (cifre, puncte, figuri, etc). Cifrul Polibios a fost utilizat de asemenea în cadrul altor sisteme de criptare, cum ar fi sistemul nihilist, cifrul ADFGVX (utilizat de armata germană în primul război mondial) sau sistemul Bifid, inventat de Dellastell în 1901.

Punctul slab al sistemelor de criptare monoalfabetice constă în frecvenţa de apariţie a caracterelor în text. Dacă un text criptat este suficient de lung şi se cunoaşte limba în care este scris textul clar, sistemul poate fi spart printr-un atac bazat pe frecvenţa apariţiei literelor într-o limbă.

Sunt construite diverse structuri de ordine relativ la frecvenţa apariţiei literelor în fiecare limbă europeană. De obicei, cu cât un text criptat este mai lung, cu atât frecvenţa literelor folosite se apropie de această ordonare generală. O comparare între cele două relaţii de ordine (cea a caracterelor din textul criptat şi cea a literelor din alfabetul limbii curente) conduce la realizarea câtorva corespondenţe (literă text clar – literă text criptat), ceea ce stabileşte în mod univoc cheia de criptare. Pentru sistemul Cezar este suficientă stabilirea unei singure perechi; pentru sistemul afin trebuiesc două perechi etc.

Pentru limba română frecvenţa literelor este prezentată în Tabelul 2.4 şi Figura 2.1.

A	Ă	Â	B	C	D	E	F	G	H	I	Î	J	K	L	M
9,95	*4,06*	*0,91*	*1,07*	*5,28*	*3,45*	*11,47*	*1,18*	*0,99*	*0,47*	*9,96*	*1,40*	*0,24*	*0,11*	*4,48*	*3,10*
N	O	P	Q	R	S	Ş	T	Ţ	U	V	W	X	Y	Z	
6,47	*4,07*	*3,18*	*0,00*	*6,82*	*4,40*	*1,55*	*6,04*	*1,00*	*6,20*	*1,23*	*0,03*	*0,11*	*0,07*	0,71	

Tabelul 2.4. *Frecvenţa literelor limbii române*

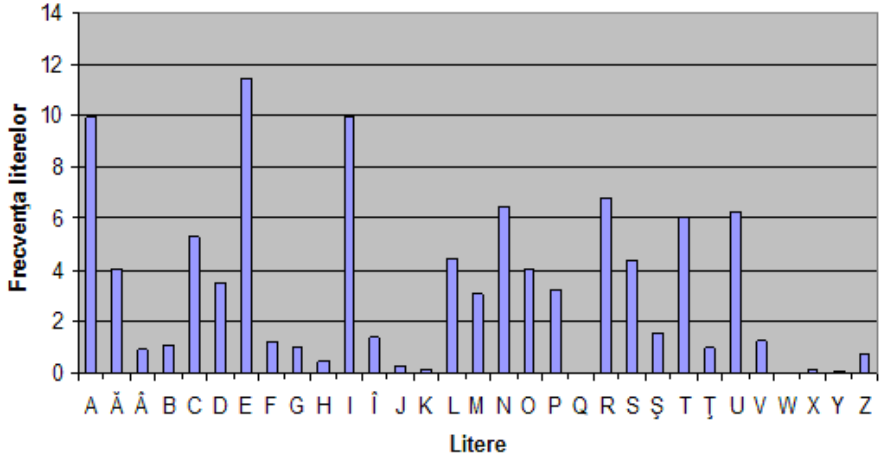


Figura 2.1. *Frecvenţa literelor limbii române*

Cifruri de substituţie polialfabetică (*polyalphabetic ciphers*).

Slăbiciunea cifrurilor monoalfabetice este dată de faptul că distribuţia lor de frecvenţă reflectă distribuţia alfabetului folosit. Un cifru este mai sigur din punct de vedere criptografic dacă prezintă o distribuţie cât mai regulată, care să nu ofere informaţii criptanalistului.

O cale de a aplatiza distribuţia este combinarea distribuţiilor ridicate cu cele scăzute. Daca *T* este criptat câteodată ca *a* şi altă dată ca *b*, şi dacă *X* este de asemenea câteodată criptat ca *a* şi altă dată ca *b*, frecvenţa ridicată a lui *T* se combină cu frecvenţa scăzută a lui *X* producând o distribuţie mai moderata pentru *a* şi pentru *b*.

Două distribuţii se pot combina prin folosirea a doua alfabete separate de criptare, primul pentru caracterele aflate pe poziţii pare în text clar, al doilea pentru caracterele aflate pe poziţii impare rezultând necesitatea de

a folosi alternativ doua tabele de translatare, de exemplu permutările $p_1(a) = (3 \cdot a) \bmod 26$ şi $p_2(a) = ((7 \cdot a) + 13) \bmod 26$.

Diferenţa dintre cifrurile polialfabetice şi cele monoalfabetice constă în faptul că substituţia unui caracter variază în text, în funcţie de diverşi parametri (poziţie, context etc.). Aceasta conduce bineînţeles la un număr mult mai mare de chei posibile. Se consideră că primul sistem de criptare polialfabetic a fost creat de Leon Battista în 1568. Unele aplicaţii actuale folosesc încă pentru anumite secţiuni astfel de sisteme de criptare.

Cifrul omofonic (homophonic ciphers) este un cifru de substituţie în care un caracter al alfabetului mesajului în clar (alfabet primar) poate să aibă mai multe reprezentări. Ideea utilizată în aceste cifruri este uniformizarea frecvenţelor de apariţie a caracterelor alfabetului textului cifrat (alfabet secundar), pentru a îngreuna atacurile criptanalitice. Astfel, litera *A* - cu cea mai mare frecvenţă de apariţie în alfabetul primar – poate fi înlocuită de exemplu cu *H*, # sau *m*.

Sistemul de criptare omofonic este un sistem intermediar între sistemele mono şi cele polialfabetice. Principalul lui scop este de a evita atacul prin frecvenţa de apariţie a caracterelor. Se presupune că a fost utilizat prima oară în 1401 de către ducele de Mantua. În cifrul omofonic fiecărui caracter $a \in \boldsymbol{m}$ i se asociază o mulţime $H(a) \subset \boldsymbol{c}$ astfel încât:

- $H(a) \cap H(b) = \emptyset \Leftrightarrow a \neq b$;
- Dacă *a* apare mai frecvent decât *b* în textele clare, atunci card $((H(a)) \geq \text{card}(H(\text{b}))$.

Criptarea unui caracter $x \in \boldsymbol{m}$ se face cu un element ales aleator din $H(a)$. Pentru decriptarea lui $y \in \boldsymbol{c}$ se caută o mulţime $H(x)$ astfel ca $y \in H(x)$.

Exemplu. Pentru limba engleză poate fi utilizat cifrul definit de Tabelul 2.5.

În primele două linii ale acestui tabel sunt aşezate literele alfabetului latin şi frecvenţele (rotunjite) ale acestora. În coloanele de sub litera *x* este situat $H(x)$. De exemplu

$$H(n) = \{18, 58, 59, 66, 71,91\}.$$

Pentru criptarea textului „ac” se poate folosi oricare din secvenţele 0948, 1248, 3348, 4748, 5348, 6748, 7848, 9248, 0981, 1281, 3381, 4781, 5381, 6781, 7881, 9281.

Deşi mai greu de spart decât cifrurile de substituţie simple (monoalfabetice), cifrul omofonic nu maschează total proprietăţile statistice ale textului clar. În cazul unui atac cu text clar cunoscut (vezi Tema 15), cifrul se sparge extrem de uşor. Atacul cu text cifrat este mai dificil, dar unui calculator îi va lua doar câteva secunde pentru a-l sparge.

a	**b**	**c**	**d**	**e**	**f**	**g**	**h**	**i**	**j**	**k**	**l**	**m**	**n**	**o**	**p**	**q**	**r**	**s**	**t**	**u**	**v**	**w**	**x**	**y**	**z**
8	2	3	4	13	2	2	6	7	1	1	4	3	7	8	2	1	6	6	9	3	1	2	1	2	1
09	48	13	01	14	10	06	23	32	15	04	26	22	18	00	38	94	29	11	17	08	34	60	28	21	02
12	81	41	03	16	31	25	39	70			37	27	58	05	95		35	19	20	61		89		52	
33		62	45	24			50	73			51		59	07			40	36	30	63					
47			79	44			56	83			84		66	54			42	76	43						
53				46			65	88					71	72			77	86	49						
67				55			68	93					91	90			80	96	69						
78				57										99					75						
92				64															85						
				74															97						
				82																					
				87																					
				98																					

Tabelul 2.5. *Exemplu de cifru omofonic pentru limba engleză*

Cifrurile bazate pe substituţie poligramică realizează substituirea unor blocuri de caractere (poligrame) din textul clar, distrugând astfel semnificaţia, atât de utilă în criptanaliză, a frecvenţelor diferitelor caractere. Vom considera un mesaj $\boldsymbol{m} = m_1 m_2 ... m_d m_{d+1} ...$ şi un cifru care prelucrează poligrame de lungime d. Criptograma rezultată este $\boldsymbol{c}=c_1 ... c_d c_{d+1} ... c_{d+d}$. Fiecare poligramă $m_{i,d+1} ... m_{i,d+d}$ va fi prelucrată în poligrama $c_{i,d+1} ... c_{i,d+d}$ prin funcţia de substitute f_j astfel:

$$c_{i,d+j} = f_j(m_{i,d+1} \ldots m_{i,d+d})$$

În cazul cifrării literelor singulare frecvenţa de apariţie a literelor în textul cifrat este egală cu frecvenţa de apariţie a literelor corespunzătoare din textul clar. Această invarianţă a frecvenţelor furnizează o cantitate de informaţie suficientă criptanalistului pentru spargerea cifrului. Pentru minimizarea informaţiei colaterale furnizate de frecvenţa de apariţie a literelor s-a utilizat cifrarea grupurilor de d litere (d-grame). În cazul când un grup de d litere este substituit printr-un alt grup de d litere, substituţia se

numeşte poligramică. Substituţia poligramică cea mai simplă se obţine pentru d=2 când digrama m_1m_2 din textul clar se substituie cu digrama c_1c_2 din textul cifrat.

Un exemplu clasic pentru substituţia diagramelor este cifrul lui *Playfair* (Tabelul 2.6).

P	L	A	Y	F
I	R	E	X	M
B	C	D	G	H
J	K	N	O	S
T	U	V	W	Z

Tabelul 2.6. *Exemplu de cifru Playfair*

Primele litere din pătrat reprezintă un cuvânt cheie ***k*** (literele care se repetă se scriu o singură dată, în acest exemplu cheia fiind ***k***=PLAYFAIR), după care pătratul se completează cu literele alfabetului, fără repetarea literelor. Cifrarea se executa după următoarele reguli:

- dacă m_1m_2 sunt dispuse în vârfurile opuse ale unui dreptunghi, atunci c_1c_2 sunt caracterele din celelalte vârfuri ale dreptunghiului, c_2 fiind în aceeaşi linie cu m_1. De exemplu AB devine PD, deci AB → PD;
- dacă m_1 şi m_2 se găsesc într-o linie, atunci c_1 şi c_2 se obţin printr-o deplasare ciclică spre dreapta a literelor m_1 şi m_2. De exemplu AF → YP iar XM → MI;
- dacă m_1 şi m_2 se află în aceeaşi coloană atunci c_1 şi c_2 se obţin printr-o deplasare ciclică a lui m_1, m_2 de sus în jos. De exemplu RC → CK iar LU → RL etc.;
- pentru separarea liniilor identice alăturate se introduc nişte caractere de separare care, de regula, au frecventa de apariţie redusă, cum sunt de exemplu literele X, Q în limba româna. În cazul în care numărul de caractere în textul clar este impar se procedează la fel. La descifrare aceste litere introduse se omit.

Descifrarea se executa după reguli asemănătoare cu cele de cifrare

Exemplu. Folosind exemplul de mai sus (***k*** = PLAYFAIR) textul clar „*VINE IARNA*” obţinem textul cifrat „TE VD EP KE YE”. Aici am introdus la sfârşitul mesajului litera X iar AX → YE. La descifrare după sensul mesajului se omite această literă.

Cifrul Playfair se folosea în scopuri tactice de către forţele militare britanice în timpul celui de-al doilea război al Burilor (1899-1902) dar şi în primul război mondial. La fel a fost utilizat de către australieni şi germani în timpul celui de-al doilea război mondial. El era utilizat deoarece era suficient de rapid în aplicare şi nu necesita nici un utilaj special. Scopul principal al utilizării lui era protecţia informaţiei importante (însă nu şi secrete) pe parcursul unei lupte. La momentul când criptanaliştii inamici spărgeau cifrul, informaţia deja nu mai era utilă pentru inamic.

Utilizarea cifrului Playfair în prezent nu are sens deoarece laptop-urile moderne pot sparge cu uşurinţă cifrul în câteva secunde. Primul algoritm de spargere pentru Playfair a fost descris în anul 1914 de către locotenentul Iosif O. Moubornom într-o broşură de 19 pagini.

Cifrul Vigenere. La fel ca cifrul Cezar, cifrul Vigenere deplasează literele, dar, spre deosebire de acesta nu se poate sparge uşor în 26 combinaţii. Cifrul Vigenere foloseşte o deplasare multiplă. Cheia nu este constituită de o singură deplasare, ci de mai multe. Cheia este constituită din câţiva întregi k_i, unde $0 \leq k_i \leq 25$.

Criptarea se face în felul următor:

$$c_i = m_i + k_i \pmod{26}.$$

Cheia poate fi, de exemplu, k = (21, 4, 2 19, 14, 17) şi ar provoca deplasarea primei litere cu 21, $c_1 = m_1 + 21 \pmod{26}$, a celei de a doua cu 4, $c_2 = m_2 + 4 \pmod{26}$, ş.a.m.d. până la sfârşitul cheii şi apoi de la început, din nou. Cheia este de obicei un cuvânt, pentru a fi mai uşor de memorat – cheia de mai sus corespunde cuvântului „vector”. Metoda cu deplasare multiplă oferă protecţie suplimentară din două motive:

- primul motiv este că ceilalţi nu cunosc lungimea cheii.
- cel de al doilea motiv este că numărul de soluţii posibile creşte; de exemplu, pentru lungimea cheii egală cu 5, numărul de combinaţii care ar fi necesare la căutarea exhaustivă ar fi $26^5 = 11\ 881\ 376$.

Cifrul Vigenere a fost spart însă folosind altceva decât forţa brută (vezi mai jos).

Decriptarea pentru cifrul Vigenere este asemănătoare criptării. Diferenţa constă în faptul că se scade cheia din textul cifrat,

$$m_i = c_i - k_i \pmod{26}.$$

Pentru simplificarea procesului de cifrare se poate utiliza următorul tabel, numit *Tabula Recta* (Tabelul 2.7). Aici toate cele 26 cifruri sunt

situate pe orizontală şi fiecărui cifru îi corespunde o anumită literă din cheie, reprezentată în colana din stânga tabelului. Alfabetul corespunzător literelor textului clar se află în prima linie de sus a tabelului. Procesul de cifrare este simplu – este necesar ca având litera ***x*** din cheie şi litera ***y*** din textul clar să găsim litera textului cifrat care se află la intersecţia liniei ***x*** şi coloanei ***y***.

	a	***b***	***c***	***d***	***e***	***f***	***g***	***h***	***i***	***j***	***k***	***l***	***m***	***n***	***o***	***p***	***q***	***R***	***s***	***t***	***u***	***v***	***w***	***x***	***y***	***z***
a	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
b	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
c	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
d	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
e	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
f	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
g	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
h	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
i	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
j	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
k	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
l	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
m	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
n	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
o	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
p	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
q	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
r	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
s	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
t	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
u	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
v	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
w	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
x	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	**V**	W
y	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
z	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y

Tabelul 2.7. *Tabula Recta pentru cifrul Vigenere*

Se poate de procedat şi în conformitate cu ecuaţiile ce definesc cifrul $c_i = m_i + k_i$ (mod 26) şi $m_i = c_i - k_i$ (mod 26), aşa cum este arătat în exemplul ce urmează.

Exemplu. De cifrat, utilizând cifrul Vigenere, mesajul „Per aspera ad astra" folosind cheia ***K***= SUPER.
Pentru a cifra sau descifra mai întâi facem corespondenţa următoare:

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25

Apoi alcătuim şi completăm tabelul:

Textul clar ***M***	P	E	R	A	S	P	E	R	A	A	D	A	S	T	R	A
Cheia ***K***	S	U	P	E	R	S	U	P	E	R	S	U	P	E	R	S
Textul clar ***M***	15	4	17	0	18	15	4	17	0	0	3	0	18	19	17	0
Cheia ***K***	18	20	15	4	17	18	20	15	4	17	18	20	15	4	17	18
M+***K*** (mod 26)	7	24	6	4	9	7	24	6	4	17	21	20	7	23	8	18
Textul cifrat ***C***	H	Y	G	E	J	H	Y	G	E	R	V	U	H	X	I	S

C= HYGEJHYGERVUHXIS.

Pentru decriptare procedăm la fel, cu excepţia $m_i = c_i - k_i$ (mod 26).
Apoi alcătuim şi completăm tabelul:

Textul cifrat ***C***	H	Y	G	E	J	H	Y	G	E	R	V	U	H	X	I	S
Cheia ***K***	S	U	P	E	R	S	U	P	E	R	S	U	P	E	R	S
Textul cifrat ***C***	7	24	6	4	9	7	24	6	4	17	21	20	7	23	8	18
Cheia ***K***	18	20	15	4	17	18	20	15	4	17	18	20	15	4	17	18
M–***K*** (mod 26)	15	4	17	0	18	15	4	17	0	0	3	0	18	19	17	0
Textul clar ***M***	P	E	R	A	S	P	E	R	A	A	D	A	S	T	R	A

M = PERASPERAADASTRA.

Criptanaliza sistemului Vigenere constă în următoarele: fie $c = c_0\, c_1 \ldots c_{n-1}$ textul criptat cu cheia $k = k_0\, k_1 \ldots k_{p-1}$; putem aranja acest text sub forma unei matrice cu p linii şi $\lceil n/p \rceil$ coloane, astfel

$$\begin{array}{cccc} c_0 & c_p & c_{2p} & \cdots \\ c_1 & c_{p+1} & c_{2p+1} & \cdots \\ \vdots & \vdots & \vdots & \vdots \\ c_{p-1} & c_{2p-1} & c_{3p-1} & \cdots \end{array}$$

Elementele de pe prima linie au fost criptate după formula

$$c_{pr} = a_{pr} + k_0 \pmod{26},\ k \geq 0,$$

adică cu un sistem Cezar (k_0 fiind o valoare fixată din $\boldsymbol{Z}_{26}$). În mod similar şi celelalte linii.

Deci, dacă s-ar cunoaşte lungimea *p* a cheii, problema s-ar reduce la criptanaliza a *p* texte criptate cu Cezar – sistem de criptare monoalfabetic. Sunt cunoscute două metode pentru aflarea lungimii cheii: testul lui Kasiski şi indexul de coincidenţă.

Prima metodă constă în studiul textului criptat şi aflarea de perechi de segmente de cel puţin 3 caractere identice (această lungime este propusă de Kasiski). Pentru fiecare astfel de pereche, se determină distanta dintre segmente. După ce s-au găsit mai multe astfel de distanţe, valoarea lui *p* va fi cel mai mare divizor comun al lor (sau – eventual un divizor al acestuia).

A doua metodă de aflare a lungimii cheii de criptare într-un sistem Vigenere se bazează pe un concept definit în 1920 de Wolfe Friedman - *indexul de coincidenţă*. Dacă $c = c_1c_2...c_n$ este o secvenţă de *n* caractere alfabetice, probabilitatea ca două caractere din *c*, alese aleator, să fie identice se numeşte "*index de coincidenţă*" $I_c(x)$ al lui *c*.

Tema 3. Cifruri clasice. Cifrul de transpoziţii

Spre deosebire de cifrurile cu substituţie, care păstrează ordinea literelor din textul sursă dar le transformă, cifrurile cu transpoziţie (*transposition ciphers*) reordonează literele, fără a le „deghiza”.

Criptarea prin metoda transpoziţiei este o tehnică mai eficientă decât criptarea prin substituţie, dar are, la rândul ei, o mulţime de dezavantaje. Textul criptat prin metoda transpoziţiei păstrează toate caracterele textului iniţial, dar în altă ordine obţinută prin aplicarea algoritmului ce va fi prezentat în continuare.

Criptarea prin transpoziţie constă în scrierea textului iniţial din care s-au eliminat spaţiile şi semnele de punctuaţie într-o matrice de dimensiune $M \times N$ şi interschimbarea anumitor linii (sau coloane) între ele. Textul criptat se obţine prin scrierea caracterelor din noua matrice de pe fiecare coloană în parte, începând cu colţul din stânga-sus. Dacă lungimea textului iniţial este mai mică decât numărul de elemente ce pot fi scrise în matrice, atunci textul se completează cu elemente aleatoare, până ajunge la dimensiunea $M \cdot N$.

Pentru textul „*Misiunea a fost îndeplinită*”, care are lungimea de 24 de caractere, se pot alege mai multe matrice de dimensiune $M \times N$, o posibilitate ar fi ca matricea să aibă 4 linii şi 6 coloane, dar pentru ca textul să fie mai greu de decodificat trebuie să conţină şi caractere alese aleator, sau într-un mod mai inteligent, care să îngreuneze munca celui care doreşte să afle conţinutul secret din mesajul criptat. Fie am ales o matrice care are 5 linii şi 6 coloane. Textului iniţial i se adaugă 6 caractere aleatoare şi se obţine textul *Misiun eaafos tîndep linită xyztwu* care se scrie în matricea din partea stângă, aşa cum e arătat în Tabelul 3.1:

	1	2	3	4	5	6
1	M	i	s	i	u	n
2	e	a	a	f	o	s
3	t	î	n	d	e	p
4	l	i	n	i	t	ă
5	x	y	z	t	w	u

	1	2	3	4	5	6
5	x	y	z	t	w	u
3	t	î	n	d	e	p
4	l	i	n	i	t	ă
1	M	i	s	i	u	n
2	e	a	a	f	o	s

Tabelul 3.1. *Exemplu de cifru cu transpoziţie*

Prin scrierea liniilor 1, 2, 3, 4, 5 în ordinea 5, 3, 4, 1, 2, se obţine matricea din partea dreaptă. Textul criptat care se obţine este: *xtlMe yîiia znnsa tdiif wetuo upăns*.

Transpoziţie cu cheie. Pentru ca procesul de decriptare să fie mai simplu şi să nu mai fie nevoie de ordinea în care au fost puse liniile din matricea creată, se foloseşte o versiune a criptării prin transpoziţie care se bazează pe o cheie.

Pentru a cripta un text folosind o cheie şi metoda transpoziţiei, se alege o cheie ale cărei litere determină ordinea în care se vor scrie coloanele din matricea aleasă. Pentru a afla ordinea în care vor fi scrise coloanele din textul iniţial, se ordonează alfabetic literele din cheie, şi fiecărei litere i se asociază numărul de ordine din şirul ordonat.

Lungimea cheii trebuie să fie egală cu numărul de coloane din matrice.

Considerăm textul anterior, scris într-o matrice de dimensiuni 5×6, şi cheia „*vultur*”. Literele din cheie se ordonează alfabetic şi se obţine şirul: *l, r, t, u, u, v*. Indicele 1 este asociat cu litera *l*, indicele 2 cu litera r, indicele 3 cu litera *t*, indicele 4 cu prima literă *u* din cheie, indicele 5 cu a doua literă *u* din cheie, iar indicele 6 este asociat cu litera *v*. Pentru a scrie coloanele, pentru fiecare indice *i* din şirul ordonat se caută indicele *j*, care reprezintă poziţia literei cu indicele *i*, din cheie şi se scrie coloana *j*, astfel:

	v	**u**	**l**	**t**	**u**	**r**
	6	**4**	**1**	**3**	**5**	**2**
1	M	i	s	i	u	n
2	e	a	a	f	o	s
3	t	î	n	d	e	p
4	l	i	n	i	t	ă
5	x	y	z	t	w	u

	1	**2**	**3**	**4**	**5**	**6**
5	s	n	i	i	u	M
3	a	s	f	a	o	e
4	n	p	d	î	e	t
1	n	ă	i	i	t	l
2	z	u	t	y	w	x

Tabelul 3.2. *Exemplu de transpoziţie cu cheie*

Textul cifrat care se obţine în final este:

sannz nspău ifdit iaîiy uoetw Metlx.

Pentru a decripta un mesaj criptat cu această metodă, criptograma se scrie în matrice pe coloane, începând cu colţul stânga-sus, şi apoi se realizează operaţia inversă, adică pentru fiecare indice *j* al literelor din cheie, se caută indicele *i* asociat literei din şirul sortat şi se scrie coloana cu

indicele *i*. Din noua matrice astfel obţinută se scriu literele de pe fiecare linie, în ordine.

O tehnică cunoscută şi foarte practică de transmitere a mesajelor folosind metoda transpoziţiei constă în înfăşurarea unei panglici în jurul unui băţ. Mesajul se scrie pe panglică, de-a lungul băţului, de la capătul superior spre capătul inferior, pe coloane şi apoi se trimite la destinaţie numai panglica, care ulterior s-a desfăcut de pe băţ. La destinaţie se înfăşoară panglica pe un băţ având aceeaşi dimensiune cu cel care a ajutat la scrierea textului şi se citeşte textul pe coloane.

Să analizăm un exemplu de text mai voluminos a transpoziţiei cu cheie.

Exemplu: De efectuat criptarea textului clar ***m*** =„*acestcursîsipropune săprezintefacilitătiledecomunicareoferitedereteleledecalculatoare*”, utilizând cheia „*PRECIS*”.

P	**R**	**E**	**C**	**I**	**S**
4	**5**	**2**	**1**	**3**	**6**
a	c	e	s	t	c
u	r	s	î	s	i
p	r	o	p	u	n
e	s	ă	p	r	e
z	i	n	t	e	f
a	c	i	l	i	t
ă	t	i	l	e	d
e	c	o	m	u	n
i	c	a	r	e	o
f	e	r	i	t	e
d	e	r	e	t	e
l	e	l	e	d	e
c	a	l	c	u	l
a	t	o	a	r	e

Tabelul 3.3. *Exemplu de cifru cu transpoziţie*

Pentru criptare (dar ulterior şi pentru decriptare) completăm tabelul de criptare (Tabelul 3.3) prin aşezarea textului clar pe linii. Citirea rezultatului pe coloane în conformitate cu cheia (în ordine alfabetică) va genera textul cifrat:

c =„*sîpptllmrieecaesoăniioarrllotsureieuettduraupezaăeifdlcacrrsictcceeeatcineftdnoeeele*”.

Spargerea unui cifru cu transpoziţie începe cu verificarea dacă acesta este într-adevăr de acest tip prin calcularea frecventelor literelor şi compararea acestora cu statisticile cunoscute. Dacă aceste valori coincid, se deduce că fiecare literă este „ea însăşi”, deci este vorba de un cifru cu transpoziţie.

Următorul pas este emiterea unei presupuneri în legătură cu numărul de coloane. Acesta se poate deduce pe baza unui cuvânt sau expresii ghicite ca făcând parte din text. Considerând sintagma „săprezinte”, cu grupurile de litere (luate pe coloane) „si”, „ăn”, „pt”, „re”, se poate deduce numărul de litere care le separă, deci numărul de coloane. Notăm în continuare cu *m* acest număr de coloane.

Pentru a descoperi modul de ordonare a coloanelor, dacă *m* este mic, se pot considera toate posibilităţile de grupare a câte două coloane (în număr de $m\ (m-1)$). Se verifică dacă ele formează împreună un text corect numărând frecventele literelor şi comparându-le cu cele statistice. Perechea cu cea mai bună potrivire se consideră corect poziţionată. Apoi se încearcă, după acelaşi principiu, determinarea coloanei succesoare perechii din coloanele rămase iar apoi - a coloanei predecesoare. În urma acestor operaţii, există şanse mari ca textul să devină recognoscibil.

Unele proceduri de criptare acceptă blocuri de lungime fixă la intrare şi generează tot un bloc de lungime fixă. Aceste cifruri pot fi descrise complet prin lista care defineşte ordinea în care caracterele vor fi trimise la ieşire (şirul poziţiilor din textul de intrare pentru fiecare caracter din succesiunea generată).

De la apariţia cifrurilor cu substituţie şi a celor cu transpoziţie anii au trecut şi tehnicile de criptare au evoluat foarte mult.

Problema construirii unui cifru imposibil de spart a preocupat îndelung pe criptanalişti; ei au dat o rezolvare teoretică simplă încă de acum câteva decenii dar metoda nu s-a dovedit fiabilă din punct de vedere practic, după cum se va vedea în continuare.

Tehnica propusă pentru un cifru perfect presupune alegerea unui şir aleator de biţi pe post de cheie şi aducerea textului sursă în forma unei succesiuni de biţi prin înlocuirea fiecărui caracter cu codul său ASCII. Apoi se aplică o operaţie logică - de tip SAU exclusiv (operaţia inversă echivalentei: 0 xor 0 = 0, 0 xor 1 = 1, 1 xor 0 = 1, 1 xor 1 = 0) - între cele două şiruri de biţi. Textul cifrat rezultat nu poate fi spart pentru că nu există indicii asupra textului sursă şi nici textul cifrat nu oferă criptanalistului informaţii. Pentru un eşantion de text cifrat suficient de mare, orice literă sau grup de litere (diftong, triftong) va apărea la fel de des.

Acest procedeu este cunoscut sub numele de metoda cheilor acoperitoare. Deşi este perfectă din punct de vedere teoretic, metoda are, din păcate, câteva dezavantaje practice:

- cheia nu poate fi memorată, astfel încât transmiţătorul şi receptorul să poarte câte o copie scrisă a ei fiindcă în caz că ar fi „capturaţi”, adversarul ar obţine cheia;
- cantitatea totală de date care poate fi transmisă este determinată de dimensiunea cheii disponibile;
- o nesincronizare a transmiţătorului şi receptorului care generează o pierdere sau o inserare de caractere poate compromite întreaga transmisie fiindcă toate datele ulterioare incidentului vor apărea ca eronate.

Tema 4. Maşini rotor

Sistemele de criptare pot fi aduse la un grad mai mare de securitate dacă se folosesc mijloace mecanice de criptare. Astfel de mecanisme special construite vor uşura operaţiile de criptare/decriptare şi în acelaşi timp vor fi capabile să creeze un număr mult mai mare de chei posibile. Primele astfel de mecanisme au apărut încă în antichitate.

În secolul V î.e.n. pentru criptarea datelor se folosea un baston, numit *Schitala*, în jurul căruia se înfăşura spiră lângă spiră o panglică foarte îngustă de piele, papirus sau pergament pe care, pe generatoare se scriau literele mesajelor. După ce textul era scris panglica se desfăşura, mesajul devenea indescifrabil, deoarece literele erau dezasamblate. Mesajul se putea descifra numai de o persoană care dispunea de un baston de grosime şi lungime identice cu bastonul iniţial pe care se înfăşura din nou panglica primită de receptor. Astfel Schitala realiza o transpoziţie, aceasta fiind o primă formă a acestei metode de criptare. Conform istoricilor greci, acest mod de comunicare era folosit de spartani în timpul campaniilor militare. El avea avantajul de a fi rapid şi nu genera erori de transmitere. Dezavantajul însă era acela că putea fi uşor de spart.

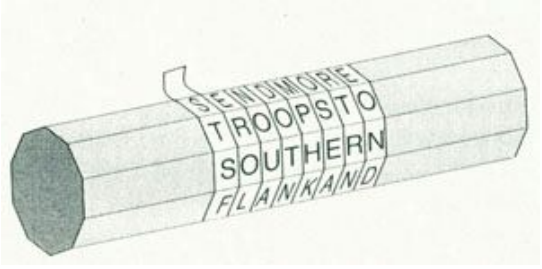


Figura 4.1. *Schitala*

Leon Battista Alberti (14.02.1404 – 25.04.1472) – scriitor, arhitect, pictor, sculptor, matematician, criptograf, filozof italian şi umanist al Renaşterii a inventat „*Criptograful lui Alberti*" (Figura 4.2), care era alcătuit din două discuri concentrice cu diametre diferite, suprapuse. Fiecare disc era împărţit în 24 sectoare pe care erau înscrise litere şi cifre. Pe discul exterior, care rămânea static, erau scrise 20 de litere ale alfabetului italian (alfabetul italian nu avea literele *H, J, K, W, X, Y*) în

ordinea lor firească, iar apoi cifrele 1, 2, 3, 4. Pe discul interior care se rotea, erau scrise 23 de litere ale alfabetului latin (fără *J, K, Y*) şi conjuncţia *ET*. Ordinea lor era arbitrară. Pentru cifrare se stabilea o cheie, de exemplu *G=a*. Aceasta însemna că pentru cifrare litera *a* de pe discul mic se aşeza în dreptul literei *G* de pe discul mare şi apoi începea cifrarea. Alberti recomanda schimbarea cheii după un număr de cuvinte.

Criptograful lui Alberti a fost perfecţionat de către Silvester, Argenti şi alţii, constituind un element de bază pentru criptografele de tip disc apărute ulterior. Silvester Porta a împărţit discurile în 26 sectoare (Figura 4.2) utilizând astfel toate cele 26 litere ale alfabetului latin (nu numai italian), criptograful său realizând astfel o substituţie simplă complet literală.

Criptograful lui Alberti avea două particularităţi care fac ca invenţia sa fie un mare eveniment în criptografie. În primul rând acest mecanism nu era altceva decât un algoritm de criptare polialfabetică. În rândul al doilea discul respectiv permitea utilizarea aşa numitelor coduri cu recifrare, care au apărut abia la sfârşitul secolului XIX, adică peste patru secole după invenţia lui Alberti. În acest scop pe discul exterior erau scrise cifrele 1, 2, 3, 4. Alberti a compus un cod care consta din 336 grupuri de coduri numerotate de la 11 la 4444. Fiecărui cod îi corespundea o oarecare frază terminată. Când fraza se întâlnea în mesaj ea se înlocuia cu codul respectiv, iar cu ajutorul discului cifrele erau criptate ca nişte semne ordinare ale mesajului, fiind transformate în litere.

Leon Alberti poate fi considerat un criptogarf ilustru şi din motivul că este autorul primei lucrări de criptologie din Europa („*De cifris*") publicată în 1946. În această lucrare erau prezentate exemple de versiuni posibile de cifrare dar şi se argumenta necesitatea aplicării criptografie în practică ca un instrument ieftin şi sigur de protecţie a informaţiei.

Ideea de maşină de criptare apare clar prima dată la Thomas Jefferson, primul secretar de Stat al Statelor Unite (preşedinte era George Washington), care a inventat un aparat de criptat numit roată de criptare, folosit pentru securitatea corespondenţei cu aliaţii – în special cei francezi.

Un cilindru Jefferson (Figura 4.3) este format din n discuri de dimensiuni egale (iniţial $n = 26$ sau $n = 36$) aşezate pe un ax. Discurile se pot roti independent pe ax, iar pe muchia fiecăruia sunt înscrise cele 26 litere ale alfabetului, într-o ordine aleatoare (dar diferită pentru fiecare disc).

Criptograful lui Alberti

Criptograful lui Silvester

Criptograful lui Silvester

Figura 4.2.

La criptare, textul clar se împarte în blocuri de *n* caractere. Fiecare astfel de bloc se scrie pe o linie (generatoare) a cilindrului, rotind corespunzător fiecare disc pentru a aduce pe linie caracterul căutat. Oricare din celelalte 25 linii va constitui blocul de text criptat.

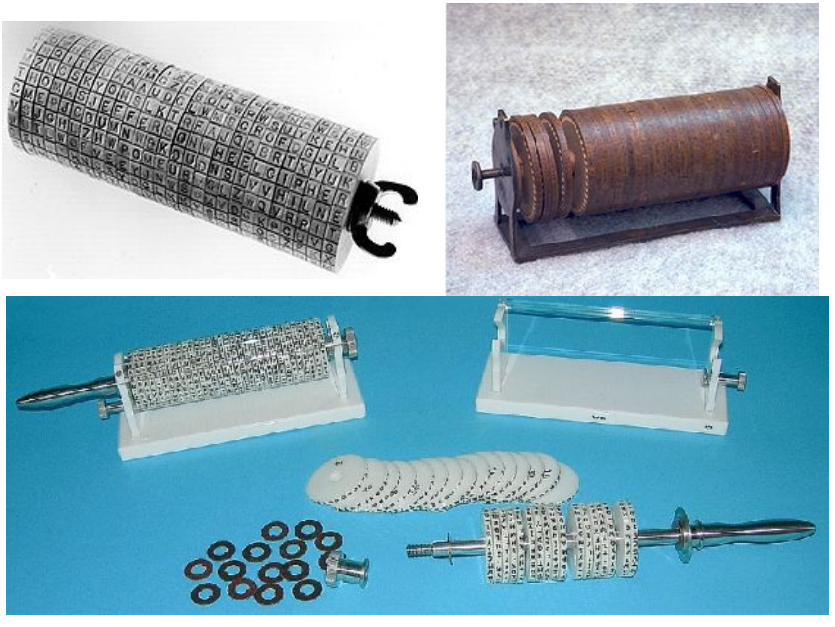


Figura 4.3. *Cilindre Jefferson*

Pentru decriptare este necesar un cilindru identic, în care se scrie pe o linie textul criptat (de *n* caractere) şi apoi se caută printre celelalte 25 linii un text cu semnificaţie semantică. Probabilitatea de a avea un singur astfel de text creşte cu numărul de discuri din cilindru.

O mică diferenţă apare dacă textul clar nu are nici o semnificaţie semantică (s-a folosit o dublă criptare). Atunci trebuie convenită dinainte o anumită distanţă de criptare *s* ($1 \le s \le 25$). Thomas Jefferson a folosit acest aparat în perioada 1790 − 1802, după care se pare că ideea s-a pierdut. Devenit preşedinte, Jefferson a fost atras de sistemul Vigenere, pe care îl consideră mai sigur şi-l recomandă secretarului său de stat James Madison ca înlocuitor al sistemului pe care îl inventase anterior.

Ordinea discurilor poate fi de asemenea schimbată. De exemplu, un cilindru cu $n = 20$ discuri poate realiza $20! = 2\,432\,902\,008\,176\,640\,000$ texte criptate diferite pentru acelaşi text clar. Cilindrul Jefferson realizează o substituţie polialfabetică de perioadă *n*. Dacă ar fi privit ca un sistem de criptare Vigenere, lungimea cheii este enormă (de multe ori n^n, în funcţie de modalităţile de aranjare a alfabetelor pe discuri). Cilindrul Jefferson a fost reinventat ulterior de mai multe ori, cea mai celebră fiind se pare maşina de criptat „*Enigma*”.

O **maşină rotor** (rotor machine, Figura 4.4) are o tastatură şi o serie de rotoare ce permit implementarea unei versiuni a cifrului Vigénère. Fiecare rotor face o permutare arbitrară a alfabetului, are 26 de poziţii şi realizează o substituţie simplă. Deoarece rotoarele se mişcă cu viteze de rotaţie diferite, perioada unei maşini cu *n* rotoare este $n \cdot 26!$.

Aplicarea practică a acestor maşini a început numai la începutul secolului XX. Una dintre primele maşini cu rotor a fost maşina germană „*Enigma*”, elaborată în anul 1917 de către Eduard Hebern şi perfectată mai târziu de mai multe ori. Din punct de vedere comercial ea a fost disponibilă pe piaţă încă din anl 1920, însă importanţa ei a fost dată de utilizarea maşinii de către diverse guverne, în mod special de către Germania nazistă înainte şi în timpul celui de-al doilea război mondial.

Dintre toate dispozitivele criptografice create de-a lungul timpului maşina Enigma a fost un echipament mai special din 2 puncte de vedere:

- criptografic;
- istoric.

Figura 4.4. *Modelul militar german numit Wehrmacht Enigma*

Importanţa din punct de vedere criptografic este dată de faptul că echipe de criptanalişti (matematicieni la origine) de toate naţionalităţile, în efort combinat, au încercat pe de o parte perfecţionarea maşinii, pe de altă parte spargerea cifrurilor. Printre cei care au participat la spargerea cifrului au făcut parte şi polonezul Rajewski şi britanicul Turing (inventatorul maşinilor Turing).

Importanţa istorică rezidă din rolul mare jucat de aceste maşini în timpul celui de-al doilea război mondial, mai precis faptul că descifrarea de către aliaţi a codului (nume de proiect ULTRA) a dus, după unii istorici, la scurtarea războiului cu aproximativ un an.

Construcţia maşinii. Maşina Enigma era o combinaţie de părţi mecanice şi electrice. Principalele ei componente erau, după cum urmează:

- *Tastatura* (Key board): o tastatură obişnuită similară cu cea pentru maşinile de scris.
- *Placa cu lămpi* (Lamp board): asemănătoare unei tastaturi cu lămpi în loc de taste. Pe lămpi erau tipărite literele alfabetului ce deveneau vizibile prin aprinderea lămpii corespunzătoare.
- *Placa cu comutatoare* (Switch board): mufe (prize) câte una pentru fiecare literă, ce se conectau prin fire în 6 perechi (Această componentă fusese adăugată de germani pentru a creşte securitatea maşinii).
- *Trei roţi* (Rotating drums): se mai numeau *rotoare* (*roţi detaşabile*) fiecare dintre ele având câte un set de 26 de contacte, câte unul pentru fiecare literă a alfabetului (Figura 4.5).
- *Roata reflectoare* (Reflecting drum): roată fixă identică cu celelalte 3, având un set de 26 de contacte grupate în perechi.

- *Cabluri* (Wiring): asigurau conexiunile între taste şi lămpi precum şi între lămpi şi primul rotor, între primul rotor şi al doilea, al doilea şi al treilea, al treilea si roata reflectoare.
- *Baterie* (Battery): pentru alimentarea circuitelor electrice.

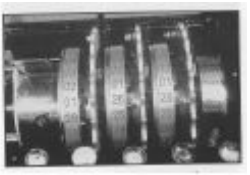


Figura 4.5. *Seturi rotor*

Principiul de funcţionare al maşinii Enigma se prezenta conform schemei din Figura 4.6:

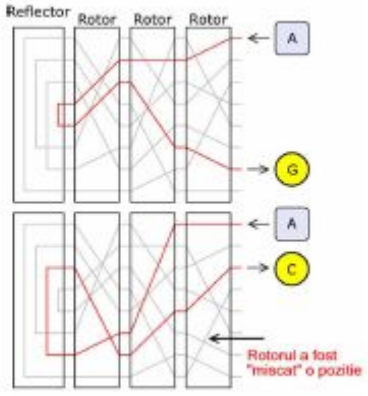


Figura 4.6. *Principiul de funcţionare al maşinii* (*Enigma*)

Prin apăsarea tastei ”A” curentul era trecut prin setul de rotoare până la reflector de unde se ”întorcea” înapoi aprinzându-se becul ”G”. Litera ”A” se criptează diferit (”G” si ”C”) doar printr-o simplă rotire a primului rotor care face ca semnalul să circule pe o rută complet diferită.

Pentru ***operarea maşinii*** în primul rând toţi operatorii aveau maşini identice (pentru asigurarea inter-operabilităţii). Iniţierea criptării unui mesaj se făcea în 2 paşi:

- *Pasul* 1: setarea maşinii – operaţie ce consta în fixarea ordinei şi poziţiei fiecărui rotor precum şi alegerea celor 6 perechi de conectori prin placa cu comutatoare (switch board).

- *Pasul 2*: scrierea propriu-zisă a mesajului – pentru criptarea mesajului operatorul apăsa pe tasta corespunzătoare primei litere din textul necodat (să zicem ”N”). În acest moment se aprindea o lampă (să zicem ”T”) corespunzătoare codificării. Repetând şi pentru celelalte litere, rezulta textul codat.

Trebuie de menţionat că toate setările din *Pasul* 1 erau înscrise în manuale de operare (code books), setări ce se schimbau de regulă zilnic. Fiecare operator avea câte un exemplar. De fapt, aceste setări constituiau cheia criptosistemului Enigma. Un atribut extrem de important al maşinii *Enigma* era că cheile de cifrare şi cele de decifrare erau aceleaşi. Cu alte cuvinte dacă la ”transmitere” ”N” se transforma în ”T”, la ”destinaţie ”T” se transforma în ”N” (folosind bine-nţeles aceleaşi setări ale maşinii).

Utilizarea intensivă colaborată cu posibilitatea transmiterii informaţiei folosind aceleaşi *day key* la care se adăugau intensele activităţi de contraspionaj i-au condus pe germani la teama că maşina ar putea fi compromisă. Efectul, a fost introducerea unui *protocol*. Acesta spunea: „*fiecare operator va transmite suplimentar, înaintea mesajului propriu zis, o cheie a mesajului* (*message key*)”. Aceste chei erau cuvinte (nu neapărat cu sens) formate din 3 litere alese în mod aleator de operatorul maşinii. Cu alte cuvinte, operatorul trebuia să seteze maşina conform instrucţiunilor zilnice din manualul de operare (*code book*), după care trimitea cheia din cele trei litere alese aleator. În aşa fel maşina se seta într-un mod complet aleator. Condiţiile radio proaste, lucrul sub presiune, precum şi alte condiţii de lucru nefavorabile puteau conduce la transmiterea (sau recepţionarea) greşită (alterată) a cheii, fapt ce ar fi făcut inutilă transmiterea mesajului propriu-zis (evident, datorită faptului că maşinile de la transmiţător şi cea de la receptor ar fi fost setate diferit). Pentru a minimiza astfel de incidente, operatorilor li s-a cerut să transmită cheia de 2 ori.

De exmplu

cheia: *the*
se transmitea: *hothot*
se recepţiona: *dugraz*

Însă în mod ironic, ceea ce se dorea o măsură de securitate în plus, de fapt a compromis maşina.

Matematic vorbind, mulţimea cheilor posibile era atât de mare încât nici nu se punea problema ”atacării” maşinii, cel puţin nu la aceea vreme,

prin metoda exhaustivă („*brute-force*”). Enigma a fost elaborată astfel încât securitatea să fie păstrată chiar dacă inamicul cunoaşte schemele rotoarelor, cu toate că în practică setările erau secrete. Cu o schemă secretă de setare cantitatea totală a configurărilor posibile era de ordinul 10^{114} (circa 380 biţi) iar dacă schema şi alte setări operaţionale erau cunoscute acest număr se reducea la 10^{23} (76 biţi). Germanii credeau că maşina Enigma este una infailibilă datorită imensităţii setărilor posibile ce i se puteau aplica. Era ireal să începi măcar să alegi o configurare posibilă.

Din punct matematic de vedere transformarea Enigmei pentru fiecare literă este rezultatul matematic a permutărilor. Pentru un aparat cu trei rotoare fie *P* transformarea pe tabela de prize, *U* - reflectorul, şi *L*, *M*, *R* - acţiunea rotorului din stânga, din mijloc, dreapta respectiv. Atunci criptarea *E* poate fi notată cu:

$$E = PRMLUL^{-1}M^{-1}R^{-1}P^{-1}$$

După fiecare apăsare de tastă rotoarele se rotesc, schimbând transformarea. De exemplu dacă rotorul de dreapta *R* e rotit cu *i* poziţii, transformarea devine: $\rho^i R \rho^{-i}$, unde ρ este permutarea ciclică. Similar, rotorul din mijloc şi cel din stânga pot fi reprezentate ca *j* şi *k* rotaţii respectiv a lui *M* şi *L*. Funcţia de criptare poate fi descrisă astfel:

$$E = P(\rho^i R \rho^{-i})(\rho^j M \rho^{-j})(\rho^k L \rho^{-k})U(\rho^k L^{-1} \rho^{-k})(\rho^j M^{-1} \rho^{-j})(\rho^i R^{-1} \rho^{-i})P^{-1}$$

Pentru elucidarea funcţionării maşinii Enigma este sugestivă simularea (în flash) de la http://enigmaco.de/enigma/enigma.swf (Figura 4.7).

Primele spargeri ale maşinii Enigma au avut loc la începutul anilor 30 de către matematicienii polonezi *Alicen Rejewski*, *Jerzy Rozycki* şi *Henryk Zygalsk*. Cu noroc şi intuiţie Rejewski şi echipa lui au reuşit să compromită maşina, totul fiind posibil nu datorită vreunei ”scăpări” în proiectarea maşinii ci deciziei nemţilor de a transmite repetitiv (de 2 ori) cheia.

Ulterior Enigma a fost perfecţionată, spargerea ei devenind practic imposibilă pentru acele timpuri. Un aport considerabil în direcţia spargerii acestei maşini a avut Alan Turing, care proiectase o maşină electromecanică (denumită „*Bombe*” după modelul original polonez) ce putea ajuta la spargerea maşinii Enigma mai rapid decât „bomba” din 1932 a lui *Rejewski*, din care s-a şi inspirat. „Bombe” (Figura 4.8), cu o îmbunătăţire sugerată de matematicianul Gordon Welchman, a devenit una din principalele unelte automate utilizate pentru a ataca traficul de mesaje protejat de Enigma.

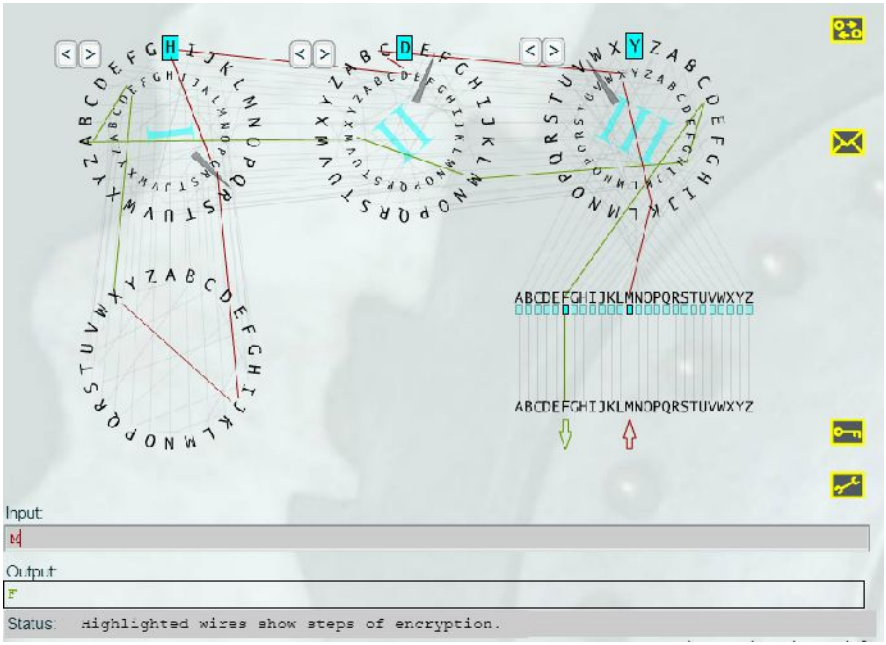
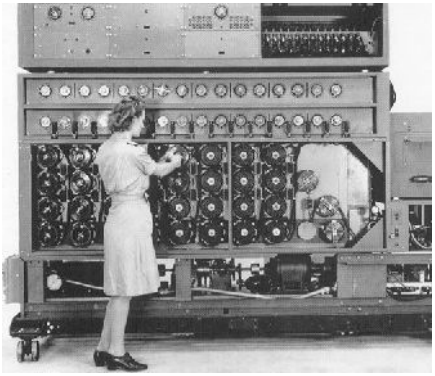


Figura 4.7. *Simulator maşina Enigma*

Maşina „Bombe” căuta setări potenţial corecte pentru un mesaj Enigma (adică, ordinea rotoarelor, setările rotoarelor, etc.), folosind un fragment de text clar probabil. Pentru fiecare setare posibilă a rotoarelor (numărul maxim posibil fiind de ordinul a 1019 stări, sau 1022 pentru maşinile Enigma de la U-boat, care aveau patru rotoare, faţă de maşina Enigma standard care avea doar trei). Aceasta efectua un lanţ de deducţii logice pe baza fragmentului probabil, deducţii implementate electric. „Bombe” detecta când avea loc o contradicţie, şi elimina setarea, trecând la următoarea. Peste două sute de astfel de maşini create de Alan Turing au fost în funcţiune până la sfârşitul războiului.

Maşinile cu rotor au fost folosite activ pe parcursul războiului II mondial. Pe lângă maşina germană Enigma au fost folosite şi Sigaba (SUA), Typex (Marea Britanie), Red, Orange, Purple (Japonia). Maşinile cu rotor au fost vârful criptografiei formale deoarece realizau cifruri suficient de rezistente într-un mod relativ simplu.



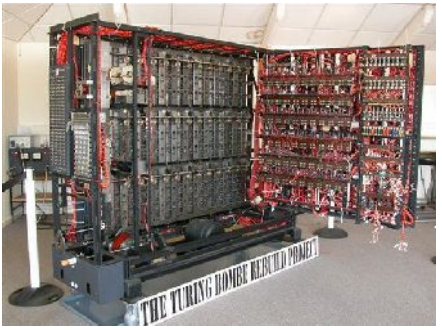


Figura 4.8. *Maşina BOMBE* (*Alan Turing*)

Atacurile încununate de succes asupra maşinilor cu rotor au fost posibile numai la începutul anilor 40 odată cu apariţia maşinilor electronice de calcul. Tot în această perioadă criptografia devine ştiinţific ramură aparte a matematicii odată cu publicarea (anul 1949) articolului lui Claude Elwood Shannon „Communication Theory of Secrecy Systems”., care a pus bazele ştiinţifice ale sistemelor de criptare cu cheie secretă (sistemelor simetrice).

Tema 5. Algoritmi simetrici de criptare. Cifruri bloc. Reţeaua Feistel

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

Criptografia modernă utilizează în principiu aceeaşi algoritmi ca şi criptografia tradiţională (transpoziţia şi substituţia), dar accentul cade pe complexitatea algoritmilor. Obiectivul criptografic din actuala perioadă este de a concepe algoritmi de criptare atât de complecşi şi de ireversibili încât atacatorul (sau criptanalistul), chiar şi în situaţia în care are la dispoziţie cantităţi mari de text criptat la alegerea sa, să nu poată face nimic fără cheia secretă.

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

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

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

Reţeaua (cifrul, schema) Feistel

Algoritmii de tip bloc sunt foarte des folosiţi în criptografia modernă, iar majoritatea algoritmilor tip bloc utilizaţi în criptarea simetrică la ora

actuală se bazează pe o structură numită *cifru bloc Feistel* sau *reţea* (uneori *schema*) *Feistel*. Ea a fost elaborată de către Horst Feistel (30.01.1915 – 14.11.1990) – unul dintre întemeietorii criptografiei moderne. După cum am mai menţionat, un cifru bloc operează asupra blocurilor de text clar de lungime *n* biţi pentru a produce un bloc de text cifrat de aceeaşi lungime (*n* biţi). Un cifru de substituţie reversibil arbitrar nu este practic pentru o dimensiune mare a blocului, din punct de vedere al implementării şi al performanţei. În general, pentru un cifru bloc de substituţie arbitrar de *n*-biţi, dimensiunea cheii este $n \cdot 2^n$. Pentru un bloc de 64 de biţi, care e o dimensiune necesară pentru a zădărnici atacurile statistice, dimensiunea cheii este

$$64 \cdot 2^{64} = 2^{70} = 10^{21} \text{ biţi.}$$

Considerând aceste dificultăţi, Feistel remarcă faptul că este nevoie de o aproximare a unui cifru bloc ideal, pentru valori mari ale lui *n*, construit din componente ce pot fi realizate uşor.

Feistel numeşte o substituţie generală de *n*-biţi ca fiind cifrul bloc ideal, deoarece permite numărul maxim de criptări posibile din blocuri de text clar în blocuri de text cifrat. 4 biţi la intrare produc una din cele 16 stări de intrare posibile, care sunt asociate de cifrul cu substituţie într-o singură stare de ieşire din cele 16 posibile, fiecare fiind reprezentată de 4 biţi de text cifrat. Funcţiile de criptare şi decriptare pot fi definite printr-un tabel.

O structură Feistel are avantajul că cifrarea şi decifrarea sunt foarte similare sau chiar identice în unele cazuri (ceea ce ne aminteşte de Enigma), cerând doar o reversie a cheii. Astfel, dimensiunea codului sau circuitului necesar pentru a implementa un astfel de cifru este practic înjumătăţit.

Reţelele Feistel şi construcţii similare combină mai multe „runde" de operaţii repetate cum ar fi:

- *amestecarea de biţi* (numită şi permutări pe cutii *P*),
- *funcţii simple ne-lineare* (numite si substituţii prin cutii *S*),
- *amestecul liniar* (în sensul algebrei modulare) utilizând XOR,

pentru a produce o funcţie care conţine cantităţi mari de date, numite de Claude Shannon „*confuzie* şi *difuzie*". Amestecarea de biţi creează difuzia iar substituţia - confuzia. În criptografie confuzia se referă la a face o relaţie între cheie şi textul cifrat cât de complex şi adânc posibil, iar difuzia este definită ca proprietatea că redundanţa în statisticele textului clar este

disipată în statisticele textului cifrat. Difuzia este asociata cu dependenţa biţilor de la ieşire de biţii de la intrare. Într-un cifru cu o difuzie perfecta, doar schimbarea unui bit de la intrare ar schimba întregul text, ceea ce se mai numeşte şi SAC (*Strict Avalanche Criterion*). Feistel utilizează *cutiile P* (*P-box* sau *Permutation-box*) şi amestecul liniar de biţi pentru a atinge o difuzie aproape perfectă şi se poate spune că îndeplineşte condiţiile SAC.

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

Un exemplu elocvent de tabel fix este tabelul de 6×4 - biţi al cutiei *S* (S_5) din *DES* (Tabelul 5.1):

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

Tabelul 5.1. *Cutia* $\boldsymbol{S_5}$ *din DES*

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

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

Reţele Feistel au fost introduce pentru prima dată, în domeniu comercial, în cifrul *Lucifer* de la IBM care a fost conceput de însuşi Feistel şi Don Coppersmith. Reţele Feistel au câştigat respect atunci când guvernul SUA a adoptat standardul de securitate a datelor DES. Ca şi alte

componente ale lui DES, exista natura iterativă a construcţiei Feistel care face foarte simple implementările criptosistemului în electronică.

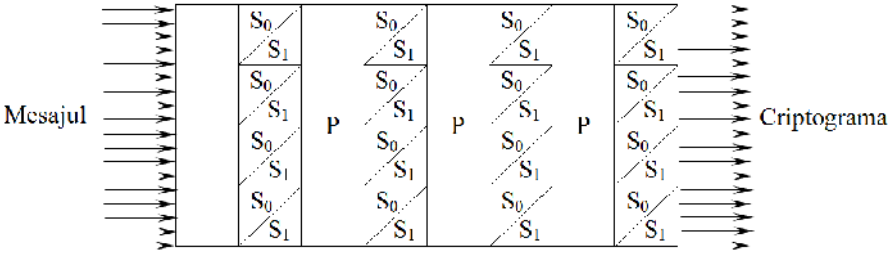


Fig. 5.1. *Schema unei „cutii-S”*

Multe cifruri simetrice de tip bloc se bazează pe reţele Feistel şi pe structura şi proprietăţile cifrului Feistel, care a fost intens explorat de cate criptologi. În special Michael Luby şi Charles Rackoff, care au analizat cifrul şi au demonstrat că dacă funcţia „*round-robin*” (un round-robin este un aranjament prin care se aleg în ponderi egale toate elementele unui grup sau a unei liste, într-o ordine raţională, de obicei de sus până jos, şi după aia pornind de la început) este un generator criptografic sigur de numere pseudoaleatoare, cu K_i utilizat ca seed, atunci 3 runde sunt suficiente pentru a face cifrul bloc sigur, pe când 4 runde sunt suficiente să facă cifrul „puternic” sigur, însemnând că este sigur pentru un atac prin text cifrat ales. Din cauza acestui rezultat, cifrurile Feistel au fost uneori numite cifruri pe blocuri.

Modul de operare al cifrului Feistel este următorul:

1. Împarte textul clar în doua blocuri egale (L_0, R_0)
2. Pentru fiecare rundă i=1,2,...,n, calculează:

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

unde f este funcţia, de obicei tot XOR, şi K_i este sub-cheia. Atunci textul cifrat va fi (L_n, R_n)

3. Repeta.

Indiferent de natura funcţiei f, decifrarea se face prin:

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

Un avantaj al acestui model este că funcţia utilizată nu trebuie neapărat să fie inversabilă şi poate să fie foarte complexă. Am menţionat că funcţia f

este de obicei *XOR*. Acest lucru nu este complet adevărat. Funcţia poate să fie oricare, însa pentru ilustrare se foloseşte o funcţie simplă şi relativ sigură, cum ar fi *XOR*.

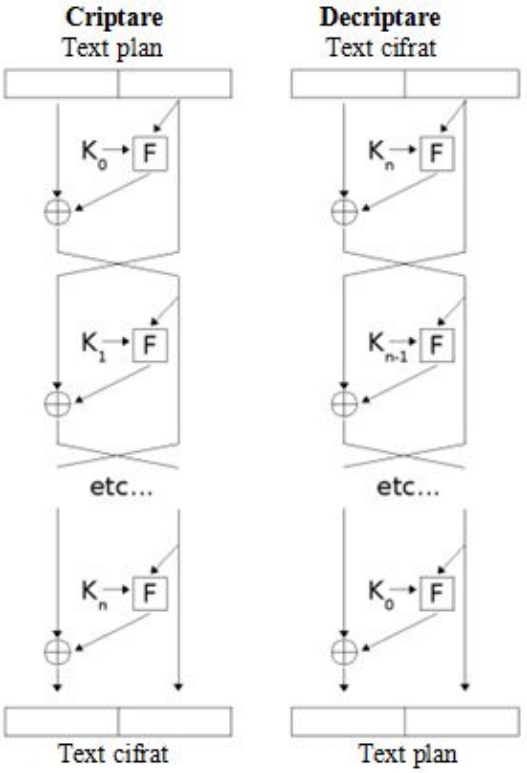


Figura 5.2. *Schema reţelei Feistel*

În Figura 5.2 este arătat cum acest model trece textul clar în text cifrat. Este de notat reversia sub-cheii pentru decriptare, aceasta fiind singura diferenţă între cifrare si descifrare.

Mai există însă un tip de cifru Feistel, numit Feistel debalansat, care utilizează o structură modificată în care L_0 şi R_0 nu sunt egale în lungime. Un exemplu de asemenea cifru este *Skipjack*.

Construcţia Feistel este utilizată şi în algoritmi care nu sunt cifruri pe blocuri. De exemplu, *Optimal Asymmetric Encryption Padding* (OAEP)

utilizează o reţea Feistel simplă pentru a randomiza textele cifrate în unele scheme de cifrare asimetrică.

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

Tema 6. Algoritmul de cifrare Lucifer

Algoritmul de criptare *Lucifer* a fost elaborat la începutul anilor 70 şi a stat la baza algoritmului DES – primului standard de cifrare din SUA. Algoritmul şi istoria sa sunt suficient de interesante pentru a fi studiate aparte.

Algoritmul Lucifer este adesea numit „primul algoritm de cifrare pentru aplicaţii civile”. În realitate Lucifer reprezintă nu un singur algoritm ci o familie de algoritmi legaţi intre ei (care au fost elaboraţi în cadrul programului Lucifer al companiei IBM şi care prevedea cercetări în domeniul criptografiei), care erau algoritmi de criptare de tip bloc realizaţi în perioade diferite de timp.

După spusele vestitului criptolog american Bruce Schneier (născut la 15.01.1963), „există cel puţin doi algoritmi diferiţi cu acest nume şi aceasta a dus la o încărcătură vizibilă”. Iar în Wikipedia sunt menţionate 4 versiuni ale acestui algoritm.

Versiunea iniţială a algoritmului Lucifer a fost elaborată de un colectiv de specialişti ai companiei IBM sub conducerea lui Horst Feistel. Această versiune a algoritmului a fost brevetată de către compania IBM în anul 1971 (brevetul a fost eliberat în 1974 în SUA cu numărul 3798359 http://www.freepatentsonline.com/3798359.pdf).

Această versiune a algoritmului cifrează datele pe blocuri de 48 biţi cu utilizarea cheii de 48 de biţi.

Algoritmul este unul bazat pe o reţea de permutări şi substituţii. În procesul de cifrare se execută 16 runde de transformări (număr de runde recomandat de autor), în fiecare dinte ele fiind realizate acţiuni în conformitate cu Figura 6.1:

1. *Asigurarea feedback-ului* (conexiunii inverse). Se efectuează operaţia logică „XOR” între fiecare bit al blocului procesat şi valoarea precedentă a aceluiaşi bit în cazul în care bitul analogic al cheii de rundă are valoarea 1; în caz contrar operaţia nu se efectuează. Valoarea precedentă a fiecărui bit procesat se salvează în registrul blocului de feedback. În prima rundă a algoritmului blocul de feedback nu efectuează operaţia XOR ci memorizează doar pentru următoarea rundă biţii blocului prelucrat.
2. *Transformarea prin confuzie.* Se efectuează o transformare neliniară a datelor obţinute la operaţia precedentă prin intermediul

substituţiei tabelare care este în funcţie de un bit concret al cheii de rundă. Această funcţie este suficient de simplă: dacă valoarea bitului respectiv este 1, se efectuează substituţia tabelară S_0, în caz contrar se aplică substituţia S_1. Fiecare niblu[1] de date se modifică independent de alte date, astfel tabele înlocuiesc valoarea de intrare de 4 biţi cu o altă valoare care la fel are 4 biţi. Trebuie de menţionat că în acest brevet nu sunt prezentate valorile tabelelor de substituţie; în calitate de exemplu este dat numai Tabelul 6.1

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

Tabelul 6.1. *Exemplu de tabel de confuzie pentru brevetul alg. Lucifer*

Aceasta înseamnă că valoarea de intrare 0 se înlocuieşte cu 2, valoarea 1 – cu 5 ş.a.m.d. până la valoarea 15 care se înlocuieşte cu 11.

3. *Transformarea prin difuzie*. Această transformare constă în redirecţionarea simplă a biţilor de intrare în aşa mod încât valorile tuturor biţilor de intrare se schimbă cu locul după o anumită regulă. La fel ca şi valorile tabelelor de substituţie, regula după care se face difuzia datelor nu este descrisă în brevet. Operaţia de difuziune se efectuează absolut independent de valoarea cheii de cifrare.
4. *Aplicarea cheii*. Această operaţie se efectuează prin aplicarea bit cu bit a operaţiei XOR asupra rezultatului operaţiei precedente şi a biţilor respectivi a cheii de rundă.

După cum se poate observa din descrierea de mai sus la fiecare rundă de a cifrării sunt necesari 108 biţi:

1. 48 de biţi pentru blocul de feedback;
2. 12 biţi pentru blocul de substituţie;
3. 48 biţi pentru aplicarea cheii.

[1] Un *niblu* este o colecţie de 4 biţi cu care se pot reprezenta 16 valori diferite.

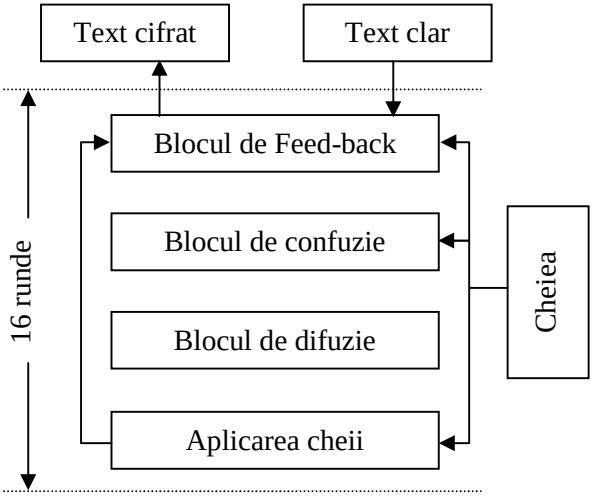


Figura 6.1. *Schema generală a Versiunii* 1 *Lucifer*

Pentru obţinerea a 16 sub-chei de rundă de 108 biţi fiecare din cheia iniţială de 48 de biţi se aplică procedura de extindere a cheii (Figura 6.2):

1. Cheia de 48 de biţi se înscrie în registrele cheii (Key Register) *KR*1, *KR*2, …, *KR*6.
2. Din aceste registre se extrage cantitatea necesară de biţi prin intermediul permutării de extensie *KEP* (Key Extension Permutation). Operaţia *KEP* nu efectuează calcule, obţinerea celor 108 biţi de informaţie din cheie din cei 48 de biţi are loc prin utilizarea multiplă a anumitor biţi din *KR*1, *KR*2, …, *KR*6. Permutarea *KEP* nu este descrisă în detalii în specificaţiile algoritmului.
3. Pentru obţinerea din cheie a informaţiei pentru runda următoare se efectuează deplasarea de un bit în fiecare din registrele *KR*1, *KR*2, …, *KR*6. Apoi din se aplică permutarea *KEP*. Acest pas se repetă ciclic un număr necesar de ori până la sfârşitul executării algoritmului.

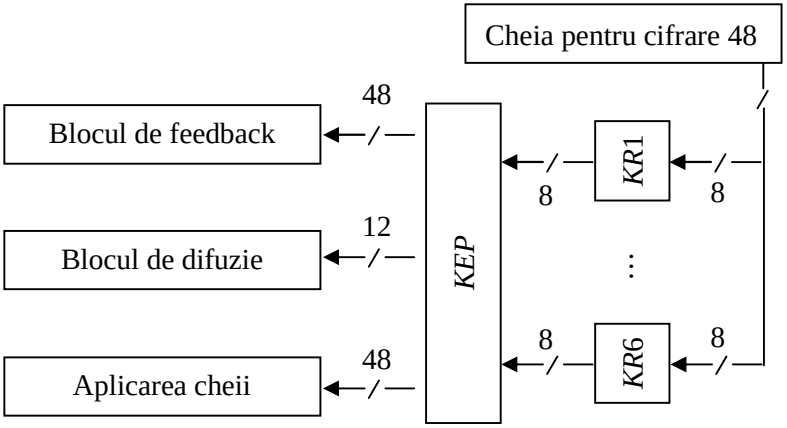


Figura 6.2. *Procedura de extensie a cheii în Versiunea* 1 *Lucifer*

Trebuie de menţionat că acest algoritm este strict axat pe implementare hardware – în brevetul respectiv sunt prezentate câteva scheme amănunţite care descriu realizările hardware posibile ale algoritmului, pe când cifrării software practic nu i se acordă atenţie în acest brevet.

Este evident că în descrierea algoritmului există multe „pete albe”. De exemplu nu sunt prezentate tabelele de substituţie, nu este descrisă în detalii transformarea liniară utilizată, lipseşte descrierea permutării *KEP* etc. De fapt, brevetul stabileşte un model pentru dezvoltarea pe această bază algoritmilor criptografici concreţi cu proceduri „rafinate”.

Concomitent cu brevetul depus la 30 iunie 1971 Horst Feistel a mai depus 2 cereri de brevetare, ambele propunând aplicaţii specifice ale algoritmului expus mai sus:

- protecţia datelor transmise şi prelucrate în sisteme cu terminale multiple cu utilizarea algoritmului Lucifer; schema brevetată presupune de asemenea şi asigurarea integrităţii datelor dar şi prezenţa a două regimuri de autentificare a utilizatorilor: cu parolă şi „solicitare-confirmare”.
- cifrarea datelor pe parcursul a mai multe etape, ceea ce asigură diverse nivele de securitate la transmiterea informaţiei în sistemele cu terminale multiple; această schemă era la fel bazată pe aplicarea algoritmului Lucifer.

Ca şi brevetul pentru algoritmul Lucifer, brevetele menţionate au fost obţinute de către compania IBM în anul 1974.

Versiunea a doua a algoritmului Lucifer a apărut foarte curând după prima – în acelaşi an 1971. Se vede că autorii algoritmului Lucifer înţelegeau că mărimea cheii de 48 de biţi este insuficientă (în prezent această mărime a cheii este nepermisă) şi au prezentat algoritmul care cifra cu o cheie de 64 biţi. Algoritmul dat efectua însă cifrarea cu blocuri de numai 32 biţi ceea ce evident este insuficient (pentru un bloc de lungime n biţi la cifrarea a $2^{\frac{n}{2}}$ blocuri de text clar cu aceeaşi cheie vor avea loc scurgeri de informaţie despre datele cifrate). Această versiune a algoritmului a fost brevetată la fel de către IBM şi descrisă în patentul nr. 3796830 în martie 1974 (http://www.freepatentsonline.com/3796830.pdf).

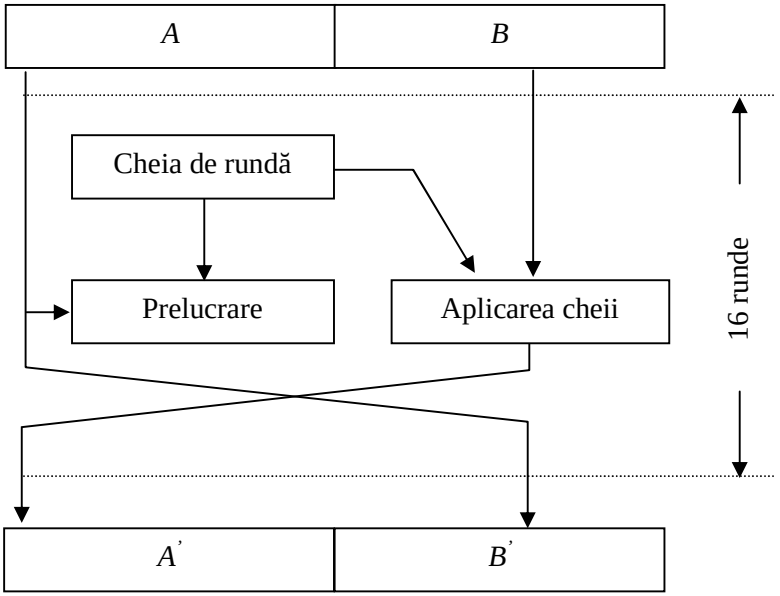


Figura 6.3. *Structura Versiunii* 2 *Lucifer*

Versiunile 1 şi 2 au semnificativ mai multe discrepanţe decât similitudini. Versiunea a doua a algoritmului divizează blocul cifrat de 32 biţi în două sub-blocuri de câte 16 biţi şi efectuează 16 runde de

transformări, în fiecare din care se realizează următoarea consecutivitate de operaţii (Figura 6.3):

1. Blocul de 32 biţi este divizat în 2 sub-blocuri *A* şi *B* de 16 biţi.
2. Sub-blocul *A* se divizează în 4 fragmente a câte 4 biţi, fiecare fragment fiind prelucrat aparte (Figura 6.4), adică celelalte operaţii se repetă pentru fiecare fragment.

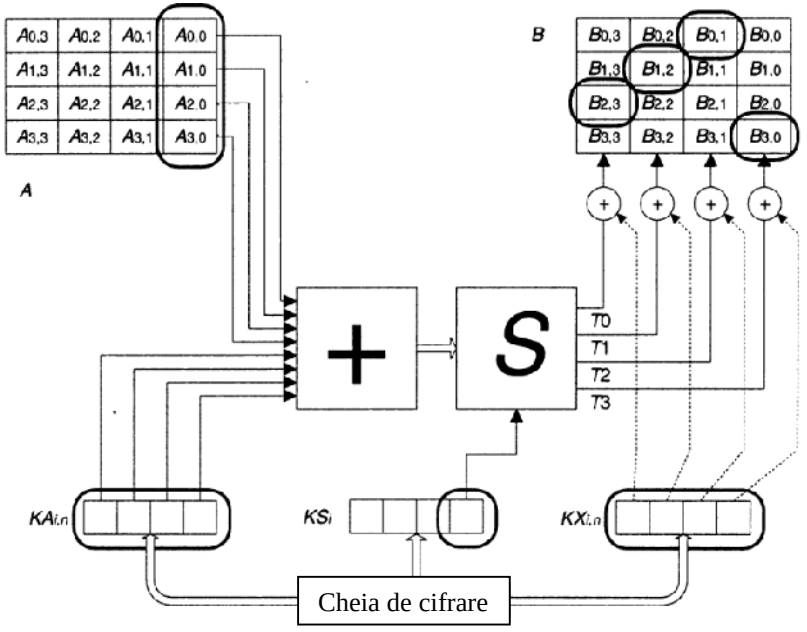


Figura 6.4. *Prelucrarea fragmentului sub-blocului A* (*vers.* 2 *Lucifer*)

3. Fiecare dintre aceste fragmente se sumează *modulo* 16 cu $KA_{i,n}$ – un fragment de 4 biţi ai cheii de rundă pentru operaţia de adunare (procedura de extindere a cheii de cifrare va fi expusă mai jos), unde:
   - n – numărul de ordine al fragmentului prelucrat;
   - i – numărul de ordine a rundei curente.
4. Asupra fiecărui fragment se aplică o substituţie care este în funcţie de cheie: fragmentul prelucrat se înlocuieşte cu valoarea de 4 biţi T_0 T_1 T_2 T_3 în conformitate cu Tabelul 6.2.

Valoarea de intrare																*Rezul-tatul*
0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	
~*k*	~*k*	*k*	1	~*k*	1	~*k*	0	*k*	*k*	0	*k*	~*k*	*k*	*k*	~*k*	$\boldsymbol{T_0}$
1	~*k*	1	0	~*k*	~*k*	0	*k*	*k*	1	0	0	*k*	0	~*k*	*k*	$\boldsymbol{T_1}$
k	1	1	~*k*	1	*k*	*k*	0	~*k*	~*k*	~*k*	*k*	*k*	*k*	0	0	$\boldsymbol{T_2}$
k	~*k*	~*k*	*k*	*k*	1	~*k*	0	0	*k*	1	~*k*	~*k*	*k*	~*k*	*k*	$\boldsymbol{T_3}$

Tabelul 6.2. *Substituţia aplicată fiecărui fragment al sub-blocului* unde:

- $k = KS_i[n]$, reprezintă bitul *n* al fragmentului de 4 biţi ai cheii de rundă, care gestionează substituţiile (KS_i),
- ~*k* – valoarea inversă a lui *k*.

În conformitate cu valoarea de intrare, din tabel se aleg valorile T_0, T_1, T_2, T_3 ale biţilor, de exemplu pentru valoarea de intrare 14 şi $k = 1$ se vor obţine următoarele valori:

$$T_0 = 1,\ T_1 = 0,\ T_2 = 0,\ T_3 = 0.$$

5. Se aplică operaţia XOR asupra cortegiului de biţi T_0, T_1, T_2, T_3 (care nu este altceva decât rezultatul prelucrării fragmentului sub-blocului *A*) şi a anumitor biţi ai sub-blocului *B*. Biţii sub-blocului *B* care participă în această operaţie se aleg cu ajutorul unui fragment determinat de 4 biţi ai cheii de rundă $KX_{i,n}$ în conformitate cu tabelul 6.3.

	Valoarea bitului de control $KX_{i,n}$	
	1	0
T_0	$B_{0,(n+1 \bmod 4)}$	$B_{0,n}$
T_1	$B_{1,(n+2 \bmod 4)}$	$B_{1,(n+3 \bmod 4)}$
T_2	$B_{2,(n+3 \bmod 4)}$	$B_{2,(n+2 \bmod 4)}$
T_3	$B_{3,n}$	$B_{3,(n+1 \bmod 4)}$

Tabelul 6.3. *Alegerea biţilor pentru cheia de rundă*

Bitul de control pentru T_i este bitul nr. *i* al fragmentului $KX_{i,n}$. De exemplu, la prelucrarea fragmentului nr. 0 şi a tuturor biţilor egali cu 1 din $KX_{i,n}$ se efectuează următoarele operaţii (vezi figura 6.4 – toate datele incluse în prelucrarea fragmentului nr. 0 al sub-blocului pentru toţi biţii egali cu 1 sunt evidenţiate cu linii de grosime mai mare):

$$B_{0,1} = B_{0,1} + T_0;$$
$$B_{1,2} = B_{1,2} + T_1;$$
$$B_{2,3} = B_{2,3} + T_2;$$
$$B_{3,0} = B_{3,0} + T_3.$$

După fiecare rundă în afară de ultima sub-blocurile se schimbă cu locurile.

Extinderea cheii, adică obţinerea cantităţii necesare de fragmente ale cheilor de rundă:

- $KA_{i,0}$, $KA_{i,1}$, $KA_{i,2}$, $KA_{i,3}$ – pentru adunarea *modulo* 16;
- KS_i – pentru participarea la substituţia tabelară;
- $KX_{i,0}$, $KX_{i,1}$, $KX_{i,2}$, $KX_{i,3}$ – pentru controlul aplicării rezultatului asupra sub-blocului *B*, se efectuează într-un mod suficient de simplu:
    1. Cheia de cifrare de 64 biţi se divizează în 16 părţi a câte 4 biţi care se numerotează de la 0 la 15.
    2. În caz de necesitate se alege un fragment determinat al cheii în conformitate cu Tabelul 6.4.

Runda	KS_i	$KA_{i,0}$	$KX_{i,0}$	$KA_{i,1}$	$KX_{i,1}$	$KA_{i,2}$	$KX_{i,2}$	$KA_{i,3}$	$KX_{i,3}$
1	0	1	2	3	4	5	6	7	8
2	9	10	11	12	13	14	15	0	1
3	2	3	4	5	6	7	8	9	10
4	11	12	13	14	15	0	1	2	3
5	4	5	6	7	8	9	10	11	12
6	13	14	15	0	1	2	3	4	5
7	6	7	8	9	10	11	12	13	14
8	15	0	1	2	3	4	5	6	7
9	8	9	10	11	12	13	14	15	0
10	1	2	3	4	5	6	7	8	9
11	10	11	12	13	14	15	0	1	2
12	3	4	5	6	7	8	9	10	11
13	12	13	14	15	0	1	2	3	4
14	5	6	7	8	9	10	11	12	13
15	14	15	0	1	2	3	4	5	6
16	7	8	9	10	11	12	13	14	15

Tabelul 6.4. *Tabelul pentru alegerea cheii în caz de necesitate*

Această versiune a algoritmului este descrisă mult mai amănunţit decât cea precedentă: în patentul respectiv (nr. 3796830) lipsesc „petele albe” deci acest algoritm poate fi pe deplin realizat numai cu ajutorul informaţiei ce se conţine în patentul respectiv.

Ca şi în cazul precedent în brevetul pentru versiunea a doua a algoritmului Lucifer a fost prezentată o informaţie amănunţită referitoare la realizarea hardware a algoritmului, particularităţile realizării software în brevet nu sunt examinate.

Versiunea 3 a algoritmului Lucifer este cel mai puţin descrisă în detalii, ea fiind publicată în articolul Feistel H. *Cryptography and Computer Pivacy*. Scientific American, May 1973, Vol. 228, No. 5, pp. 15-23 (http://www.apprendre-en-ligne.net/crypto/bibliotheque/feistel/). Dacă să judecăm după descriere, algoritmul constă dintr-o succesiune de operaţii (Figura 6.5):

- Tabelele de substituţie gestionate de biţii de control ai cheii secrete – analogic versiunii 1, în funcţie de valoare bitului de control al cheii se efectuează substituţia S_0 sau S_1;
- Permutări fixate *P* ale biţilor.

Aplicarea multiplă a acestor operaţii reprezintă acest algoritm de criptare.

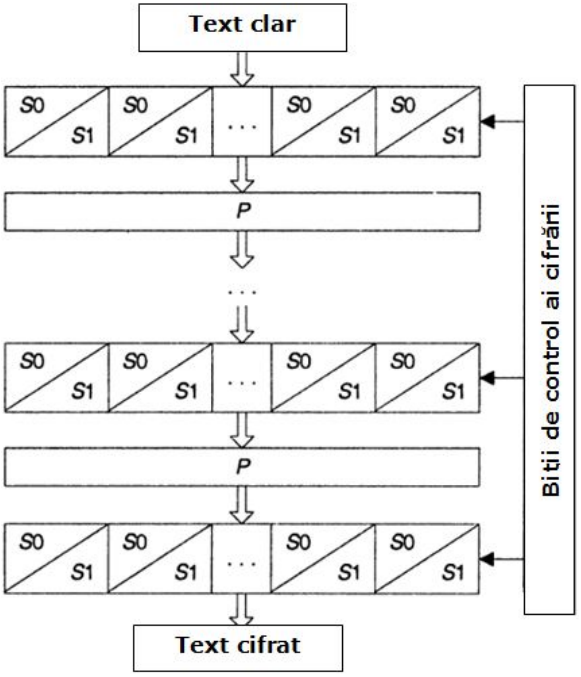


Figura 6.5. *Structura Versiunii 3 Lucifer*

În articolul menţionat mai sus nu sunt indicaţi majoritatea covârşitoare a parametrilor algoritmului: nu este dată nici numărul exact de runde, nici valorile tabelelor de substituţie şi permutare, nu sunt indicate dimensiunile blocului şi a cheii (mărimile de 128 biţi ale blocului şi cheii sunt indicate doar ca valori posibile). În acest mod versiunea dată a algoritmului este mult mai „şablonizată” decât prima versiune.

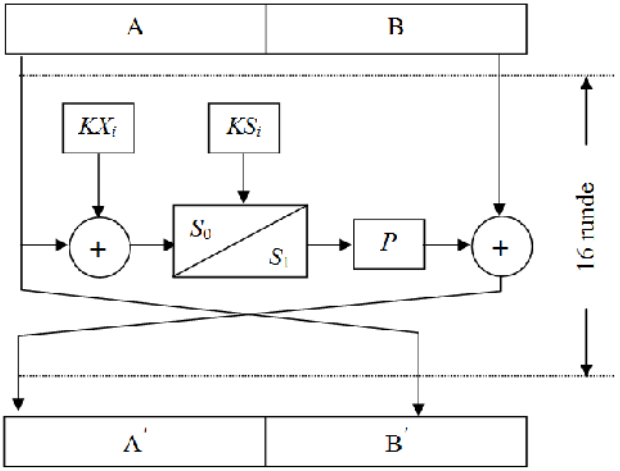


Figura 6.6. *Structura Versiunii 4 Lucifer*

Versiunea Lucifer 4. Putem afirma că versiunea dată este asemănătoare concomitent cu toate cele trei versiuni precedente. La fel ca la versiunea 2 aici fiecare bloc de 128 biţi ai textul clar se divizează în două sub-blocuri unul dinte care se prelucrează în felul indicat în figura 6.6 (Savard J. *Lucifer: the first block cipher*, http://www.quadibloc.com/crypto/co0401.htm):

1. Se aplică cheia de rundă prin efectuarea operaţiei XOR asupra biţilor sub-blocului prelucrat şi a 64 biţi ai fragmentului cheii de rundă KX_i (i – numărul de ordine a rundei curente).
2. Substituţia tabelară dirijată de cheie este înfăptuită într-un mod foarte asemănător cu versiunile 1 şi 3:
   - dacă valoarea bitului j (j = 0, 1, …, 7) al fragmentului de 8 biţi ai cheii de rundă KS_i este egală cu 1, atunci

cei doi nibli ai octetului *j* care este prelucrat la moment se schimbă cu locurile;

- niblului superior al fiecărui octet i se aplică tabelul de substituţie S_0, pe când niblului inferior – tabelul S_1. Tabelele S_0 şi S_1 sunt definite cum este indicat în Tabelul 6.5.

3. Se execută permutarea fixă *P* a biţilor sub-blocului în conformitate cu Tabelul 6.6. Valoarea bitului 10 de intrare se înscrie în bitul de ieşire 0, valoarea bitului 21 – în bitul 1 etc. (se numerotează de la stânga la dreapta începând cu bitul 0)

	Valoarea de intrare															
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
S_0	12	15	7	10	14	13	11	0	2	6	3	1	9	4	5	8
S_1	7	2	14	9	3	11	0	4	12	13	1	10	6	15	8	5

Tabelul 6.5. *Tabelele S_0 şi S_1 pentru Lucifer* - 4

10	21	52	56	27	1	47	38	18	29	60	0	35	9	55	46
26	37	4	8	43	17	63	54	34	45	12	16	51	25	7	62
42	53	20	24	59	33	15	6	50	61	28	32	3	41	23	14
58	5	36	40	11	49	31	22	2	13	44	48	19	57	39	30

Tabelul 6.6. *Permutarea fixă P a biţilor sub-blocului*

4. Asupra rezultatului operaţiei precedente şi a sub-blocului neprelucrat se efectuează operaţia XOR.
5. Sub-blocurile se schimbă cu locurile (în toate rundele cu excepţia ultimei).

În total se efectuează 16 runde de transformări descrise mai sus.

Procedura de extindere a cheii rezolvă problema obţinerii a 16 chei de rundă a câte 72 biţi fiecare (64 biţi KX_i şi 8 biţi KS_i) din cheia de cifrare iniţială. Extinderea cheii se efectuează întru-un mod foarte simplu:

1. În calitate de fragment KS_i este utilizat primul octet al cheii de cifrare iniţiale (calculând octeţii de la stânga la dreapta, începând cu 1).
2. În calitate de fragment KX_i se utilizează primii 8 octeţi ai cheii de cifrare iniţiale (adică primul octet se utilizează în ambele fragmente).
3. Cheia de cifrare este deplasată în mod ciclic spre stânga cu 7 octeţi, după care analogic se aleg fragmentele de cheie pentru runda următoare.

Spre deosebire de versiunile 1 şi 3 versiunea 4 este descrisă suficient de amănunţit pentru realizare atât hardware cât şi software.

Criptanaliza versiunilor algoritmilor. Primele două versiuni ale algoritmului Lucifer nu „au avut parte" de publicaţii de criptanaliză. Celelalte două versiuni însă au avut „noroc" mai mult. Sunt cunoscute câteva rezultate în această direcţie.

- În anul 1991 criptologii Eli Biham şi Adi Shamir au cercetat versiunea 3 (în articolul A Differential Cryptanalisis of Snefru, Khafre, REDOC-II, LOKI and Lucifer – http://pdf.aminer.org/000/119/748/differential_cryptanalysis_of_snefru_khafre_redoc_ii_loki_and_lucifer.pdf).
  Pentru a fi un caz determinat ei au utilizat permutarea *P* din algoritmul DES iar în calitate de S_0 şi S_1 au fost luate liniile 3 şi 4 din tabelul S_1 din DES (Tabelul 6.7).

Valoarea de intrare															
0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
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

Tabelul 6.7. *Substituţiile S_0 şi S_1* (c*azul studiat de Biham şi Shamir*)

În conformitate cu lucrarea menţionată versiunea nr. 3 cu 8 runde şi blocul de 32 biţi se poate sparge având doar 24 de texte în clar alese şi a textelor cifrate respective prin efectuarea a 2^{21} operaţii de testare pentru cifrare. În aceeaşi lucrare autorii au descris un atac posibil asupra algoritmului analogic cu blocuri de 128 biţi, pentru succesul căreia este necesar de avut 64-70 texte în clar alese şi circa 2^{53} operaţii

de cifrare. În plus, Biham şi Shamir afirmă că versiunea 4 a algoritmului Lucifer est şi mai vulnerabilă.

- Ultima afirmaţie a fost demonstrată în lucrarea autorilor Ishai Ben-Aroya şi Eli Biham *Differential Cryptanalysis of Lucifer*, Tehnio, Haifa, Israel, 1993. În această lucrare este descris atacul asupra versiunii 4 a algoritmului Lucifer, în care a fost descoperită o submulţime a cheilor (suficient de mare – circa 55% din toate valorile posibile ale cheilor), slabe la criptanaliza diferenţială. La utilizarea cheii de criptare din această submulţime algoritmul putea fi spart având 2^{36} texte în clar alese.

Deci, algoritmul Lucifer nu a fost un algoritm suficient de sigur pentru a se menţine în aplicare, ba mai mult chiar – a fost unul cu multe neajunsuri, însă el a fost „strămoşul" algoritmilor simetrici de criptare care au fost implementaţi ulterior, primul urmaş al său fiind chiar vestitul DES (Data Encription Standard) – primul standard de criptare din SUA.

Tema 7. Algoritmul DES

Plecând de la algoritmul "Lucifer" conceput în Laboratoarele IBM a fost dezvoltat algoritmul de criptare *DES* (Data Encryption Standard). Algoritmul a fost conceput pentru guvernul Statelor Unite dar şi pentru folosinţă publică. Trebuie de menţionat că chiar şi cei mai mari experţi în domeniul criptografiei fac diverse presupuneri referitor la versiunea algoritmului Lucifer care a fost predecesorul lui DES. În mai 1973, revista *Federal Register* a sintetizat principiile care trebuie să stea la baza proiectării unui algoritm criptografic standard:

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

DES a fost oficial adoptat ca standard federal la 23 noiembrie 1976, iar în 1977 specificaţiile sale au fost făcute publice.

Algoritmul DES este o combinaţie complexă, folosind două blocuri fundamentale în criptografie: substituţia şi permutarea (transpoziţia). Acest cifru bloc acceptă un bloc de 64 de biţi la intrare şi generează un bloc cifrat de 64 de biţi. DES este un algoritm simetric. Acelaşi algoritm şi aceeaşi cheie sunt folosiţi atât la criptare cât şi la decriptare.

Algoritmul este constituit din 16 cicluri repetate ale blocurilor fundamentale. Textul iniţial este descompus în blocuri de 64 de biţi. Cheia este de 64 biţi din care doar 56 sunt efectivi, ceilalţi fiind biţi de paritate. Folosirea substituţiei provoacă confuzie prin sistematica substituire a unor biţi cu alţii. Transpoziţiile provoacă difuzie prin re-ordonarea biţilor.

Algoritmul foloseşte numai operaţii aritmetice şi logice clasice cu număr de până la 64 de biţi, ceea ce face relativ uşor de implementat atât

software cât mai ales hardware: unul din scopurile declarate ale algoritmului fiind uşoara lui implementare hardware într-un cip specializat.

Parcurgerea celor 16 cicluri are loc după schema din figura 7.1:

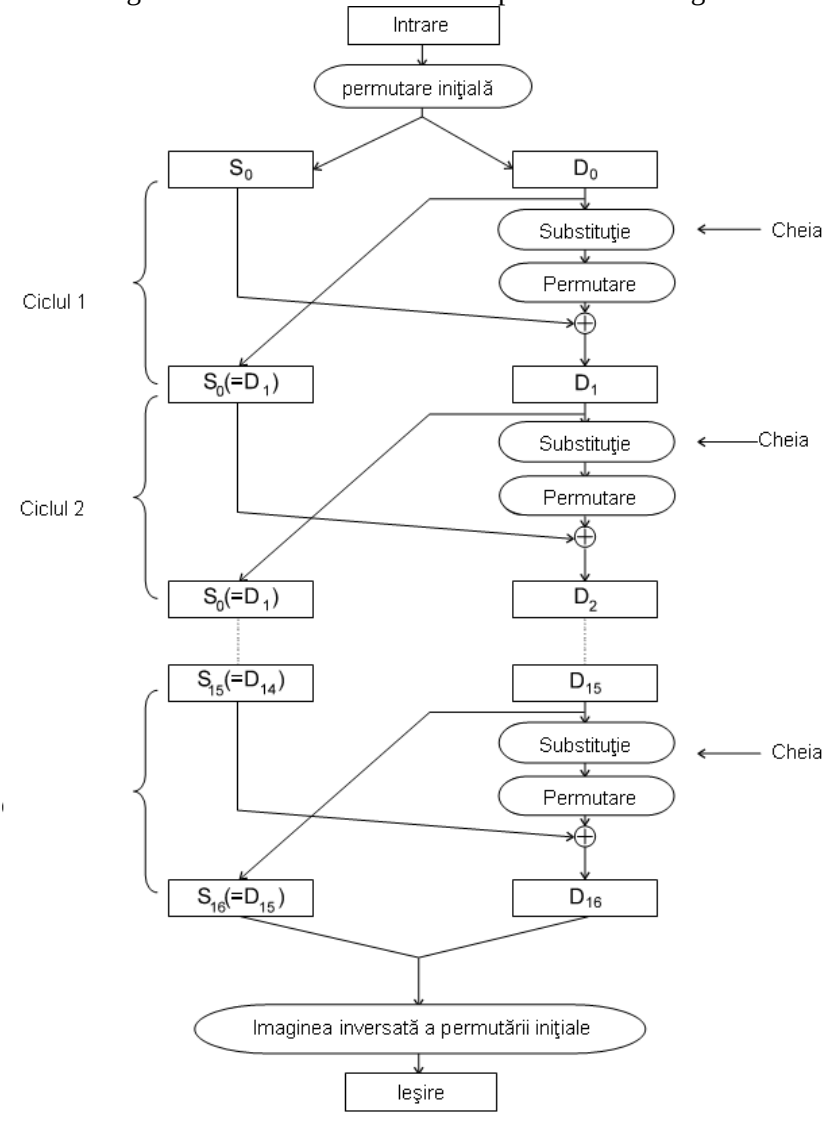


Figura 7.1 *Detalii pentru folosirea algoritmului DES*

La intrare datele sunt împărţite în blocuri de 64 biţi, care sunt transformate folosind cheia de 64 de biţi. Cei 64 de biţi sunt permutaţi prin „*permutarea iniţială – IP*”. În continuare, urmează operaţiile ce constituie un ciclu. Blocul de 64 de biţi este separat în două, „jumătatea stângă” şi “jumătatea dreaptă”, fiecare de 32 de biţi. Cheia este deplasată la stânga cu un număr de biţi şi permutată: ea se combină cu “partea dreaptă” care apoi se combină cu “partea stângă”; rezultatul devine noua “parte dreaptă”; vechea “parte dreaptă” devine noua “parte stângă” (figura 7.2).

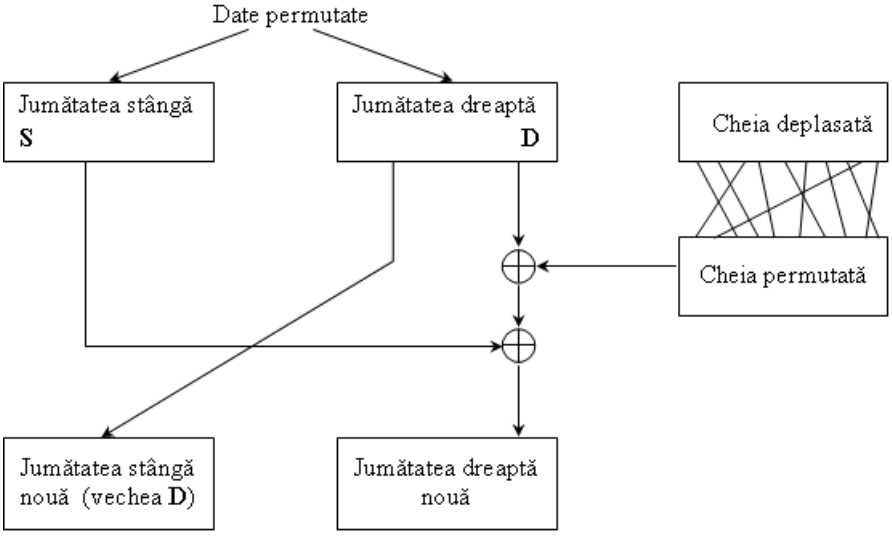
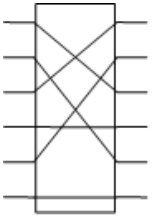


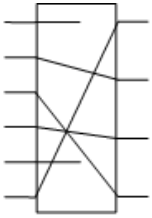
Figura 7.2 *Manipularea cheii în algoritmul DES*

După repetarea acestui ciclu de 16 ori se face permutarea finală care este inversă permutării iniţiale.

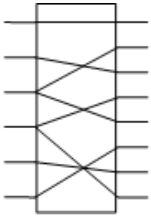
Pentru combinarea unei secvenţe de 32 biţi cu cheia de 64 biţi se folosesc expandări de la 32 biţi la 48 biţi şi reducerea cheii de la 64 biţi la 48 biţi prin alegerea anumitor biţi, operaţii ce le numim “permutare expandată” şi “permutare aleasă” (figura 7.3).



Permutare



Permutare aleasă



Permutare expandată

Figura 7.3 *Manipularea permutării în algoritmul DES*

În fiecare ciclu practic au loc patru operaţii separate:

1. partea dreaptă este expandată de la 32 la 48 biţi;
2. partea dreaptă este combinată cu o formă a cheii;
3. rezultatul este substituit şi condensat în 32 biţi,
4. cei 32 biţi sunt permutaţi şi apoi combinaţi cu partea stângă pentru a da o nouă parte dreaptă (figura 7.4).

Permutarea expandată este definită în Tabelul 7.1:

Bit	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
se mută la	2,48	3	4	5,7	6,8	9	10	11,13
Bit	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**
se mută la	12,14	15	16	17,19	18,20	21	22	23,25
Bit	**17**	**18**	**19**	**20**	**21**	**22**	**23**	**24**
se mută la	24,26	27	28	29,31	30,32	33	34	35,37
Bit	**25**	**26**	**27**	**28**	**29**	**30**	**31**	**32**
se mută la	36,38	39	40	41,43	42,44	45	46	47,1

Tabelul 7.1 *Definirea permutării expandate în DES*

Cheia este împărţită în două părţi de 28 biţi deplasate la stânga cu un număr de biţi apoi reunite şi 48 din cei 56 de biţi sunt permutaţi şi folosiţi ca o cheie de 48 de biţi de-a lungul ciclului.

Cheia dintr-un ciclu este combinată printr-o funcţie sau exclusiv cu "partea dreaptă" expandată. Rezultatul este operat în 8 "*cutii-S*" care efectuează substituţia. O "*cutie-s*" este o tabelă în care 6 biţi de date sunt înlocuiţi de 4 biţi. Permutările sunt efectuate de tabele numite "*cutii-P*".

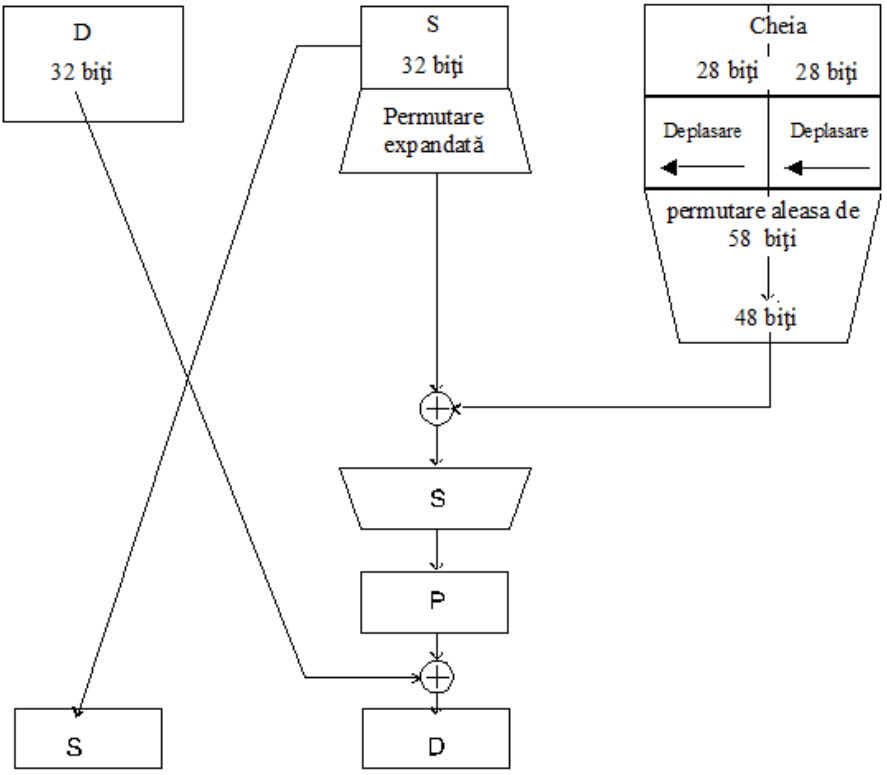


Figura 7.4. *Ciclul în algoritmul DES*

Cu algoritmul DES se poate face atât criptarea cât şi decriptarea unui mesaj. Rezultatul este adevărat pentru că ciclul j derivă din ciclul $(j-1)$ astfel:

$$S_j = D_{j-1} \tag{1}$$

$$D_j = S_{j-1} \oplus f(D_{j-1}, k_j), \tag{2}$$

unde $\oplus$ este operaţia sau exclusiv, f este funcţia rezultată din operaţiile dintr-un ciclu.

Aceste ecuaţii arată că rezultatul fiecărui ciclu depinde numai de ciclul precedent.

Descriind ecuaţiile pentru D_{j-1} şi S_{j-1} avem :

$$D_{j-1} = S_j \tag{3}$$

şi

$$S_{j-1} = D_j \oplus f(D_{j-1}, k_j), \quad (4)$$

Înlocuind (3) în (4) avem:

$$S_{j-1} = D_j \oplus f(S_j, k_j), \quad (5)$$

Ecuaţiile (3) şi (5) arată că aceleaşi valori pot fi obţinute în cicluri ulterioare. Această proprietate face algoritmul DES reversibil.

Deci putem face criptarea unor date şi decriptarea lor folosind acelaşi algoritm, făcând observaţia că la decriptare cheia se ia în ordine inversă.

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

Criptanaliza DES. Deşi DES a fost cel mai celebru algoritm al secolului XX este considerat la această oră nesigur pentru multe aplicaţii. Pare paradoxal, dar aceasta este consecinţa măririi considerabile a puterii de calcul de la confirmarea DES – ului ca un standard criptografic şi până in anul 2000. Slăbiciunea pleacă de la lungimea prea mică a cheii de 56 de biţi. Varianta algoritmului cunoscută ca triplu-DES este cea care este considerată sigură şi la această oră.

Insecuritatea DES-ului pleacă de la premiza că un atac "în forţă" are şanse de reuşită în condiţiile puterii de calcul disponibile astăzi; până în 2004 cel mai eficient atac este datorat criptanalizei liniare care folosind 2^{43} texte cunoscute generează o complexitate temporală de 2^{39-43} (Junod 2001); în condiţiile unui atac cu text ales complexitatea poate fi redusă de patru ori (Knudsen şi Mathiassen, 2000).

Variante de DES

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

2DES constă în două acționări succesive ale algoritmului DES, cu două chei DES diferite sau egale. Utilizarea sa este pur didactică deoarece este vulnerabilă la atacurile cu întâlnire la mijloc (*Meet-in-the-middle*

attack). 2DES este exemplul cel mai des folosit pentru a demonstra viabilitatea unui astfel de atac, dar valoarea sa practică este aproape de zero.

Utilizarea a trei paşi este esenţială pentru a evita atacurile cu întâlnire la mijloc care sunt eficiente împotriva criptării cu 2DES. Mulţimea funcţiilor de criptare DES cu toate cheile posibile nu formează cu operaţiunea de compunere a funcţiilor o structură matematică de grup; dacă ar fi fost aşa, construcţia 3DES ar fi fost echivalentă cu o operaţiune DES şi astfel, la fel de nesigură ca aceasta. Deoarece DES nu este un grup, textul cifrat rezultat este mult mai greu de spart folosind căutarea exhaustivă (forţa brută): 2^{112} încercări în loc de 2^{56} încercări.

Cea mai simplă variantă de 3DES funcţionează astfel (figura 7.4):

$$C = DES(k_3; DES(k_2; DES(k_1; M))),$$

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

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

şi 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.

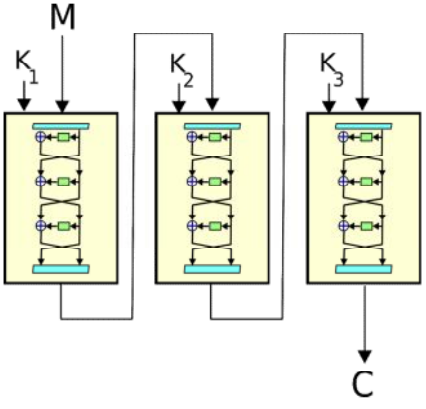


Figura 7.4. *Schema algoritmului 3DES*

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

DES cu sub-chei independente. O altă variantă de DES constă în folosirea unei sub-chei diferite pentru fiecare trecere, în loc de a o genera dintr-o singură cheie de 56 de biţi. Deoarece în fiecare din cele 16 treceri se foloseşte o cheie de 48 de biţi, rezultă că lungimea cheii pentru această variantă este de 768 biţi, ceea ce va creşte semnificativ dificultatea unui atac în forţă împotriva algoritmului, acesta având complexitatea de 2^{768}.

Totuşi, un atac de tip "întâlnire la mijloc" este posibil, ceea ce reduce complexitatea atacului la 2^{384}; încă destul de lung pentru orice nevoie imaginabilă de securitate.

Această variantă poate fi analizată folosind criptanaliza diferenţială şi poate fi spartă cu 2^{61} texte în clar date. Se pare că nici o modificare în planificarea cheilor nu conduce la întărirea semnificativă a algoritmului DES.

DESX este o variantă DES dezvoltată de RSA Data Security, care a fost inclusă încă din 1968 în programul de securitate pentru poştă electronică MailSafe. DESX foloseşte o tehnică numită albire, pentru a ascunde intrările şi ieşirile DES. În plus faţă de cheia DES de 56 de biţi, DESX are o cheie suplimentară de albire de 64 de biţi. Aceşti 64 de biţi sunt operaţi XOR cu textul în clar înainte de prima trecere DES. 64 de biţi suplimentari, calculaţi ca o funcţie bijectivă de toţi cei 120 de biţi ai cheii DES, sunt operaţi XOR cu textul cifrat înaintea ultimei treceri. Albirea îl face pe DESX mult mai puternic decât DES faţă de un atac în forţă; atacul necesită $(2^{120})/n$ operaţii cu n texte în clar cunoscute. De asemenea se îmbunătăţeşte securitatea împotriva criptanalizei liniare şi diferenţiale – atacul necesită 2^{61} texte în clar date şi 2^{60} de texte în clar cunoscute.

CRYPT(3) este o variantă de DES întâlnită în sistemele UNIX. Este folosită în mod obişnuit pentru parole, dar uneori şi pentru criptare. Diferenţa între CRYPT(3) şi DES este că CRYPT(3) are o permutare de

[2] National Institute of Standards and Technology

chei cu 2^{12} posibilităţi, astfel încât să nu permită folosirea cipurilor DES la construcţia unui dispozitiv hardware de spart parole.

DES-ul generalizat (*GDES*) a fost proiectat să mărească viteza DES-ului şi să întărească algoritmul. Mărimea totală a blocului creşte, în timp ce suma calculelor rămâne constantă.

GDES operează pe blocuri de text clar de lungime variabilă. Blocurile criptate sunt împărţite în *q* sub-blocuri; numărul exact depinde de mărimea totală a blocului. În general *q* este egal cu lungimea blocului împărţită la 32.

Funcţia *f* este calculată o dată la fiecare trecere, pe ultimul bloc din dreapta. Rezultatul este operat XOR cu toate celelalte părţi, care sunt apoi rotite spre dreapta. GDES are un număr variabil de treceri, *n*. Exista o mică modificare la ultima trecere, astfel încât procesele de criptare şi decriptare diferă doar prin ordinea sub-cheilor. De fapt, pentru q=2 şi n=16 se obţine algoritmul DES.

Biham şi Shamir arată că, folosind criptanaliza diferenţială, GDES cu q=8 şi n=16 este vulnerabil cu doar şase texte în clar date. Dacă se folosesc şi sub-chei independente, sunt necesare 16 texte în clar date. Pentru q=8 şi n=64, GDES e mai slab decât DES; sunt necesare 2^{49} texte în clar date pentru a-l sparge. De fapt, orice schemă GDES este mai rapidă decât DES, dar este de asemenea mai puţin sigură.

RDES este o variantă care înlocuieşte schimbarea stânga-dreapta de la sfârşitul fiecărei treceri cu o schimbare dependentă de cheie. Schimbările sunt fixe, depinzând doar de cheie. Aceasta înseamnă că cele 15 schimbări dependente de cheie se petrec cu 2^{15} posibilităţi şi că această variantă nu rezistă la criptanaliza diferenţială.

O idee mai bună este ca schimbarea să aibă loc doar în partea dreaptă, la începutul fiecărei treceri, iar schimbarea să depindă de datele de intrare şi nu de cheie. În RDES-1 se practică o schimbare dependentă de date de cuvinte pe 16 biţi la începutul fiecărei treceri. În RDES-2 există o schimbare de octeţi dependentă de date la începutul fiecărei treceri, după o schimbare ca în RDES-1. Se poate continua în acelaşi mod până la RDES-4. RDES-1 este sigură atât faţă de criptanaliza liniară cât şi faţă de cea diferenţială.

În final este prezentată o istorie cronologică a algoritmului de criptare DES (tabelul 7.2).

Data	Anul	Evenimentul
15 mai	1973	NBS publică prima cerere pentru un algoritm standard pentru criptare
27 august	1974	NBS publică a doua cerere pentru un algoritm standard pentru criptare
17 martie	1975	DES este publicat în *Federal Register*[3] pentru comentarii
august	1976	Se organizează primul workshop despre DES
septembrie	1976	Al doilea workshop despre fundamentele matematice ale DES-ului
noiembrie	1976	DES este aprobat ca un standard
15 ianuarie	1977	DES este publicat în FIPS PUB 46
	1983	DES este reconfirmat pentru prima dată
22 ianuarie	1988	DES este reconfirmat pentru a doua oară ca FIPS 46-1
	1992	Biham şi Shamir publică primul atac teoretic cu o complexitate mai mică decât atacul "în forţă brută" : criptanaliza diferenţială ; metoda cerea un număr nerealist (2^{47}) de texte alese
30 decembrie	1993	DES este reconfirmat pentru a treia oară ca FIPS 46-2
	1994	Prima criptanaliză experimentală folosind criptanaliza liniară (Matsui, 1994)
iunie	1997	Proiectul DESCHALL sparge pentru prima dată în public un mesaj criptat cu DES
iulie	1998	EFF găseşte o cheie pentru DES în 56 de ore
ianuarie	1999	EFF folosind putere de calcul distribuită găseşte o cheie pentru DES în 22 de ore şi 15 minute
25 octombrie	1999	DES este reconfirmat pentru a patra oară ca FIPS 46-3 cu specificaţia preferinţei pentru Triplu DES
26 noiembrie	2001	AES este publicat în FIPS 197
26 mai	2002	Standardul AES devine efectiv
26 iulie	2004	Retragerea standardului FIPS 46-3 (şi a celor conexe) este propusă în *Federal Register*

Tabelul 7.2 *Cronologia evenimentelor algoritmului DES*

[3] Publicaţie a NIST

Tema 8. Cifrul 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* (Standard de Criptare Avansat). Regulile concursului erau:

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

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

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

În octombrie 2000 NIST a stabilit câştigătorul. Acesta este algoritmul Rijndael, dezvoltat de doi tineri cercetători belgieni, Joan Daemen şi Vincent Rijmen şi care devine standard guvernamental al SUA. Se spera ca Rjindael să devină standardul criptografic dominant în lume pentru următorii 10 ani.

Rijndael permite lungimi de chei şi mărimi de blocuri de la 128 de biţi la 256 de biţi, în paşi de câte 32 de biţi. Lungimea cheii şi lungimea blocului pot fi alese în mod independent, dar în practică se presupunea folosirea a două variante: bloc de 128 biţi cu cheie de 128 biţi şi bloc de 128 biţi cu cheie de 256 biţi. Standardul comercial trebuia să devenină cel mai probabil varianta 128/128. O cheie de 128 biţi permite un spaţiu al cheilor de 2^{128} chei.

Preliminarii matematice. Rijndael se bazează pe teoria câmpului Galois, în sensul că anumite operaţiuni sunt definite la nivel de octet iar octeţii reprezintă elemente în câmpul finit GF(2^8).

Cum toate reprezentările câmpului finit GF(2^8) sunt izomorfe, se poate alege reprezentarea clasică polinomială, cu impact pozitiv asupra complexităţii implementării.

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

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

Operaţia de adunare este definită ca suma a două polinoame în care coeficienţii se adună modulo 2 şi care corespunde operării XOR a celor doi octeţi corespondenţi. Sunt îndeplinite axiomele grupului abelian: operaţia este internă, asociativă, comutativă, există element neutru şi element invers.

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

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

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

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

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

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

Ca şi DES, AES foloseşte substituţii şi permutări, ca şi runde multiple. Numărul de runde depinde de mărimea cheii şi de mărimea blocului, fiind 10 în cazul 128/128 şi mărindu-se până la 14 pentru cazul 256/128. Spre deosebire de DES, toate operaţiile sunt la nivel de octet, pentru a permite implementări hardware şi software eficiente.

Descrierea AES

În algoritmul AES rezultatul cifrat intermediar este numit vector *state*, care poate fi reprezentat ca un tabel cu patru linii şi patru coloane, acestea fiind numerotate începând de la 0.

Vectorul *state* se iniţializează cu blocul de 128 biţi de text clar (în ordinea coloanelor, cu primii patru octeţi în coloana 0) şi va fi modificat la fiecare pas al calculului, prin substituţii, permutări şi alte transformări, rezultând în final blocul de 128 biţi de text cifrat.

Cheia de 128 de biţi este expandată în 11 tabele 4x4 notate *rk*(0), *rk*(1),...., *rk*(10). Expandarea este realizată prin rotiri repetate şi operaţii XOR asupra unor grupuri de biţi din cheia originală.

Înainte de a începe cele 10 runde, cheia *rk*(0) se operează XOR cu vectorul *state.*

Calculul principal constă în execuţia a 10 runde, folosind cheia *rk*(*i*) la iteraţia *i*. Fiecare rundă constă în patru paşi.

Pasul 1 realizează o substituţie octet cu octet asupra vectorului *state* folosind o cutie *S*.

Pasul 2 roteşte la stânga fiecare din cele 4 rânduri ale vectorului *state*: rândul 0 este rotit cu 0 octeţi, rândul 1 este rotit cu 1 octet, rândul 2 este rotit cu 2 octeţi şi rândul 3 este rotit cu 3 octeţi, realizând difuzia datelor.

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

În fine, pasul 4 operează XOR cheia *rk* din runda respectivă cu vectorul *state*.

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.

Avantajele AES relativ la implementare sunt:

- AES se poate implementa pe un procesor Pentium Pro şi va rula cu o viteză mai mare decât orice alt cifru bloc;
- AES se poate implementa pe un dispozitiv Smart Card, folosind un spaţiu redus de memorie RAM şi un număr redus de cicluri;

- transformarea din cadrul unei runde este paralelă prin proiectare, ceea ce constituie un avantaj pentru viitoarele procesoare;
- AES nu foloseşte operaţii aritmetice, ci doar operaţii la nivel de şiruri de biţi.

Simplitatea proiectării AES:

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

Lungimea variabilă a blocului

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

Extensii:

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

Limitările AES sunt în legătură cu algoritmul de decriptare:

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

Iteraţiile algoritmului AES

Fiecare iteraţie obişnuită se efectuează în 4 paşi.

Primul pas este cel de substituire al octeţilor, *the Byte Sub* (*BS*) *step* (figura 8.1).

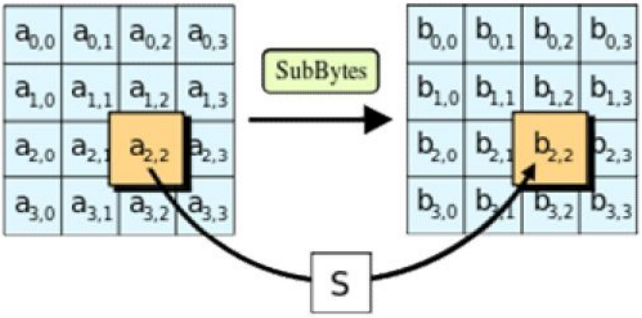


Figura 8.1. *Transformarea Byte Sub*

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 matricea ilustrată în Tabelul 8.1.

Cel de al **doilea pas** al unei iteraţii uzuale se numeşte deplasarea liniilor, *the Shift Row* (*SR*) *step* (Figura 8.2).

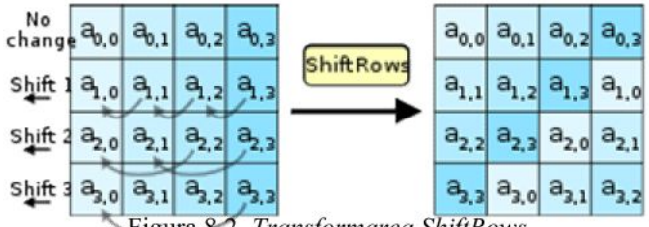


Figura 8.2. *Transformarea ShiftRows*

Considerând că blocul care trebuie construit este alcătuit cu octeţii numerotaţi de la 1 la 16, aceşti octeţi se aranjează într-un dreptunghi şi se deplasează după cum urmează:

De la		la	
	1 5 9 13		1 5 9 13
	2 6 10 14		6 10 14 2
	3 7 11 15		11 15 3 7
	4 8 12 16		16 4 8 12

99 124 119 123 242 107 111 197

48	1	103	43	254	215	171	118
202	130	201	125	250	89	71	240
173	212	162	175	156	164	114	192
183	253	147	38	54	63	247	204
52	165	229	241	113	216	49	21
4	199	35	195	24	150	5	154
7	18	128	226	235	39	178	117
9	131	44	26	27	110	90	160
82	59	214	179	41	227	47	132
83	209	0	237	32	252	177	91
106	203	190	57	74	76	88	207
208	239	170	251	67	77	51	133
69	249	2	127	80	60	159	168
81	163	64	143	146	157	56	245
188	182	218	33	16	255	243	210
205	12	19	236	95	151	68	2332
196	167	126	61	100	93	25	115
96	129	79	220	34	42	144	136
70	238	184	20	222	94	11	219
224	50	58	10	73	6	36	92
194	211	172	98	145	149	228	121
231	200	55	109	141	213	78	169
108	86	244	234	101	122	174	8
186	120	37	46	28	166	180	198
232	221	116	31	75	189	139	138
112	62	181	102	72	3	246	14
97	53	87	185	134	193	29	158
225	248	152	17	105	217	142	148
155	30	135	233	206	85	40	223
140	161	137	13	191	230	66	104
65	153	45	15	176	84	187	22

Tabelul 8.1. *Matricea pentru cutia de tip S*

Cel de al **treilea pas** al algoritmului de criptare Rijndael este numit amestecarea coloanelor, *the Mix Column* (*MC*) *step* (Figura 8.3).

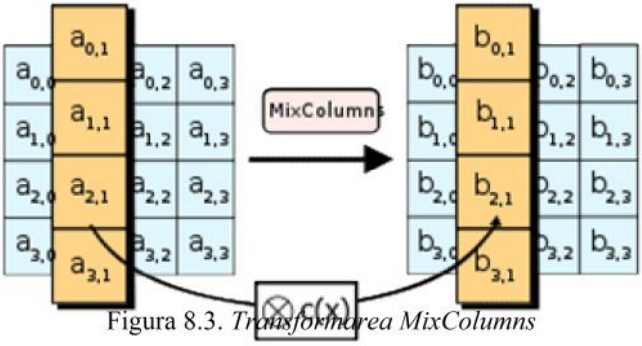


Figura 8.3. *Transformarea MixColumns*

Acest pas se realizează prin înmulţire matricială: fiecare coloană, în aranjamentul pe care l-am observat, este înmulţită cu matricea:

2 3 1 1
1 2 3 1
1 1 2 3
3 1 1 2

Această înmulţire matricială corespunde unei înmulţiri specifică câmpului Galois al lui 2^8, definită de polinomul modul

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

Octeţii care trebuie înmulţiţi sunt priviţi ca şi polinoame şi nu ca şi numere. De exemplu prin înmulţirea unui octet cu 3 se obţine rezultatul operaţiei sau-exclusiv dintre acel octet şi varianta sa obţinută prin rotirea acelui octet cu o poziţie la stânga. Dacă rezultatul acestei înmulţiri are mai mult de 8 biţi, biţii suplimentari nu sunt pur şi simplu ignoraţi. Pentru eliminarea lor se calculează sau-exclusiv între rezultatul obţinut în urma „înmulţirii" deja efectuate (deplasat la stânga dacă este necesar) şi şirul binar cu lungimea de 9 biţi **100011011** (care corespunde polinomului modul).

Cel de al **patrulea pas** al algoritmului Rijndael este cel de adăugare a sub-cheii, *the Add Round Key step* (Figura 8.4).

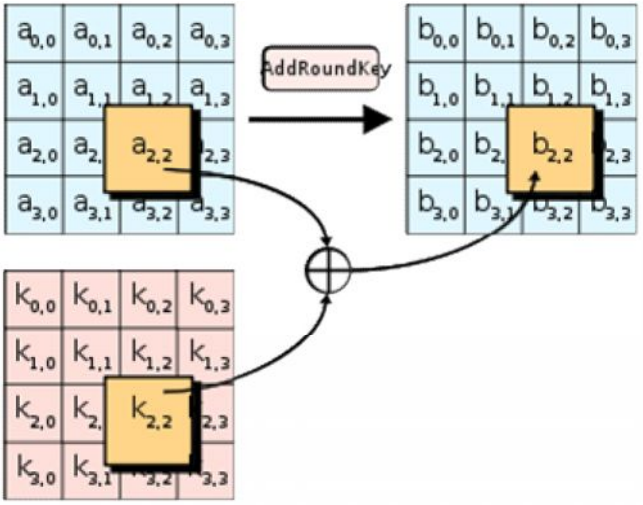


Figura 8.4. *Transformarea Add Round Key*

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

Din ultima iteraţie este omis pasul de amestecare a coloanelor.

Decriptarea în AES

Pentru a decripta mesajul fabricat de algoritmul Rijndael este necesar ca operaţiile descrise să fie înlocuite cu operaţiile lor inverse şi ca acestea să fie aplicate în ordine inversă (prima operaţie din algoritmul de decriptare trebuie să fie inversa ultimei operaţii din algoritmul de criptare). Succesiunea paşilor în algoritmul Rijndael este:

ARK,
BS – SR – MC – ARK,
BS – SR – MC – ARK ,
...,
BS – SR – MC – ARC,
BS – SR – ARC

Deşi această secvenţă nu este simetrică, ordinea unor operaţii poate fi modificată fără ca procesul de criptare să fie afectat. De exemplu pasul BS de substituire a octeţilor (notat cu B în continuare), poate fi la fel de bine făcut şi după pasul SR de deplasare a liniilor (notat cu S în continuare). Această observaţie este utilă pentru procesul de decriptare. Făcând această inversare secvenţa algoritmului, de forma:

A BSMA BSMA ... BSMA BSA

se transformă într-o secvenţă de forma:

A SBMA SBMA ... SBMA SBA (1R)

Pentru fiecare pas s-a folosit notaţia bazată pe prima literă a denumirii engleze a pasului. Dacă se inversează secvenţa care descrie algoritmul se obţine:

ASB AMSB ... AMSB AMSB A (2R)

Comparând secvenţele (1R) şi (2R) se constată că pe lângă diferita poziţionare a spaţiilor (acestea marchează începutul unei noi iteraţii a algoritmului de criptare) singura diferenţă care mai apare este că grupurile „MA" din (1R) sunt înlocuite cu grupuri „AM" în (2R).

E clar că nu este suficientă inversarea ordinii paşilor folosiţi la criptare pentru a se face decriptarea ci trebuie inversate şi operaţiile care compun aceşti paşi. Pentru inversarea pasului ARK trebuie inversată funcţia sau-exclusiv. Dar această inversare se realizează tot cu funcţia sau-exclusiv. De aceea pasul ARK nu trebuie inversat la decriptare. Nu acelaşi lucru se poate spune despre ceilalţi paşi. Este de exemplu cazul pasului de amestecare a coloanelor, MC, (notat cu M în relaţiile (1R) şi (2R)) pentru inversarea căruia, în procesul de decriptare este necesară inversarea matricei cu care se înmulţeşte fiecare vector. La fel trebuie procedat şi cu matricea cutiei de tip S din pasul de substituţie a octeţilor, BS (notat cu B în relaţiile (1R) şi (2R)).

Revenind la relaţiile (1R) şi (2R), deoarece operaţia de înmulţire a matricelor este distributivă în raport cu operaţia de adunare pe câmpul Galois al lui 2^8, nu trebuie inversată ordinea secvenţei paşilor „MA" şi „AM" pentru decriptare. Operaţia de sau-exclusiv din cadrul pasului MC (M) este de fapt identică cu operaţia de adunare definită pe câmpul Galois al lui 2^8. De aceea cheile de iteraţie, implicate în procesul de inversare al pasului de amestecare a coloanelor, trebuiesc înmulţite cu inversa matricei de amestecare a coloanelor şi apoi se pot calcula funcţiile sau-exclusiv, la fel ca la criptare (bineînţeles cheile de iteraţie trebuiesc luate în ordine

inversă în raport cu ordinea folosită la criptare). Matricea pentru inversarea pasului de amestec al coloanelor este:

$$\begin{matrix} 14 & 11 & 13 & 9 \\ 9 & 14 & 11 & 13 \\ 13 & 9 & 14 & 11 \\ 11 & 13 & 9 & 14 \end{matrix}$$

iar forma sa binară, folosită în algoritmul de decriptare este:

1110	1011	1101	1001	01 00 00 00
1001	1110	1011	1101	00 01 00 00
1101	1001	1110	1011	00 00 01 00
1011	1101	1001	1110	00 00 00 01
111	101	110	100	01 01 00 00
110	100	111	101	00 00 01 01
1100	1000	1110	1010	00 00 10 10
1011	1101	1000	1110	01 01 10 10
0	0	1	0	01 01 10 11

Generarea cheilor în AES

Pentru cazul în care se foloseşte o cheie iniţială cu lungimea de 128 biţi sau de 192 de biţi, toate subcheile necesare pentru toate iteraţiile, se obţin din cheia iniţială (prima subcheie fiind chiar cheia iniţială) sau din variante ale cheii iniţiale şi au aceeaşi lungime cu aceasta. Subcheile sunt alcătuite din cuvinte de 4 octeţi. Fiecare cuvânt se obţine calculând sau-exclusiv între cuvântul anterior de 4 octeţi şi cuvântul corespunzător dintr-o variantă anterioară sau rezultatul aplicării unei funcţii acestui cuvânt (din varianta precedentă). Pentru stabilirea primului cuvânt dintr-o anumită variantă, cuvântul iniţial (cel curent pentru iteraţia respectivă) este pentru început rotit cu opt poziţii spre stânga, apoi octeţii săi sunt modificaţi folosind cutia de tip S din pasul BS (B) de substituţie a biţilor corespunzător, iar apoi se calculează sau exclusiv între primul octet al rezultatului obţinut anterior şi o constantă dependentă de iteraţie. Constantele dependente de iteraţie sunt puterile lui 2 succesive în reprezentarea din câmpul Galois al lui 2^8 folosit:

1 2 4 8 16 32 64 128
27 54 108 216 171 77 154 47
94 188 99 198 151 53 106 212
179 125 250 239 197 145 57 114
228 211 189 97...

sau în binar:

00000001 00000010 00000100 00001000 00010000 00100000
01000000 10000000
00011011 00110110 01101100 11011000 10101011 01001101
10011010 00101111
01011110 10111100 01100011 11000110 10010111 00110101
01101010 11010100
10110011 01111101 11111010 11101111 11000101 10010001
00111001 01110010
11100100 11010011 10111101 01100001...

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

Atacul cel mai realizabil împotriva AES este îndreptat împotriva variantelor Rijndael cu număr redus de iterații. AES are 10 iterații la o cheie de 128 de biți, 12 la cheie de 192 de biți și 14 la cheie de 256 de biți. La nivelul anului 2008, cele mai cunoscute atacuri erau accesibile la 7, 8, respectiv 9 iterații pentru cele trei lungimi ale cheii.

Deoarece căutarea permanentă rezultă cu succese pentru criptanaliști, pentru siguranţă e recomandată (B. Schneier) trecerea de la 10 la 16 runde pentru AES-128, de la 12 la 20 pentru AES 192 şi de la 14 la 28 pentru AES-256.

[4] National Security Agency

Tema 9. Algoritmi simetrici de tip şir (stream cypher)

Cifrurile şir (sau cifruri *fluide*) formează o clasă importantă de algoritmi de criptare; ele pot fi cifruri cu chei simetrice sau cu chei publice. Ceea ce le caracterizează şi le diferenţiază faţă de cifrurile bloc este faptul că cifrurile şir procesează textul clar în unităţi oricât de mici, chiar bit cu bit, aplicând funcţia XOR între biţii cheii şi biţii de cifrat, iar funcţia de criptare se poate modifica pe parcursul criptării. Cifrurile şir sunt algoritmi cu memorie, în sensul că procesul de criptare nu depinde doar de cheie şi de textul în clar, ci şi de starea curentă. În cazul în care probabilitatea erorilor de transmisie este mare, folosirea cifrurilor şir este avantajoasă deoarece au proprietatea de a nu propaga erorile. Ele se folosesc şi în cazurile în care datele trebuie procesate una câte una, datorită lipsei de spaţiu de memorie.

Reamintim că într-un sistem de criptare bloc elementele succesive ale textului clar sunt criptate folosind aceeaşi cheie $\boldsymbol{k}$. Adică dacă $M = m_1 m_2 m_3 ...$ este textul clar, atunci textul criptat este $C = c_1 c_2 c_3 ... = e_k(m_1) e_k(m_2) e_k(m_3) ...$.

Definiţie. Fie (***P***, ***C***, ***K***, ***E***, ***D***) un sistem de criptare. O secvenţă de simboluri $k k_1 k_2 k_3 ... \in K$ se numeşte cheie fluidă.

Definiţie. Un sistem de criptare (***P***, ***C***, ***K***, ***E***, ***D***) care criptează un text clar

$$M = m_1 m_2 m_3 ...$$

în

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

se numeşte sistem fluid de criptare, unde $k_1 k_2 k_3 ... \in K$ este o cheie fluidă.

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

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

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

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

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

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

Printre proprietăţile cifrurilor şir sincrone se numără:

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

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

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

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

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

Cifrurile şir asincrone posedă următoarele proprietăţi:

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

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

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

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

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

Criptografii încearcă să obţină o complexitate liniară ridicată prin combinarea ieşirilor mai multor LFSR-uri într-un mod nonliniar. Pericolul este ca unul sau mai multe şiruri interne generate – de obicei ieşiri ale LFSR-urilor individuale – să fie corelate cu şirul combinat, ceea ce permite un atac bazat pe algebra liniară numit *atac de corelaţie*. Thomas Siegenthaler a arătat că imunitatea de corelare poate fi precis definită şi că există o legătură între aceasta şi complexitatea liniară. Ideea de bază a atacului de corelaţie este identificarea unor corelaţii între ieşirea generatorului şi ieşirea uneia din componentele sale interne. Apoi, observând şirul de ieşire, se pot obţine informaţii despre ieşirea internă. Folosind aceste informaţii şi alte corelaţii se colectează informaţii despre celelalte ieşiri interne ale generatorului, până când acesta este spart în totalitate.

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

Cifrul SEAL (*Software-Optimized Encryption ALgorithm*) este un sistem de criptare aditiv binar (adică are la bază operaţia $\oplus$ - XOR), a fost elaborat în 1993 de către Phillip Rogaway şi Don Coppersmith. Este unul din cele mai eficiente sisteme implementabile pe procesoare de 32 biţi.

SEAL este o funcţie pseudo-aleatoare care scoate o cheie fluidă de L biţi folosind un număr n de 32 biţi şi o cheie secretă a de 160 biţi.

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

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

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

În primul rând trebuie de generat tabelele T, R şi S, fiecare din care este în funcţie de cheia a. Unica misiune a cheii a în algoritm este de a genera aceste tabele prin intermediul funcţiei G, construite în baza cunoscutului algoritm al funcţiei hash SHA-1 care este un standard de stat în SUA.

Algoritmul de generare a tabelei G pentru SEAL 2.0 este următorul (Figura 9.1):

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

1. Se definesc constantele (de 32 biţi):

$$y_1 = 0x5a827999,$$
$$y_2 = 0x6ed9eba1,$$
$$y_3 = 0x8f1bbcdc,$$
$$y_4 = 0xca62c1d6$$

2. *a*. $X_0 \leftarrow i$;
b. *for* $j \leftarrow 1$ *to* 15 *do* $X_j \leftarrow 0x00000000$;
c. *for* $j \leftarrow 16$ *to* 79 *do* $X_j \leftarrow ((X_{j-3} \oplus X_{j-8} \oplus X_{j-14} \oplus X_{j-16}) << 1)$;
d. $(A, B, C, D, E) \leftarrow (H_0, H_1, H_2, H_3, H_4)$ *where* $a = H_0 H_1 H_2 H_3 H_4$;
e. (Runda 1): *for* $j \leftarrow 0$ *to* 19 *do*
$t \leftarrow ((A << 5) + f(B, C, D) + E + X_j + y_1)$;
$(A, B, C, D, E) \leftarrow (t, A, B << 30, C, D)$;
f. (Runda 2): *for* $j \leftarrow 20$ *to* 39 *do*
$t \leftarrow ((A << 5) + h(B, C, D) + E + X_j + y_2)$;
$(A, B, C, D, E) \leftarrow (t, A, B << 30, C, D)$;
g. (Runda 3): for j $\leftarrow$ 40 to 59 do
$t \leftarrow ((A << 5) + h(B, C, D) + E + X_j + y_3)$;
$(A, B, C, D, E) \leftarrow (t, A, B << 30, C, D)$;
h. (Runda 4): for j $\leftarrow$ 60 to 79 do
$t \leftarrow ((A << 5) + h(B, C, D) + E + X_j + y_4)$;
$(A, B, C, D, E) \leftarrow (t, A, B << 30, C, D)$;
i. $(H_0, H_1, H_2, H_3, H_4) \leftarrow (H_0 + A, H_1 + B, H_2 + C, H_3 + D, H_4 + E)$;
$G_a(i) = H_0 \| H_1 \| H_2 \| H_3 \| H_4$.

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

Intrare: *a* – cheia secretă (160 biţi), $n \in [0, 2^{32})$ – întreg,
L - lungimea solicitată pentru cheia fluidă.

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

1. Se generează tabelele *T*, *S*, *R* având ca elemente cuvinte de 32 biţi. Fie funcţia $F_a(i) = H^i_{i(\mathrm{mod}5)}$ unde $H^i_0 H^i_1 H^i_2 H^i_3 H^i_4 = G_a(\lfloor i/5 \rfloor)$.

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) şi l (întreg) şi ieşirile A, B, C, D, n_1, n_2, n_3, n_4 (cuvinte).

a. $A \leftarrow n \oplus R[4 \cdot l]$, $B \leftarrow (n >> 8) \oplus R[4 \cdot l + 1]$, $C \leftarrow (n >> 16) \oplus R[4 \cdot l + 2]$,
$D \leftarrow (n >> 24) \oplus R[4 \cdot l + 3]$;

b. *for* $j \leftarrow 1$ *to* 2 *do*
$P \leftarrow A \wedge 0x000007fc$, $B \leftarrow B + T[P/4]$, $A \leftarrow (A >> 9)$,

$P \leftarrow B \wedge 0x000007fc$, $C \leftarrow C + T\,[P/4]$, $B \leftarrow (B >> 9)$,
$P \leftarrow C \wedge 0x000007fc$, $D \leftarrow D + T\,[P/4]$, $C \leftarrow (C >> 9)$,
$P \leftarrow D \wedge 0x000007fc$, $A \leftarrow A + T\,[P/4]$, $D \leftarrow (D >> 9)$,
$(n_1, n_2, n_3, n_4) \leftarrow (D, A, B, C)$;
$P \leftarrow A \wedge 0x000007fc$, $B \leftarrow B + T\,[P/4]$, $A \leftarrow (A >> 9)$,
$P \leftarrow B \wedge 0x000007fc$, $C \leftarrow C + T\,[P/4]$, $B \leftarrow (B >> 9)$,
$P \leftarrow C \wedge 0x000007fc$, $D \leftarrow D + T\,[P/4]$, $C \leftarrow (C >> 9)$,
$P \leftarrow D \wedge 0x000007fc$, $A \leftarrow A + T\,[P/4]$, $D \leftarrow (D >> 9)$;

3. $l \leftarrow 0$, $y \leftarrow \lambda$ (şirul 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*

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

c. $l \leftarrow l + 1$.

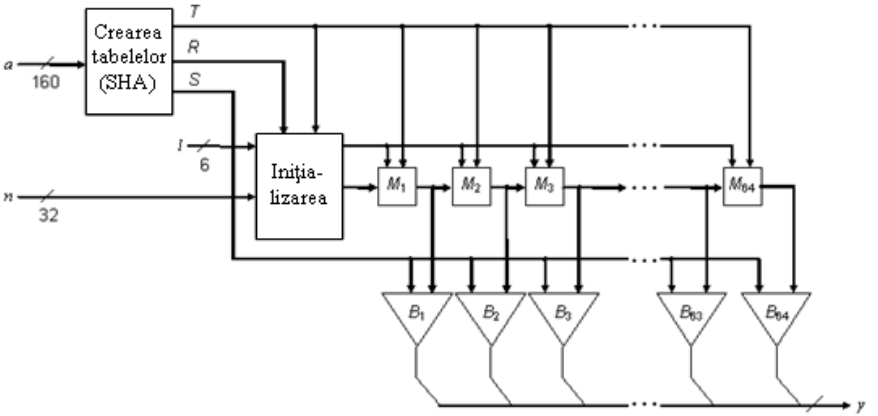


Figura 9.1. *Schema ciclului intern SEAL*

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

Algoritmul *SEAL* este relativ nou şi la momentul actual nu sunt publicate metode eficiente de spargere a acestui algoritm. În anul 1997 a fost publicat o metodă de atac bazată pe Criptanaliza χ^2 care permite determinarea unei părţi mari din tabelele interne, însă era aplicabilă numai la o versiune simplificată a lui *SEAL*. Trebuie de menţionat aici că Don Coppersmith (care este şi coautor al lui DES) este şi unul dinte cei mai abili criptanalişti.

Cifrul A5 este un cifru stream folosit pentru a cripta fluxul de date *GSM* (Group Special Mobile), reprezentând standardul non-american pentru telefonia mobilă celulară. A5 criptează linia dintre telefon şi celula de bază, restul legăturii rămânând necriptată. A5 este format din trei LFSR-uri, care au registre de lungime 19, 22 şi respectiv 23. Toate polinoamele de feedback sunt cu un număr redus de coeficienţi. Ieşirea este obţinută prin operarea XOR a celor trei LFSR-uri. A5 foloseşte un clock control variabil. Fiecare registru face un clocking bazat pe bitul central, care este operat XOR cu inversa funcţiei prag (threshold function)

a biţilor de la mijlocul celor trei registre. Un registru este clock-at dacă bitul său de clocking (orange) este în concordanţă cu unul sau ambii biţi de clocking a celorlalte două registre (Figura 9.2). În mod normal, două din LFSR-uri sunt clock-ate la fiecare iteraţie.

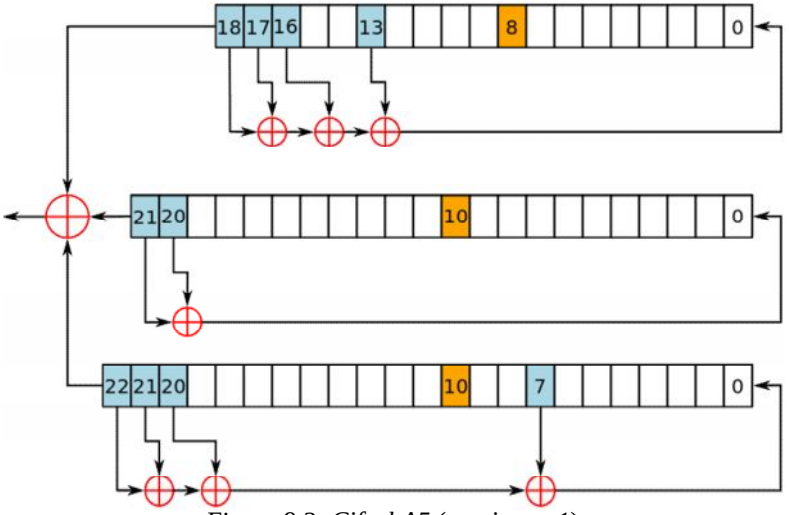


Figura 9.2. *Cifrul A5* (*versiunea 1*)

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

Cifrul RC4 este un cifru şir cu cheie de lungime variabilă, dezvoltat în 1987 de către Ron Rivest pentru RSA Data Security. În 1994 codul sursă al algoritmului este făcut public pe Internet. RC4 este un algoritm simplu de descris: şirul cheie este independent de textul clar. Funcţionează în baza „cutiilor-*S*”: S_0, S_1, ..., S_{255}. Intrările sunt permutări ale numerelor de la 0 la 255, iar permutarea este o funcţie de o cheie de lungime variabilă. Există doi indici, *i* şi *j*, iniţializaţi cu zero. Pentru a genera un octet aleator se procedează astfel:

$i = (i + 1) \bmod 256;$
$j = (j + S_i) \bmod 256;$
$T = S_i; S_i = S_j; S_j = T;$
$t = (S_i + S_j) \bmod 256;$
$K = S_t$.

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

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

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

şi un alt vector de 256 de octeţi cu cheia, repetând cheia, dacă este necesar, pentru a completa vectorul cu componentele:

$K_0, K_1, \ldots, K_{255}$.

$j = 0;$

For $i = 0$ to 255:

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

se schimbă S_i cu S_j între ele (Figura 9.3).

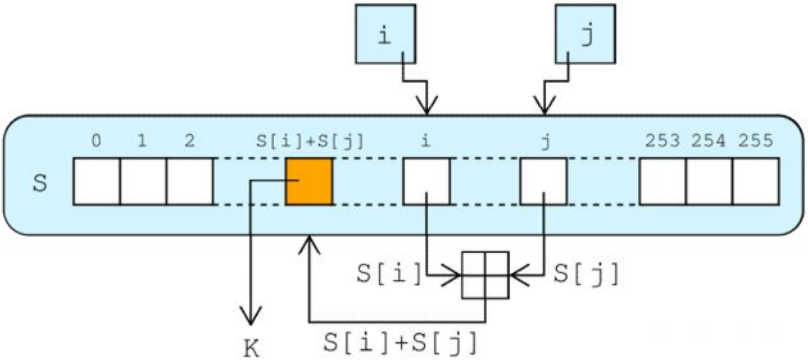


Figura 9.3. *Generatorul de chei al algoritmului RC*4

Nu există rezultate publice ale criptanalizei. Se crede ca algoritmul este imun la analiza diferenţială şi liniară; RC4 poate fi în aproximativ 2^{1700} stări posibile. "Cutiile-S" evoluează lent în timpul întrebuinţării: *i* asigură că fiecare element se schimbă, iar *j* că aceste schimbări sunt aleatoare. RC4 are un statut special de export, acesta fiind permis doar pentru chei de până

la 40 de octeţi. Acest algoritm este implementat în multe produse comerciale, dintre care Lotus Notes şi Oracle Secure SQL.

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

Tema 10. Criptarea cu cheie publică

Conceptul de criptografie cu chei publice a fost inventat de Whitfield Diffie şi Martin Hellman. Contribuţia lor constă în propunerea de a folosi un nou criptosistem în care cheile de criptare şi decriptare sunt diferite, iar cheia de decriptare (care este secretă) nu poate fi dedusă din cheia de criptare (care este publică). În anul 1976 conceptul a fost prezentat în premieră la National Computer Conference SUA, iar câteva luni mai târziu lucrarea a fost publicată.

Sistemele cu cheie publică (sau sisteme asimetrice) au un mare avantaj faţă de sistemele cu chei secrete: oricine poate transmite un mesaj secret utilizatorului (cunoscându-i *cheia publică*), iar mesajul rămâne protejat faţă de interceptor. Cu un sistem cu cheie convenţională pentru fiecare pereche de utilizatori este necesară o cheie separată *secretă* (*privată*).

În general, un sistem cu n utilizatori necesită $\frac{n(n-1)}{2}$ chei, pentru ca oricare pereche de utilizatori să poată comunica între ei şi mesajele lor să rămână secrete faţă de ceilalţi utilizatori. Numărul de chei creşte rapid o dată cu numărul de utilizatori; generarea, distribuirea şi menţinerea securităţii cheilor constituie o problemă datorită numărului lor mare.

Într-un sistem cu cheie publică, un utilizator deţine două chei: o cheie publică şi o cheie privată (secretă). Utilizatorul îşi poate face cunoscută oricui cheia publică. Fie *SK* cheia secretă şi *PK* cheia publică corespunzătoare. Atunci:

$$m=d(SK, e(PK, m)).$$

Utilizatorul poate decripta cu cheia privată ceea ce oricine altcineva a criptat cu cheia publică corespunzătoare.

Cu al doilea algoritm de criptare cu cheie publică

$$m=d(PK, e(SK, m))$$

utilizatorul poate cripta un mesaj cu cheia privată, iar mesajul poate fi decriptat doar cu cheia publică corespunzătoare.

Aceste două proprietăţi presupun că cele două chei, publică şi privată, pot fi aplicate în orice ordine (sistemul RSA nu face distincţie între cheia publică şi cheia privată; orice cheie din perechea de chei poate fi folosită fie ca cheie publică, fie ca cheie privată).

Definiţie. O schemă de criptare cu cheie publică constă în trei algoritmi (Figura 10.1):

1. *Generatorul de chei*, care returnează i pereche cheie secretă-cheie publică (*SK*, *PK*);
2. *Algoritmul de criptare*, care primeşte la intrare un mesaj *m* din mulţimea mesajelor posibile, o cheie publică *PK* şi returnează criptotextul *c*.
3. *Algoritmul de decriptare*, care ia ca intrare un text cifrat *c* din mulţimea textelor cifrate, o cheie secretă *SK* şi returnează un mesaj *m*.

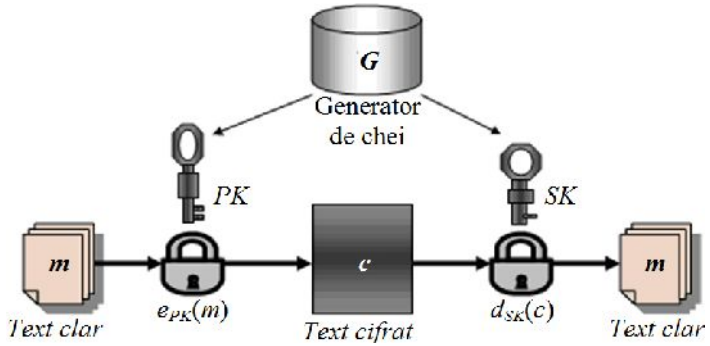


Figura 10.1. *Schema unui algoritm cu cheie publică*

Trebuie de menţionat aici unele condiţii care trebuie respectate:

- Receptorul *B* poate uşor să genereze cheia publică PK_B şi cheia privată SK_B.
- Emiţătorul *A*, ştiind cheia publică a lui *B* şi mesajul clar *m*, poate să genereze textul cifrat corespunzător:
$$c = e_{PK_B}(m).$$
- Receptorul *B* poate uşor să decripteze textul cifrat *c*:
$$m = d_{SK_B}(c) = d_{SK_B}(e_{PK_B}(m)).$$
- Un atacator care ştie PK_B nu poate să determine cheia privată SK_B.
- Un atacator care ştie cheia publică PK_B şi textul cifrat *c* nu poate să determine mesajul original *m*.
- Are loc următoarea relaţie (menţionată mai sus):

$$m = d_{SK_B}(e_{PK_B}(m)) = d_{PK_B}(e_{SK_B}(m)).$$

La momentul actual cele mai cunoscute sisteme de criptare cu cheie publlică sunt:

- Sistemul *RSA*: se bazează pe dificultatea descompunerii în factori primi a numerelor mari (de sute de cifre). Este sistemul cel mai larg utilizat în acest moment.
- Sistemul *El Gamal*: se bazează pe dificultatea calculului logaritmului discret într-un corp finit.
- Sistemul *Merkle-Hellman*: primul sistem definit cu cheie publică, bazat pe problema {0, 1} a rucsacului, problemă *NP* - completă.
- Sistemul *McEliece*: este bazat pe teoria algebrică a codurilor, decodificarea unui cod liniar fiind de asemenea o problemă *NP* - completă.
- *Curbe eliptice*: Sunt sisteme de criptare care îşi desfăşară calculele pe mulţimea punctelor unei curbe eliptice (în locul unui inel finit Z_n).

Atât conceptul criptării cu chei publice cât şi primul algoritm publicat care folosea chei publice a fost dezvoltat de W. Diffie şi M. Hellman. Algoritmul este numit *Diffie-Hellman key exchange* (1976) şi este folosit în numeroase produse comerciale. Acest algoritm nu se aplică la criptarea mesajelor sau la crearea de semnături digitale. Scopul său este *distribuirea cheilor*, adică scopul este ca doi utilizatori să poată schimba o cheie secretă în siguranţă, deci algoritmul este limitat la schimbul cheilor secrete.

Algoritmul Diffie-Hellman este bazat pe problema complicată a determinării logaritmului discret. Să încercăm o explicaţie succintă acestei noţiuni.

Definiţie: Fie G – un grup ciclic de ordinul n şi α – elementul generator al său. Fie β – un element din G. *Logaritmul discret* β în baza α, care se notează cu $\log_\alpha \beta$, este unicul număr întreg a, $0 \le a \le |G-1|$, astfel încât $\beta = \alpha^a$.

Pentru comoditate şi exactitate vom considera G – un grup multiplicativ $\boldsymbol{Z_p^*}$ de ordin $p-1$, unde operaţia este înmulţirea *modulo p*.

Exemplu. Fie $p = 97$. Atunci $\boldsymbol{Z}_{97}^*$ este un grup ciclic multiplicativ de ordinul $n = 96$. Elementul generator al grupului $\boldsymbol{Z}_{97}^*$ este $\alpha = 5$. Deoarece $5^{32} \equiv 35 \pmod{97}$, avem $\log_5 35 = 32$ în $\boldsymbol{Z}_{97}^*$.

Unele proprietăţi ale logaritmilor discreţi:

- $\log_\alpha(\beta\gamma) = (\log_\alpha \beta + \log_\alpha \gamma) \bmod n$;
- $\log_\alpha \beta^s = s \cdot \log_\alpha \beta \bmod n$.

Problema logaritmului discret (DLP – *Discrete Logarithms Problem*) constă în următoarele:

fiind date un grup ciclic finit $G = \boldsymbol{Z}_p^*$, un generator α al lui G şi un element y din G, se cere să se găsească un număr întreg a, $0 \le a \le p - 2$, astfel încât $y = \alpha^a \pmod p$.

Algoritmul Diffie-Hellman pentru schimbul de chei constă în următoarele (Figura 10.2):

- *Alice* şi *Bob* convin asupra unui număr mare prim p şi a unui generator α.
- Alice alege (generează aleator) un număr secret a, şi îl trimite lui Bob numărul A:
$$A = \alpha^a \pmod p.$$
- Bob alege (generează aleator) un număr secret b, şi îl trimite lui Alice numărul B:
$$B = \alpha^b \pmod p.$$
- Alice calculează $B^a \bmod p = (\alpha^b \bmod p)^a \bmod p) = k$.
- Bob calculează $A^b \bmod p = (\alpha^a \bmod p)^b \bmod p) = k$.

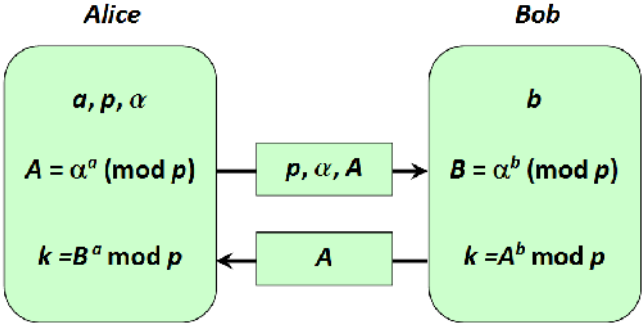


Figura 10.2. *Schema schimbului de chei Diffie-Hellman*

Cheia secretă stabilită de cei doi utilizatori Alice şi Bob este *k*. Menţionăm că *p* şi α nu este necesar de ţinut în secret. În realizările practice ale Algoritmului Diffie-Hellman pentru *a* şi *b* se utilizează chei de ordinul 10^{100} şi *p* de ordinul 10^{300}. Numărul α nu este neapărat mare şi de obicei are valori mai mici ca 10. Ţinând cont de aceste mărimi ale parametrilor *a*, *b* şi *p* putem afirma cu certitudine că securitatea algoritmului este una foarte mare, deoarece determinarea cheii secrete cunoscând numai α, *p*, α^b mod *p*, α^b mod *p* (fără a cunoaşte *a* şi *b* – ele fiind menţinute în secret) folosind cel mai performant este o problemă extrem de complicată care nu poate fi rezolvată într-un timp rezonabil.

Algoritmul Diffie-Hellman poate fi utilizat şi pentru criptarea cu cheie secretă. În acest caz schema este aceeaşi ca şi mai sus însă cu unele particularităţi. Alice nu transmite direct lui Bob valorile *p*, α şi *A* însă le publică din timp, ele având calitatea de cheie publică. Bob la rândul său efectuează calculele sale, după care cifrează mesajul cu un algoritm simetric, folosind *k* în calitate de cheie, apoi transmite spre Alice textul cifrat împreună cu valoarea lui *B*.

Exemplu de utilizare a schimbului de chei Diffie-Hellman:

- Alice şi Bob convin asupra lui $p = 23$ şi $\alpha = 5$.
- Alice alege aleatoriu $a = 6$ and şi trimite lui Bob $A = 5^6 \bmod 23 = 8$.
- Bob alege aleatoriu $b = 15$ şi trimite lui Alice $5^{15} \bmod 23 = 19$.
- Alice calculează $19^6 \bmod 23 = 2$.
- Bob calculează $8^{15} \bmod 23 = 2$.
- Alice şi Bob au abţinut aleşi rezultat, deci cheia secretă comună este $k = 2$.

În practică însă algoritmul Diffie-Hellman nu este utilizat astfel. Algoritmul cu chei publice care domină este algoritmul RSA, deoarece anume pentru RSA a fost creat un centru de cerificare. În plus algoritmul Diffie-Hellman nu poate fi utilizat la semnarea certificatelor.

Un alt sistem de criptare cu cheie publică bazat pe problema logaritmului discret este *sistemul ElGamal*, care conţine în sine şi algoritmul de semnătură digitală. *Schema ElGamal* se află la baza standardului de semnătură digitală în SUA (standardul DSA), precum şi în Rusia (ГОСТ Р 34.10-94).

Schema a fost propusă de către Taher Elgamal în anul 1984. Elgamal a elaborat una din versiunile algoritmului Diffie-Hellman. El a perfecţionat

algoritmul Diffie-Hellman şi a obţinut doi algoritmi care sunt utilizaţi pentru criptare şi pentru asigurarea autenticităţii. Spre deosebire de algoritmul RSA, algoritmul ElGamal nu a fost brevetat şi deci este o alternativă mai puţin costisitoare, deoarece nu este necesară achitarea unei taxă pentru licenţă. Se consideră că algoritmul ElGamal este acoperit de brevetul Diffie-Hellman.

Algoritmul ElGamal constă în următoarele:

1. *Generarea cheilor*:
   - Se generează aleatoriu un număr prim p de n biţi.
   - Se alege aleatoriu elementul primitiv α al câmpului $\boldsymbol{Z_p}$
   - Se alege aleatoriu un număr întreg a încât $1 < a \leq p - 2$.
   - Se calculează $y = \alpha^a \pmod p$.
   - Cheia publică este tripletul (p, α, y), cheia privată – numărul a.
2. *Cifrarea* mesajului m în sistemul ElGamal este de fapt o modalitate de generare a cheii publice Diffie-Hellman:
   - Se alege aleatoriu cheia de sesiune – un număr întreg k, $1 < k \leq p - 2$.
   - Se calculează $A = \alpha^k \pmod p$ şi $B = y^k \cdot m \pmod p$.
   - Textul cifrat este perechea de numere (A, B).

   Este evident că lungimea textului cifrat cu algoritmul ElGamal este de două ori mai mare decât lungimea textului clar m.

3. *Descifrarea*. Cunoscând cheia privată a, textul clar m poate fi calculat din textul cifrat (A, B) în felul următor:
   - $m = B \cdot (A^a)^{-1} \pmod p$.

   Aceasta este posibili deoarece
   - $(A^a)^{-1} = \alpha^{-ka} \bmod p$;
   - $B \cdot (A^a)^{-1} = (\alpha^{ka} \cdot m) \cdot \alpha^{-ka} = m \bmod p$.

   În scopuri practice pentru descifrare este mai convenabilă formula
   - $m = B \cdot (A^a)^{-1} = B \cdot A^{(p-1-a)} \pmod p$.

Deoarece în schema ElGamal se introduce o mărime aleatoare k acest cifru poate fi considerat cifru polialfabetic. Alegerea aleatorie a lui k a generat noţiunea de schemă probabilistică de cifrare. Caracterul probabilistic cal cifrării reprezintă o prioritate a cifrului ElGaml, deoarece schemele probabilistice au o rezistenţă mai mare decât alte scheme cu un

proces determinist de cifrare. Dezavantajul schemei ElGamal îl reprezintă dublarea lungimii textului cifrat în raport cu textul clar. Pentru schemele probabilistice de cifrare mesajul *m* şi cheia nu determină univoc textul cifrat. În schema ElGamal este necesar de a alege valori diferite ale mărimii aleatoare *k* pentru cifrarea a două mesaje diferite *m* şi m'. Dacă am folosi aceleaşi valori ale lui *k* pentru textele cifrate (A, B) şi (A', B') atunci din relaţia $B \cdot (B')^{-1} = m \cdot (m')^{-1}$, se poate uşor de calculat m' dacă cunoaştem *m*.

Exemplu de utilizare a algoritmului ElGamal, textul clar este $m = 65$, pe care Alice trebuie să-l trimită cifrat lui Bob.

Algoritmul ElGamal va consta în următoarele:

1. *Generarea cheilor*:
   - Se alege aleatoriu numărul prim $p = 281$.
   - Se alege aleatoriu elementul primitiv $\alpha = 3$ al câmpului $\mathbf{Z}_{281}$
   - Se alege aleatoriu un număr întreg $a = 57$ încât $1 < a \leq 279$.
   - Se calculează $y = \alpha^a \pmod p = 3^{57} mod\ 281 = 258$.
   - Cheia publică este tripletul $(p, \alpha, y) = (\mathbf{281}, \mathbf{3}, \mathbf{258})$.
   - cheia privată este numărul $a = \mathbf{57}$.

1. *Cifrarea*:
   - Alice alege aleatoriu cheia de sesiune $k = 49$ – un număr întreg, $1 < k \leq 279$.
   - Ea calculează
     $A = \alpha^k \pmod p = 3^{49}\ mod\ 281 = 146$ şi
     $B = y^k \cdot m \pmod p = (3^{49} \cdot 65)\ mod\ 281 = (152 \cdot 65)\ mod\ 281 = 45$.
   - Textul cifrat este perechea de numere $(A, B) = (146, 45)$.
   - Alice transmite lui Bob textul cifrat (146, 45).

2. *Descifrarea*. Bob, fiind titularul cheii secrete $a = 57$, primeşte textul cifrat (146, 45) de la Alice. Pentru descifrarea acestui mesaj el procedează în felul următor:
   - $m = B \cdot (A^a)^{-1} (mod\ p) = 45 \cdot (146^{57})^{-1} (mod\ 281)$.
   - Pentru aceasta Bob poate calcula mai întâi $146^{57} \bmod 281 = 152$.
   - Apoi calculează $152^{-1}\ mod\ 281 = 220$ (pentru aceasta poate aplica algoritmul Euclid Extins).

- Calculează textul clar
  $m = (45 \cdot 220) \ mod \ 281 = 9900 \ mod \ 281 = 65$.
- Textul clar putea fi calculat şi fără a aplica calculul inversului – aplicând formula $m = B \cdot A^{(p-1-a)} \ (mod \ p)$:
  $m = 45 \cdot 146^{(281-1-57)} \ (mod \ 281) = 45 \cdot 146^{223} \ (mod \ 281) =$
  $= 45 \cdot 220 \ (mod \ 281) = 9900 \ (mod \ 281) = 65$.

Un alt algoritm clasat printre primii algoritmi cu cheie publică este *Algoritmul Merkle-Hellman*. Ralph Merkle şi Martin Hellman au dezvoltat un algoritm de criptare bazat pe problema rucsacului, o problemă *NP*-completă, care a fost publicat în anul 1978. Problema rucsacului conţine o mulţime de întregi pozitivi şi o sumă ţintă, şi constă în găsirea unei submulţimi de întregi a căror sumă coincide cu suma ţintă.

Ideea pe care se bazează schema rucsacului Merkle-Hellman este codificarea unui mesaj binar ca o soluţie la o problemă a rucsacului, reducând mesajul în text cifrat la suma ţintă obţinută prin adunarea termenilor corespunzători valorilor de 1 din şirul binar.

Un rucsac este reprezentat ca un vector de numere întregi în care ordinea termenilor este foarte importantă. Există două tipuri de rucsacuri: unul simplu, pentru care există un algoritm rapid (în timp liniar) şi unul complicat, obţinut din cel simplu prin modificarea elementelor sale. Modificarea este astfel proiectată încât o soluţie cu elementele oricărui rucsac este de asemenea soluţie pentru celălalt. Această modificare se numeşte *trapă*, permiţând utilizatorilor legitimi să rezolve problema simplu. Deci, problema generală este ***NP***-completă, dar există o versiune restrânsă care are o soluţie foarte rapidă.

Algoritmul începe cu o mulţime de întregi în care fiecare element este mai mare decât suma predecesorilor săi. Să presupunem că avem un şir în care fiecare element a_k este mai mare decât $a_1+a_2+...+a_{k-1}$. Dacă o sumă este între a_k şi a_{k+1}, trebuie să-l conţină pe a_k, deoarece nici o combinaţie de termeni a_1, a_2, ..., a_{k-1} nu pot produce un total mai mare decât a_k. Analog, dacă o sumă este mai mică decât a_k, evident nu îl va conţine ca termen pe a_k.

Modificarea algoritmului schimbă elementele mulţimii din problema simplă a rucsacului, prin alterarea acestei proprietăţi de ordonare crescătoare într-un fel care păstrează soluţia. Modificarea se realizează prin înmulţire cu o constantă modulo *n*.

Problema rucsacului presupune un şir $a_1, a_2, ..., a_n$ de întregi şi o sumă ţintă *T*. Problema este de a găsi un vector de valori 0 şi 1 astfel încât suma întregilor asociaţi cu 1 să dea *T*. Deci, dându-se $S=[a_1, a_2, ..., a_n]$, şi *T*, să se găsească un vector *V* cu valori 0 şi 1 astfel încât:

$$\sum_{i=1}^{n} a_i \cdot v_i = T \ .$$

Rezolvarea se face considerând fiecare întreg din *S* ca participând la *T* şi reducând problema corespunzător. Când o soluţie nu produce suma ţintă, se elimină întregul ales iniţial şi se continuă cu următorul. Acest back-traking deteriorează viteza soluţiei.

Să presupunem problema rucsacului cu o restricţie suplimentară: întregii din *S* formează un *şir supercrescător*, adică unul în care fiecare întreg este strict mai mare decât suma predecesorilor săi. Atunci, orice întreg a_k satisface relaţia

$$a_k > \sum_{j=1}^{k-1} a_j \ .$$

Soluţia rucsacului supercrescător (numit şi rucsacul simplu) este uşor de găsit. Se începe cu *T*, care se compară cu cel mai mare întreg din *S*. Dacă acesta este mai mare decât *T*, nu este termen al sumei, deci valoarea corespunzătoare din *V* este 0. Dacă acest cel mai mare întreg din *S* este mai mic sau egal cu *T*, el este termen al sumei, deci valoarea corespunzătoare din *V* este 1. Reluăm algoritmul pentru *T* din care scădem sau nu termenul analizat (conform cu valoarea din *V*) şi pentru întregii rămaşi.

Tehnica de criptare Merkle-Hellman este un sistem de criptare cu cheie publică. Fiecare utilizator are o cheie publică, care poate fi distribuită oricui şi o cheie privată, care se păstrează secretă. Cheia publică este mulţimea întregilor din problema rucsacului (nu unul supercrescător); cheia privată este rucsacul supercrescător corespondent. Contribuţia lui Merkle şi Hellman a fost să proiecteze o tehnică de conversie a rucsacului supercrescător într-unul normal, prin schimbarea numerelor de o manieră reversibilă.

Tema 11. Sisteme asimetrice bazate pe curbe eliptice

În categoria sistemelor de criptare ale mileniului III pot fi cu certitudine introduse sistemele bazate pe curbe eliptice. Aceste sisteme de criptare ne permit să realizăm algoritmul de criptare asimetrică, protocolul de generare cheii secrete partajabile pentru criptarea simetrică, precum şi algoritmi de semnătură digitală. Aceste sisteme de criptare au o productivitate mai înaltă şi permit utilizarea cheilor substanţial mai mici ca mărime păstrând nivelul necesar de securitate.

Pentru diverse implementări se utilizează curbe eliptice de două tipuri:

- curbă eliptică peste un câmp finit F_p, unde p este un număr prim, $p > 3$;
- curbă eliptică peste un câmp finit F_2^m.

Curbele eliptice sunt un domeniu al matematicii cu o istorie ce se întinde pe parcursul a peste un secol. Utilizarea curbelor eliptice în criptografie a fost propusa pentru prima oară în 1985 de Victor Miller, cercetător de la IBM şi, independent, de Neal Kobitz, profesor la Universitatea din Washington. Ideea de baza era folosirea grupului punctelor de pe o curbă eliptică în locul grupului $\boldsymbol{Z}_p^*$ din sistemele criptografice existente. La momentul descoperirii lor, sistemele bazate pe curbe eliptice au fost considerate nepractice. De atunci însa s-a întreprins asupra lor o cercetare aprofundata şi intensă. Datorită perioadei de studiu de peste 20 de ani a proprietăţilor criptosistemelor bazate pe curbe eliptice, se poate considera ca acestea sunt suficient de bine cunoscute.

Încă de la acestor sisteme de criptare au existat discuţii extinse şi au fost publicate numeroase cărţi şi articole asupra securităţii şi eficientei lor. Anii ’90 au fost însa extrem de importanţi pentru acest tip de criptosisteme, deoarece acestea au început să cunoască acceptarea comercială odată cu standardizarea unor algoritmi şi protocoale bazate pe curbe eliptice. Eficienta curbelor eliptice ţine de un avantaj important pe care îl au aceste obiecte matematice în faţa altor criptosisteme. Avantajul constă în inexistenţa sau mai bine spus, nu se cunoaşte un algoritm cu timp subexponenţial care să găsească logaritmi discreţi pentru grupurile generate de o curbă eliptică. În plus, un alt avantaj ar fi acela că utilizarea unei astfel de structuri care este mai mica decât altele, în ceea ce priveşte numărul de elemente, conduce la păstrarea aceluiaşi nivel de securitate cu

dimensiuni ale cheilor mai mici decât în cazul altor criptosisteme. Dimensiunile reduse ale cheilor şi ale reprezentărilor elementelor conduc implicit la utilizarea a mai puţine resurse pentru criptarea datelor şi reducerea lăţimii de bandă necesare transmiterii textelor criptate. Este evident ca astfel de proprietăţi fac din curbele eliptice şi criptosistemele bazate pe aceste obiecte matematice soluţii ideale de securitate a datelor pentru mediile în care puterea de procesare şi conexiunea la reţea sunt limitate. Printre aceste medii se pot enumera: telefoanele mobile, PDA-uri, smart-carduri, carduri pc, etc.

În ultimii ani, curbele eliptice au fost folosite pentru conceperea unor algoritmi eficienţi de factorizare a întregilor şi pentru demonstrarea primalităţii. Ele au fost utilizate în construirea criptosistemelor cu chei publice, în construirea generatoarelor de biţi pseudoaleatoare şi a permutărilor neinversabile. Curbele eliptice şi-au găsit aplicabilitate şi în teoria codurilor, unde au fost întrebuinţate pentru obţinerea unor coduri corectoare de erori foarte bune. Curbele eliptice au jucat un rol important şi în recenta demonstraţie a *Ultimei Teoreme* a lui Fermat. Folosirea sistemelor de criptare bazate pe curbe eliptice permite creşterea securităţii, scăzând în acelaşi timp overhead-ul (supraîncărcarea) şi timpul de latenţă.

Securitatea criptosistemelor bazate pe curbe eliptice constă în dificultatea calculului logaritmilor în câmpuri discrete (problema logaritmilor discreţi): date fiind A (un element dintr-un câmp finit) şi A^x, este practic imposibil să se calculeze x, atunci când elementele sunt suficient de mari. Şi alte sisteme criptografice se bazează pe problema logaritmilor discreţi in $\boldsymbol{Z}^*_p$: *ElGamal*, algoritmul de semnătură *Schnorr*, algoritmul de semnătura *Nyberg-Rueppel*, *DSA*. În mod clasic, aceste sisteme au fost definite în grupul multiplicativ $\boldsymbol{Z}^*_p$. Ele pot fi însa definite la fel de bine în orice alt grup finit, cum ar fi grupul punctelor de pe o curbă eliptică.

Serviciile de securitate oferite de criptosistemele bazate pe curbe eliptice sunt:

- autentificarea entităţilor
- confidenţialitatea
- integritatea datelor
- ne-repudierea
- schimbul de chei autentificat.

Curbele eliptice sunt benefice în aplicaţii în care :

- puterea de calcul este limitată (cartele inteligente, dispozitive fără fir, placi PC);
- spaţiul pe circuit integrat este limitat (cartele inteligente, dispozitive fără fir, placi PC);
- este necesară viteza mare de calcul;
- se foloseşte intens semnarea şi verificarea semnăturii;
- mesajele semnate trebuie memorate sau transmise;
- lăţimea de bandă este limitată (comunicaţii mobile, anumite reţele de calculatoare).

Aplicaţii de genul transferurilor bancare sau transmisiile de date prin reţele radio (fără fir), care necesită folosirea intensivă a semnării digitale, autentificării, viteză ridicată şi lărgime de bandă limitată, vor beneficia din plin de avantajele oferite de implementările bazate pe conceptul curbelor eliptice. Sistemele bazate pe curbe eliptice se pot implementa mult mai uşor şi eficient atât în hardware cât şi în software. Implementările existente la momentul actual indică faptul că aceste sisteme sunt pe departe mai eficiente decât orice alt sistem cu chei publice. Un cip construit de Certicom Corporation pentru realizarea operaţiilor pe o curbă eliptică peste câmpul $F_{2^{155}}$ are o frecvenţa de ceas de 40MHz şi poate efectua aproximativ 40000 de operaţii pe secundă. Cipul are doar 12000 de porţi şi este de 10 ori mai rapid decât DSA sau RSA pe 1024 de biţi.

Definiţie*:* Fie p (p >3) un număr prim. Curba eliptica $y^2 = x^3 + ax + b$ peste $\boldsymbol{Z}_p$ constă din mulţimea soluţiilor $(x, y) \in \boldsymbol{Z}_p \times \boldsymbol{Z}_p$ ecuaţiei

$$y^2 = x^3 + ax + b \pmod{p},$$

unde $a, b \in \boldsymbol{Z}_p$ sunt constante astfel încât $4a^3 + 27b^2 \neq 0 \pmod{p}$ şi dintr-un punct O numit „punct la infinit”.

O curbă eliptică E se poate structura ca un grup abelian finit. Legea de compoziţie (notată aditiv) este definită astfel:

Fie $P, Q \in E$, $P = (x_1, y_1)$, $Q = (x_2 , y_2)$.

Dacă $x_2 = x_1$, $y_2 = -y_1$, atunci $P + Q = O$; altfel, $P + Q = (x_3, y_3)$, unde $x_3 = \lambda^2 - x_1 - x_2$, $y_3 = \lambda(x_1 - x_2) - y_1$, iar

$$\lambda = \begin{cases} \dfrac{y_2 - y_1}{x_2 - x_1} \\ \dfrac{3x_1^2 + a}{2y_1} \end{cases}$$ dacă $P \neq Q$ sau $P = Q$.

Se mai defineşte $P+O = O+P = P$, $\forall P \in E$. Elementul neutru este O.

Operaţii cu punctele de pe curbele eliptice

Adunarea punctelor de pe curbele eliptice poate fi explicată cel mai simplu prin prisma reprezentărilor geometrice. Operaţia de adunare pe o curbă eliptică este corespondentă operaţiei de înmulţire în sistemele cu chei publice obişnuite iar adunarea multiplă este corespondentă exponenţierii modulare din acestea.

Să examinăm reprezentarea curbei din Figura 11.1. Adunarea punctelor $P(x_1, y_1)$ şi $Q(x_2, y_2)$ este echivalentă cu găsirea punctului $R(x_3, y_3)$ prin trasarea unei drepte între punctele P şi Q care intersectează curba într-un al treilea punct. Acest punct este inversul punctului R, iar R va fi punctul de reflexie al acestui punct. Mai exact vom trasa o perpendiculară din punctul „$-R$” pe axa OX care va intersecta curba în punctul R.

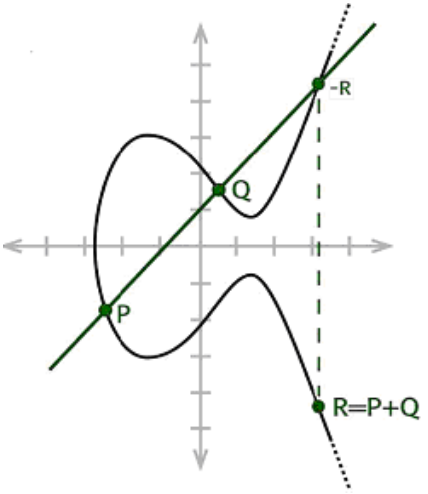


Figura 11.1. *Adunarea punctelor de pe curba eliptica*

O altă operaţie posibilă cu punctele unei curbe este dublarea unui punct. Dublul unui punct se obţine asemănător, cu diferenţa că de aceasta dată trasăm o dreaptă tangentă la punctul $P(x_1, y_1)$. Conform Figurii 11.2 se obţine la fel ca la adunare inversul punctului R, din care vom obţine R. Din punct de vedere algebric pentru a putea folosi aceste elemente pentru a defini un criptosistem bazat pe operaţii cu puncte de pe curbe eliptice, avem nevoie sa definim o structură algebrică. Cea mai simpla structură care permite acest lucru este grupul.

Fie E o curbă eliptică peste F_p sau peste F_{p^m} şi două puncte P şi Q de pe curba eliptica E. Mulţimea punctelor de pe curba E, notată cu $E(F_p)$ are structura de grup împreuna cu adunarea punctelor.

1. *Exista element neutru*: Dacă P este punctul la infinit notat cu ∞, atunci definim punctul $-P$ ca fiind ∞. Pentru orice alt punct Q definim $Q + \infty = Q$.
2. *Există element invers*: În F_p definim punctul invers al lui $P(x, y)$ ca fiind $-P(x,-y)$. Dacă există un punct $Q = -P$, atunci $Q + P = \infty$.
3. *Operaţia de adunare*: O dreapta care intersectează curba eliptică în două puncte P şi Q va intersecta curba şi într-un al treilea punct. Definim $P + Q = -R$, $-R(x_3, y_3)$ vor fi coordonatele celui de-al treilea punct de intersecţie al dreptei cu curba eliptică E.
4. *Dublarea punctelor*: O dreaptă d tangentă la curba eliptica E va intersecta curba eliptică în punctul $-R(x_3, y_3)$. Definim dublarea punctului P prin $2P = -R$

Din descrierea de mai sus pot fi calculate formule exacte pentru adunarea şi dublarea punctelor în funcţie de coordonatele punctelor şi de tipurile de curbe prezentate mai sus, rezultate prin schimbările acceptabile de variabile. Deşi regulile de calcul în grupul punctelor unei curbe eliptice par destul de complicate, aritmetica acestora poate fi implementată extrem de eficient, calculele în acest grup fiind realizate mult mai rapid decât cele din grupul $\boldsymbol{Z_p}$.

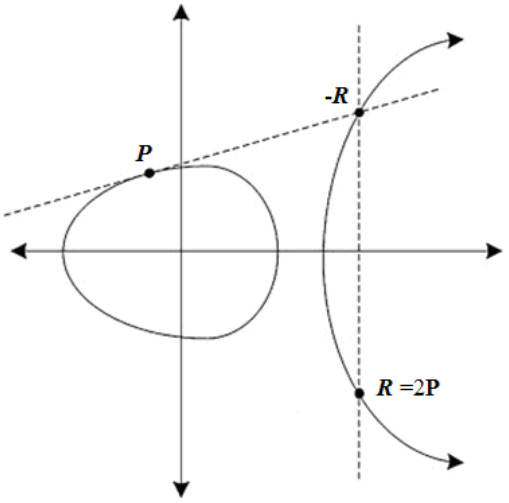


Figura 11.2. *Adunarea punctelor de pe curba eliptica*

Exemplu:

Fie curba eliptica E: $y^2 = x^3 + x + 6$ pe $\boldsymbol{Z}_{11}$. Calculăm mai întâi punctele lui E: pentru orice $x \in \boldsymbol{Z}_{11}$, se calculează $z = x^3 + x + 6 \bmod 11$. Se testează dacă z este un rest pătratic (pentru un x dat) folosind criteriul lui Euler. Aplicând formula de calcul a rădăcinilor pătrate a unui rest pătratic modulo p se obţine: $\pm z^{\frac{11+1}{4}} \bmod 11 = \pm z^3 \bmod 11$. Calculele sunt reprezentate în Tabelul 11.1.

Deci curba eliptica E admite 13 puncte. Ordinul grupului este prim, deci grupul este ciclic. Presupunem că se ia α =(2, 7) generator al grupului. Se pot atunci calcula multiplii lui α (care sunt puteri ale lui α deoarece grupul este aditiv). Pentru a calcula 2α =(2, 7) + (2, 7) se calculează mai întâi $\lambda = (3 \times 2^2 + 1)(2 \times 7)^{-1} \bmod 11 = 2 \times 3^{-1} \bmod 11 = 2 \times 4 \bmod 11 = 8$. Atunci avem $x^3 = 8^2 - 2 - 2 \bmod 11 = 5$ şi $y^3 = 8(2-5) - 7 \bmod 11 = 2$.

Observam că α ales mai sus este cu adevărat un generator al grupului.

Să analizăm în continuare un *exemplu de criptare* El Gamal pe curba eliptica din exemplul anterior:

Fie α = (2, 7) şi exponentul secret da = 7. Avem $\beta = 7\alpha$ = (7, 2). Criptarea textului clar x cu cheia k se face in modul următor:

$e_K(x, k) = (k\alpha, x+\beta k) = (k\,(2, 7), x + k\,(7, 2))$, unde $0 \le k \le 12$ şi $x \in E$

Operaţia de decriptare se desfăşoară astfel:

$$d_K(y_1, y_2) = y_2 - 7y_1.$$

Să presupunem că utilizatorul *A* vrea să cifreze mesajul $x = (10, 9)$ (care este un punct de pe *E*) spre a-l trimite utilizatorului *B*. Pentru aceasta el alege valoarea aleatoare $k = 3$ şi calculează:

$$y_1 = 3(2,7) = (8,3)$$

şi

$$y_2 = (10, 9) + 3(7, 2) = (10, 9)+(3, 5) = (10, 2)$$

Deci textul cifrat este $y = ((8, 3), (10, 2))$.

La recepţie, utilizatorul *B* descifrează mesajul în următorul mod:

$x = (10, 2) – 7(8, 3) = (10, 2) – (3, 5) = (10, 2) + (3, 6) = (10, 9)$

rezultând sensul descifrat al mesajului.

x	$x^3+x+6 \bmod 11$	este rest pătratic modulo 11 ?	y
0	6	Nu	
1	8	Nu	
2	5	**Da**	4, 7
3	3	**Da**	5, 6
4	8	Nu	
5	4	**Da**	2, 9
6	8	Nu	
7	4	**Da**	2, 9
8	9	**Da**	3, 8
9	7	Nu	
10	4	**Da**	2, 9

Tabelul 11.1. *Calculul punctelor curbei eliptice E*

Tema 12. Sistemul de criptare RSA

Un alt criptosistem bazat pe o problemă dificilă este **Algoritmul RSA**, numit astfel după inventatorii săi, Rivest, Shamir şi Adelman. A fost publicat în 1978 şi rămâne un algoritm foarte folosit şi astăzi, în ciuda eforturilor criptanaliştilor de a-l sparge.

Algoritmul de criptare RSA incorporează rezultate din teoria numerelor, combinate cu dificultatea determinării factorilor primi pentru un număr ţintă. Ca în cazul algoritmului Merkle-Hellman şi algoritmul RSA operează cu aritmetica modulo *n*. Un bloc în text clar este tratat ca un întreg, iar pentru criptare şi decriptare se folosesc două chei, *e* şi *d*, care sunt interschimbabile. Blocul de text clar *P* este criptat ca $P^e \bmod n$. Deoarece exponenţierea este modulo *n*, este foarte dificil să se factorizeze P^e pentru a descoperi textul original. Pentru aceasta, cheia de decriptare *d* este astfel aleasă încât

$$(P^e)^d = P \bmod n.$$

Astfel *P* este regăsit fără a fi necesară descompunerea în factori primi a lui P^e.

Problema pe care se bazează algoritmul de criptare este cea a factorizării numerelor mari. Problema factorizării nu se cunoaşte a fi NP-completă; cel mai rapid algoritm cunoscut este exponenţial în timp.

Cu algoritmul RSA mesajul în text clar *p* este criptat prin intermediul cheii de criptare *e* obţinându-se mesajul în text cifrat *c*:

$$c = p^e \bmod n.$$

Mesajul în text clar este regăsit cu ajutorul cheii de decriptare *d*:

$$p = c^d \bmod n.$$

Din cauza simetriei din aritmetica modulară, criptarea şi decriptarea sunt mutual inverse şi comutative:

$$p = c^d \bmod n = (p^e)^d \bmod n = (p^d)^e \bmod n.$$

Cheia de criptare constă în perechea de întregi (*e, n*), iar cheia de decriptare este (*d, n*). Punctul de plecare în găsirea cheilor pentru acest algoritm este selectarea unei valori pentru *n*. Valoarea lui *n* trebuie să fie suficient de mare, dată de un produs a două numere prime *p* şi *q*. Atât *p* cât şi *q* trebuie să fie ele însele suficient de mari. În mod obişnuit, *p* şi *q* au aproximativ 100 de cifre fiecare, astfel încât *n* are aproximativ 200 de cifre. Această lungime inhibă încercarea de a factoriza pe *n*, pentru a afla pe *p* şi pe *q*.

În continuare, se alege un întreg *e* relativ mare, astfel încât *e* este relativ prim cu (*p*–1)·(*q*–1). Satisfacerea acestei condiţii se face alegându-l pe *e* ca un număr prim mai mare decât *p*–1 şi *q*–1. În final, se alege *d* astfel încât:

$$e \cdot d \equiv 1 \bmod (p-1)\cdot(q-1).$$

Funcţia lui Euler $\varphi(n)$ este numărul întregilor pozitivi mai mici decât *n* care sunt relativ primi cu *n*. Dacă *p* este prim, atunci:

$$\varphi(p) = p-1.$$

Dacă $n = p \cdot q$, unde *p* şi *q* sunt ambele prime, atunci

$$\varphi(n) = \varphi(p) \cdot \varphi(q) = (p-1) \cdot (q-1)$$

Identitatea Euler-Fermat afirmă că

$$x^{\varphi(n)} \equiv 1 \bmod n$$

pentru orice întreg *x*, dacă *n* şi *x* sunt reciproc prime.

Să presupunem că mesajul în text clar *p* este criptat cu algoritmul RSA, astfel încât $e(p)=p^e$. Trebuie să fim siguri că putem decripta mesajul. Valoarea *e* este astfel aleasă încât inversa sa *d* să poată fi găsită uşor. Deoarece *e* şi *d* sunt inverse modulo $\varphi(n)$,

$$e \cdot d \equiv 1 \bmod \varphi(n) \text{ sau } e \cdot d = k \cdot \varphi(n) + 1$$

pentru anumiţi întregi *k*.

La implementarea practică a algoritmului, utilizatorul algoritmului RSA alege numerele prime *p* şi *q*, din care se obţine $n = p \cdot q$. Apoi alege *e*, relativ prim la (*p*–1)·(*q*–1), de obicei un număr prim mai mare decât *p*–1 şi decât *q*–1. În final, *d* se calculează ca inversul lui *e* mod φ(n).

Utilizatorul distribuie *e* şi *n*, şi păstrează cheia *d* secretă; *p*, *q* şi $\varphi(n)$ pot fi ignorate, dar nu făcute publice. Chiar dacă se ştie că *n* este produsul a două numere prime, datorită mărimii sale – peste 200 de cifre, nu va fi posibil să se determine factorii *p* şi *q*, şi nici cheia privată, *d* din *e*. De asemenea, verificarea că *p* şi *q* sunt prime, presupune luarea în considerare a 10^{50} factori.

La momentul actual în RSA se utilizează numere prime găsite prin intermediul algoritmilor probabilistici, cel mai performant fiind testul Miller-Rabin. El este considerat suficient de bun pentru generarea numerelor prime aplicate în criptografie. Dacă însă acest test ar genera la un moment dat un număr compus, urmările vor fi imprevizibile pentru utilizatorii algoritmului RSA cu cheile respective.

Teoretic sunt trei posibilităţi de abordare a unui atac în cazul algoritmului RSA: atacul în forţă, atacul bazat pe metode matematice (încercarea factorizării produsului a două numere prime mari) şi atacul temporal. Analiza acestor atacuri duce la concluzia că nici unul nu are sorţi de izbândă. În pofida unor intense cercetări, au fost identificate doar probleme minore în comparaţie cu cele din cazul algoritmului rucsacului a lui Merkle şi Hellman.

Aşadar algoritmul RSA constă din următorii 3 algoritmi: generatorul de chei, algoritmul de criptare şi algoritmul de decriptare.

Generatorul de chei.

1. Generează două numere prime mari p şi q;
2. Calculează $n = pq$ şi indicatorul Euler $\varphi(n) = (p - 1)(q - 1)$;
3. Alege aleator un număr e ($1<e<\varphi(n)$) astfel ca cmmdc(e, $\varphi(n)$) = 1;
4. Calculează $d = e^{-1} \ mod \ \varphi(n)$ (adică $d \cdot e = 1 \ mod \ \varphi(n)$) folosind algoritmul extins al lui Euclid;
5. Face public n şi e, adică cheia publică este (e, n) iar cea privată este (d, n).

Algoritmul de criptare.

Fie m – mesajul în clar dat sub forma unui număr natural mai mic decât n (m poate fi reprezentarea zecimală conform tabelului *ASCII* a caracterului ce trebuie criptat). Atunci textul cifrat, notat cu c, este

$$c = m^e \bmod n.$$

Algoritmul de decriptare.

Fie c – textul cifrat primit. Atunci textul clar m, este

$$m = c^d \bmod n.$$

Pentru calcularea inversului d al uni număr întreg e în clasele de resturi *modulo n* putem aplica algoritmul Euclid extins, care poate fi implementat cu ajutorul Tabelului 12.1. În acest tabel avem:

$$x_1 = 1,\ y_1 = 0,\ r_1 = n,\ x_2 = 0,\ y_2 = 1,\ r_2 = e.$$

$$q_i = \left[\frac{r_{i-1}}{r_i} \right],\ i = 2, 3, \ldots,$$

unde $[x]$ reprezintă partea întreagă a numărului x. Pentru $i = 3, 4, 5 \ldots$ avem:

$$r_i = r_{i-2} \bmod r_{i-1}, ,$$
$$x_i = x_{i-2} - q_{i-1} \cdot x_{i-1},$$
$$y_i = y_{i-2} - q_{i-1} \cdot y_{i-1}.$$

Procesul continuă până când obţinem $r = 1$. În acest caz (la iteraţia *k*) $e^{-1} \bmod n = y_k$. În cazul în care $y_k < 0$ se adună *n*.

Pentru calcularea rapidă a lui $a^b \bmod n$ se poate aplica următoarea metodă:

1. se determină reprezentarea binară a lui *a*;
2. fiecărei unităţi din această reprezentare binară îi corespund operaţiile de ridicare la pătrat şi înmulţirea cu baza $((x^2 \bmod n) \cdot a) \bmod n$, iar pentru fiecare 0 – numai înmulţirea cu baza $(x \cdot a) \bmod n$; prima unitate din această reprezentare nu o luăm în consideraţie (pe primul loc numaidecât se află 1).

i	*x*	*y*	*r*	*q*
1	1	0	*n*	
2	0	1	*e*	$q_2 = \left[\frac{r_1}{r_2}\right]$
3	$x_i = x_{i-2} - q_{i-1} \cdot x_{i-1}$	$y_i = y_{i-2} - q_{i-1} \cdot y_{i-1}$	$r_i = r_{i-2} \bmod r_{i-1}$	$q_3 = \left[\frac{r_2}{r_3}\right]$
…	…	…	…	…
k	$x_k = x_{k-2} - q_{k-1} \cdot x_{k-1}$	$y_k = y_{k-2} - q_{k-1} \cdot y_{k-1}$	1	$q_k = \left[\frac{r_{k-1}}{r_k}\right]$

Tabelul 12.1. *Schema algoritmului Euclid extins*

Să analizăm un exemplu al algoritmului RSA. Din start trebuie de menţionat că acest exemplu este unul pur instructiv, el neavând nici un grad de securitate.

Exemplu. *Bob* trebuie să genereze o pereche de chei pentru algoritmul *RSA* şi să-i transmită lui *Alice* cheia sa publică. *Alice* va cripta cu această cheie mesajul „***A***” şi va trimite lui *Bob* textul cifrat, care, folosind cheia sa privată, trebuie să-l decripteze.

Pentru generarea cheilor *Bob* alege 2 numere prime:

$$p = 31 \text{ şi } q = 23.$$

Calculează produsul

$$n = 31 \cdot 23 = 713.$$

Apoi calculează indicatorul Euler

$$\varphi(n) = (p - 1)\cdot(q - 1) = 30 \cdot 22 = 660.$$

În continuare *Bob* alege aleator un număr e ($1 < e < \varphi(n)$) astfel ca cmmdc(e, $\varphi(n)$) = 1, adică $1 < e < 660$ şi cmmdc(e, 660) = 1. Fie că alege

$$e = 223$$

Calculează $d = 223^{-1} \bmod 660$ aplicând algoritmul Euclid extins (Tabelul 12.2).

i	x	y	r	q
1	1	0	660	
2	0	1	223	2
3	1	-2	214	1
…	–1	3	9	23
k	24	–71	7	1
	–25	74	2	3
	99	**–293**	**1**	2
			Stop!	

Tabelul 12.2. *Schema algoritmului Euclid extins pentru* $d = 223^{-1}$ *mod* 660

Aşadar,

$$d = 223^{-1} \textit{ mod } 660 = -293 \textit{ mod } 660 = -293 + 660 = 367$$

Deci cheia privată a lui *Bob* este (367, 713).

Ceia sa publică (223, 713) – Bob o transmite lui *Alice*.

Alice trebuie să trimită mesajul „***A***” lui *Bob* cifrând-ul cu algoritmul *RSA*. Conform tabelului *ASCII* valoarea zecimală a lui „***A***” este 65. Deci $m = 65$. Pentru aceasta *Alice* calculează

$$c = 65^{223} \bmod 713.$$

Reprezentarea binară al exponentului 223 este:

$$223_{10} = 11011111_2.$$

Aşadar,

$$65^{223} \bmod 713 = ((((((((((((65^2 \bmod 713) \cdot 65 \bmod n)^2 \bmod n)^2 \bmod n) \cdot$$
$$\cdot 65 \bmod n)^2 \bmod n) \cdot 65 \bmod n)^2 \bmod n) \cdot$$
$$\cdot 65 \bmod n)^2 \bmod n) \cdot 65 \bmod n)^2 \bmod n) \cdot 65 \bmod n).$$

Efectuând consecutive calculele se obţine

$$65^{223} \bmod 713 = 396.$$

Alice trimite lui *Bob* textul cifrat

$$c = 396.$$

Bob aplică aceeaşi metodă şi calculează

$$m = c^d \bmod n = 396^{367} = 65,$$

adică conform tabelului *ASCII* – litera „***A***”.

Dacă numărul *n* este insuficient de mare, atunci algoritmul poate fi spat. În cazul nostru $n = 713$. Acest număr poate fi foarte simplu factorizat: $n = 713 = 31 \cdot 23$.

Răuvoitorul cunoaşte cheia publică, deci ştie că $e = 223$ şi la fel ca *Bob* calculează *d* şi obţine cheia lui privată. Algoritmul este spart.

În conformitate cu recomandările unor specialişti din domeniu lungimea cheii RSA trebuie să fie în concordanţă cu Tabelul 12.3.

Durata de viaţă a datelor	Lungimea cheii RSA
până în 2010	1024 biţi
până în 2030	2048 biţi
începând cu 2031	3072 biţi

Tabelul 12.3. *Recomandări referitoare la lungimea cheii RSA*

Tema 13. Funcţiile HASH criptografice

Unul dintre instrumentele importante ale criptografiei moderne sunt *funcţiile HASH*. Aceste funcţii efectuează transformări asupra unei intrări de o lungime arbitrară generând o ieşire de lungime fixă care este o imagine a intrării. Valoarea „imaginii" returnate se numeşte valoare HASH. Aceste funcţii se mai numesc şi funcţii „*message digest*", deoarece din valoarea HASH este imposibil de reconstruit intrarea din care s-a făcut imaginea. Există multe funcţii care pot transforma intrarea de lungime arbitrară în o ieşire de lungime fixă, însă funcţiile care prezintă interes din punct de vedere criptografic sunt funcţiile hash de sens unic (*one–way hash functions*). Ecuaţia generală care descrie funcţiile hash este

$$h = H(m),$$

unde m intrarea, H este funcţia şi h – valoarea hash.

Caracteristicile speciale ale funcţiilor de sens unic sunt următoarele:

- Pentru orice intrare M este uşor de calculat h.
- Având h este dificil de calculat M astfel încât se fie satisfăcută relaţia $h = H(m)$.
- Pentru un M dat este dificil de găsit M_1 astfel încât $H(M) = H(M_1)$.

Aceste proprietăţi ale funcţiilor hash de sens unic folosite în criptografie sunt extrem de importante deoarece dacă un atacator ar putea găsi cu uşurinţă o valoare de intrare m_1 care produce acelaşi rezultat cu m atunci dacă cineva semnează m, atacatorul ar putea afirma că a fost semnat m_1. O cerinţă adiţională pentru aceste funcţii este ca pentru ele să fie dificil de găsit două mesaje aleatoare care să aibă acelaşi imagine, altfel spus:

- Este dificil de găsit mesajele aleatoare m şi m_1 astfel încât $H(m) = H(m_1)$.

Un posibil atac, dacă ultima cerinţă nu este îndeplinită, ar fi atacul numit „atacul zilei de naştere" (birthday atack). Acest tip de atac şi-a primit numele de la paradoxul zilelor de naştere. În mod surprinzător, se poate dovedi matematic că probabilitatea ca în orice grup de 23 de persoane doi sau mai mulţi indivizi să aibă acelaşi zi de naştere este mai mare decât $\frac{1}{2}$.

Atacul descris de către criptologul *Bruce Schneier* presupune folosirea unei funcţii hash care produce un rezultat de 64 biţi. Paşii care trebuie parcurşi pentru ca Bob să obţină o semnătură de la Alice pe un document care n-ar fi semnat, în condiţii normale, niciodată de Alice:

1. Bob pregăteşte două versiuni ale documentului, una care este în ordine şi va fi semnat de Alice, şi una care conţine informaţii false care nu ar fi semnate de Alice.
2. Bob face schimbări subtile pe fiecare document (de ex. inserare de spaţii etc.), obţinând 2^{23} variante ale documentelor.
3. Bob calculează valorile Hash a fiecărei variante ale celor două documente, şi găseşte câte una din fiecare grup care produc aceleaşi valori HASH.
4. Bob trimite varianta modificată a documentului pe care Alice este dispusă să semneze, folosind un protocol unde se semnează numai valoarea HASH.
5. Acuma Bob poate susţine că documentul pereche cu informaţiile false a fost semnată de Alice.

Nu este uşor de găsit o funcţie care satisface toate cerinţele de mai sus, totuşi există destul de multe, mai mult sau mai puţin sigure, funcţii HASH folosite în criptografie. Vom oferi o descriere a celor mai cunoscute dintre aceste funcţii.

MD5 (MD provine din *Message Digest*) este o funcţie hash de sens unic, proiectat de Ron Rivest (unul dintre autorii algoritmului RSA). Algoritmul produce un hash, sau altfel zis imagine de 128 biţi a mesajului de intrare.

Principiile după care s-a realizat algoritmul sunt următoarele:

- *Securitate*. Este imposibil de găsit două mesaje care au aceiaşi imagine, presupunând că nu există altă metodă de criptanaliză decât cea a forţei brute.
- *Securitate directă*. Securitatea nu se bazează pe presupuneri, cum ar fi de exemplu dificultatea de a factoriza numere mari în cazul RSA.
- *Viteză*. Algoritmul trebuie să fie potrivit pentru implementări rapide de software, bazându-se pe manipulaţii de bit cu operanzi de 32 biţi.

- *Simplicitate şi compactitate*. Algoritmul trebuie să fie cât se poate de simplu, fără structuri mari de date sau un program complicat.
- *Favorizare de arhitecturi Little-Endian.* Algoritmul este optimizat pentru arhitecturi de microprocesoare (mai ales Intel). Calculatoarele mai performante fac transformările necesare.

Prima variantă a algoritmului a fost MD4, dar acesta după ce a fost introdus a fost criptanalizat cu succes în părţi, ce l-a îndemnat pe autor să-si îmbunătăţească codul. Astfel a fost conceput MD5, ca variantă MD4 îmbunătăţită.

*Descrierea algoritmului MD*5. După nişte procesări iniţiale MD5 procesează textul de intrare în blocuri de 512 de biţi, care sunt mai departe separaţi în 16 sub-blocuri de 32 biţi fiecare. Algoritmul produce un set de 4 blocuri de 32 biţi, care concatenate dau ieşirea de 128 bit.

Mesajul este mărit pentru a fi multiplul lui 512. Acest procedeu se realizează prin adăugarea unui bit de 1 la sfârşitul mesajului şi atâtea zerouri câţi sunt necesari ca mesajul original să aibă o lungime cu 64 de biţi mai scurtă decât un multiplu al lui 512. O reprezentare pe 64 de biţi este apoi adăugată la sfârşitul mesajului. Acest procedeu asigură ca complementarea să arate diferit pentru mesajele diferite.

La început sunt iniţializate patru variabile, numite variabile de legătură:

$$A = 0x01234567$$
$$B = 0x89abcdef$$
$$C = 0xfedcba98$$
$$D = 0x76543210$$

Apoi începe ciclul principal al algoritmului, care este repetat în funcţie de numărul de blocuri de 512 din mesajul de intrare. Cele patru variabile sunt copiate în alte variabile, anume: *A* în *a*, *B* în *b*, *C* în *c* iar *D* în *d.*

Ciclul principal are patru runde asemănătoare. Fiecare rundă foloseşte o operaţie diferită de 16 ori. Fiecare operaţie aplică o funcţie ne-liniară pe trei din variabilele *a,b,c,d*, adăugând rezultatul la al patrulea variabilă, la un sub-bloc a textului de intrare şi o constantă. Apoi roteşte rezultatul obţinut la dreapta cu un număr variabil de biţi şi adaugă rezultatul la unul dintre variabilele *a,b,c*, sau *d*. În final rezultatul este copiat într-unul dintre

variabilele de dinainte. Funcţionarea buclei principale a algoritmului este prezentată în Figura 13.1.

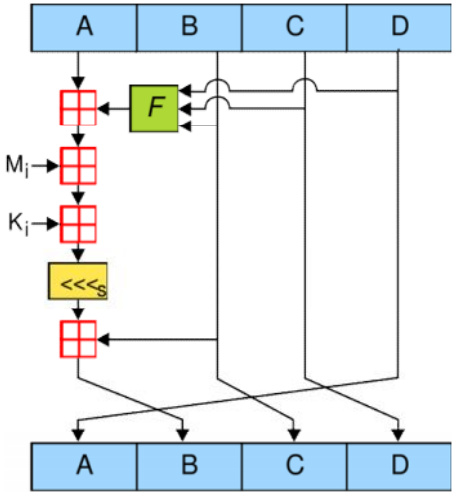


Figura 13.1. *Bucla principală MD5*

Sunt patru funcţii neliniare folosite în fiecare operaţie. Se foloseşte o funcţie diferită în fiecare rundă. Funcţiile neliniare sunt:

$$F(X,Y,Z) = (X \,\&\, Y) \mid ((!X) \,\&\, Z)$$
$$G(X,Y,Z) = (X \,\&\, Z) \mid (Y \,\&\, (!Z))$$
$$H(X,Y,Z) = X \text{ xor } Y \text{ xor } Z$$
$$I(X,Y,Z) = Y \text{ xor } (X \mid (!Z))$$

Considerând M_j fiind blocul j din sub-blocul de mesaje şi „<<< s” fiind o rotaţie circulară de s biţi, cele patru operaţii sunt:

FF $(a, b, c, d, M_j, s, t_i)$ înseamnă $a = b+((a + F(b, c, d)+M_j + t_i) <<< s)$
GG $(a, b, c, d, M_j, s, t_i)$ înseamnă $a = b+((a + G(b, c, d)+M_j + t_i) <<< s)$
HH $(a, b, c, d, M_j, s, t_i)$ înseamnă $a = b+((a + H(b, c, d)+M_j + t_i) <<< s)$
II $(a, b, c, d, M_j, s, t_i)$ înseamnă $a = b + ((a + I(b, c, d) + M_j + t_i) <<< s)$

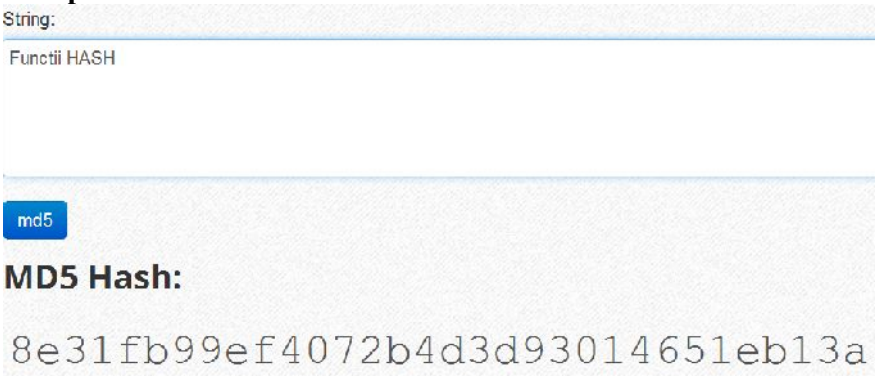
Cele patru runde (64 de paşi) sunt prezentate în Tabelul 13.1. Constantele t_i sunt alese astfel încât t_i în pasul i este partea întreagă a valorii $232 \cdot abs(sin(i))$, unde i este în radiani. După parcurgerea celor 64 de paşi valorile a, b, c, d sunt adăugate la A, B, C, D şi ciclul se repetă cu următorul bloc de date. Ieşirea finală va fi concatenarea valorilor A, B, C şi D.

Runda 1	Runda 2
FF (a, b, c, d, M0, 7, 0xd76aa478)	GG (a, b, c, d, M1, 5, 0xf61e2562)
FF (d, a, b, c, M1, 12, 0xe8c7b756)	GG (d, a, b, c, M6, 9, 0xc040b340)
FF (c, d, a, b, M2, 17, 0x242070db)	GG (c, d, a, b, M11, 14, 0x265e5a51)
FF (b, c, d, a, M3, 22, 0xc1bdceee)	GG (b, c, d, a, M0, 20, 0xe9b6c7aa)
FF (a, b, c, d, M4, 7, 0xf57c0faf)	GG (a, b, c, d, M5, 5, 0xd62f105d)
FF (d, a, b, c, M5, 12, 0x4787c62a)	GG (d, a, b, c, M10, 9, 0x02441453)
FF (c, d, a, b, M6, 17, 0xa8304613)	GG (c, d, a, b, M15, 14, 0xd8a1e681)
FF (b, c, d, a, M7, 22, 0xfd469501)	GG (b, c, d, a, M4, 20, 0xe7d3fbc8)
FF (a, b, c, d, M8, 7, 0x698098d8)	GG (a, b, c, d, M9, 5, 0x21e1cde6)
FF (d, a, b, c, M9, 12, 0x8b44f7af)	GG (d, a, b, c, M14, 9, 0xc33707d6)
FF (c, d, a, b, M10, 17, 0xffff5bb1)	GG (c, d, a, b, M3, 14, 0xf4d50d87)
FF (b, c, d, a, M11, 22, 0x895cd7be)	GG (b, c, d, a, M8, 20, 0x455a14ed)
FF (a, b, c, d, M12, 7, 0x6b901122)	GG (a, b, c, d, M13, 5, 0xa9e3e905)
FF (d, a, b, c, M13, 12, 0xfd987193)	GG (d, a, b, c, M2, 9, 0xfcefa3f8)
FF (c, d, a, b, M14, 17, 0xa679438e)	GG (c, d, a, b, M7, 14, 0x676f02d9)
FF (b, c, d, a, M15, 22, 0x49b40821)	GG (b, c, d, a, M12, 20, 0x8d2a4c8a)
Runda 3:	**Runda 4:**
HH (a, b, c, d, M5, 4, 0xfffa3942)	II (a, b, c, d, M0, 6, 0xf4292244)
HH (d, a, b, c, M8, 11, 0x8771f681)	II (d, a, b, c, M7, 10, 0x432aff97)
HH (c, d, a, b, M11,16,0x6d9d6122)	II (c, d, a, b, M14, 15, 0xab9423a7)
HH (b, c, d, a, M14, 23, 0xfde5380c)	II (b, c, d, a, M5, 21, 0xfc93a039)
HH (a, b, c, d, M1, 4, 0xa4beea44)	II (a, b, c, d, M12, 6, 0x655b59c3)
HH (d, a, b, c, M4, 11, 0x4bdecfa9)	II (d, a, b, c, M3, 10, 0x8f0ccc92)
HH (c, d, a, b, M7, 16, 0xf6bb4b60)	II (c, d, a, b, M10, 15, 0xffeff47d)
HH (b, c, d, a, M10, 23, 0xbebfbc70)	II (b, c, d, a, M1, 21, 0x85845dd1)
HH (a, b, c, d, M13, 4, 0x289b7ec6)	II (a, b, c, d, M8, 6, 0x6fa87e4f)
HH (d, a, b, c, M0, 11, 0xeaa127fa)	II (d, a, b, c, M15, 10, 0xfe2ce6e0)
HH (c, d, a, b, M3, 16, 0xd4ef3085)	II (c, d, a, b, M6, 15, 0xa3014314)
HH (b, c, d, a, M6, 23, 0x04881d05)	II (b, c, d, a, M13, 21, 0x4e0811a1)
HH (a, b, c, d, M9, 4, 0xd9d4d039)	II (a, b, c, d, M4, 6, 0xf7537e82)
HH (d, a, b, c, M12, 11,0xe6db99e5)	II (d, a, b, c, M11, 10, 0xbd3af235)
HH (c, d, a, b, M15, 16, 0x1fa27cf8)	II (c, d, a, b, M2, 15, 0x2ad7d2bb)
HH (b, c, d, a, M2, 23, 0xc4ac5665)	II (b, c, d, a, M9, 21, 0xeb86d391)

Tabelul 13.1. *Operaţiile din rundele MD5*

Există diverse implementări ale algoritmului de generare a lui MD, spre exemplu http://www.miraclesalad.com/webtools/md5.php.

Exemplu MD5.



Deci, **MD5(*Funcţii HASH*) = 8e31fb99ef4072b4d3d93014651eb13a**.

Secure Hash Algorithm SHA. NIST împreună cu NSA au propus algoritmul SHA pentru standardul de semnătură digitală. Diferenţa cea mai importantă dintre MD5 şi SHA este lungimea imaginii produse, care în cazul lui SHA este 160 biţi. Cu cât este mai lungă imaginea produsă de funcţie cu atât mai mare este fidelitatea cu care această imagine caracterizează sursa, deci scade probabilitatea ca două mesaje diferite să producă acelaşi hash. Însă ideea de bază, pentru care s-a introdus crearea de imagine a mesajelor este tocmai să se obţină date de lungime redusă care vor fi semnate, asigurând astfel viteza necesară semnării digitale a documentelor. Deci putem afirma că lungimea este de fapt un compromis între fidelitatea imaginii şi viteza procesului de semnare.

Algoritmul SHA a fost proiectat inspirându-se din MD5, astfel multe din operaţiile acestor doi algoritmi sunt similare. Completarea mesajului de intrare la multiplu de 512 biţi este realizat exact la fel ca şi la MD5.

SHA foloseşte cinci variabile de 32 biţi (MD5 foloseşte numai 4), obţinându-se astfel ieşirea de 160 biţi prin concatenarea ieşirilor din cele cinci blocuri.

Menţionăm faptul că funcţiile hash MD5, SHA şi mai recent SHA-1 (Figura 13.2) nu mai oferă rezistenţă secundară a imaginii, însă în ciuda

acestui fapt ele sunt încă folosite în multe aplicaţii. Atacuri asupra SHA au fost anunţate pentru prima oară în două articole non-tehnice ale lui Schneier cu privire la atacurile asupra funcţiilor hash. Pentru soluţii contemporane se recomandă folosirea SHA-256 sau mai puternic, şi nicidecum a MD5 sau SHA-1.

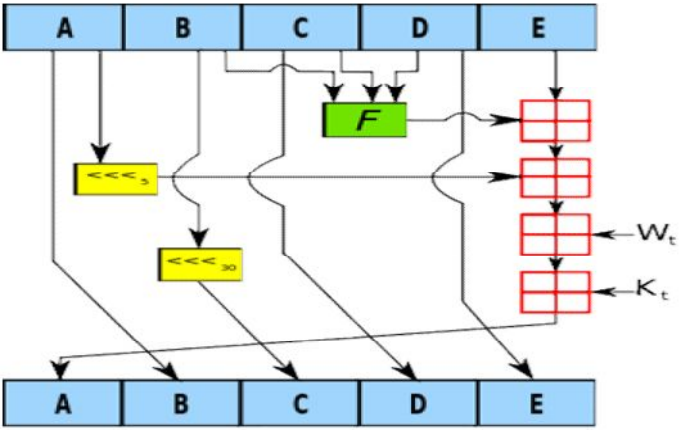


Figura 13.2 *Bucla principală SHA*–1

Un înlocuitor pentru unul dintre cei mai folosiţi algoritmi în securitatea informatică a fost ales în urma unui concurs între criptografi desfăşurat timp de cinci ani. Concursul a fost conceput pentru a linişti experţii în securitate cibernetică. Aceştia fuseseră speriaţi de deficienţele standardului „gold" precedent. Acum se pare că nu avem o nevoie stringentă de un algoritm nou...

Exemplu SHA-1.

SHA–1(*Funcţii HASH*) =745503396156e64889f1ae2386a410f03887ebf7

În data de 2 octombrie 2012 un algoritm numit **Keccak** a fost desemnat câştigătorul competiţiei *Secure Hash Algorithm* - 3 (*SHA*-3) organizat de NIST în Gaithersburg, Maryland. Algoritmii aleşi de NIST sunt consideraţi ca fiind *standardul de aur* în criptografie. La sfârşitul competiţiei anterioare, definitivate în anul 2000, s-a ales **Advanced Encryption Standard** (AES), folosit pe scară largă, de la Skype la Agenţia Naţională de Securitate (US National Security Agency, NSA). NIST a dat startul acestui concurs pentru prima dată în 2007, după ce au început să apară temeri că algoritmii hash existenţi la acea vreme, SHA-1 şi SHA-2, ar putea fi defectuoşi.

Algoritmii hash sunt utilizaţi de către agenţiile guvernamentale şi întreprinderile din întreaga lume pentru a realiza tranzacţii on-line în siguranţă, pentru a stoca parole sigure şi pentru a verifica fişiere digitale şi semnături.

Pentru ca un algoritm hash să funcţioneze, trebuie să fie foarte greu de produs coincidenţe. În limbajul criptografilor acest lucru se numeşte „*coliziune*": două texte diferite care produc acelaşi hash. În anul 2004 criptograful Xiaoyun Wang a descoperit un defect la SHA-1, care a redus în mod drastic timpul necesar pentru a găsi o coliziune făcând nesigure sistemele care utilizează acest algoritm. Până în acest moment NIST a aprobat deja succesorul său, familia de algoritmi SHA-2, creând motive de îngrijorare că defectul s-ar putea extinde şi la SHA-2.

Dar, în timp ce cercetătorii de la NIST analizau cerinţele pentru competitorul SHA-3, aceştia au constatat că algoritmul SHA-2 nu a fost viciat în acest mod. „Ei sunt de fapt algoritmi hash foarte buni, atât în privinţa performanţei, cât şi a securităţii", afirmă Tim Polk, savant de la NIST.

În loc să întrerupă proiectul de cercetare, NIST a decis că SHA-3 ar trebui să fie o opţiune complementară, explică Polk. Algoritmul ideal ar avea o structură diferită criptografic, făcând mai puţin probabil ca un atac asupra SHA-2 să afecteze de asemenea SHA-3. Mai mult chiar, acesta ar fi mai potrivit pentru o gamă mare de dispozitive de calcul.

Dintr-un total de 64 de înregistrări care au avut loc iniţial, NIST a identificat cinci finalişti în 2010. Câştigătorul, algoritmul Keccak, se bazează pe o „construcţie hash de tip burete", numită astfel deoarece funcţia sa este flexibilă, asemeni unui burete fizic. Dacă vă gândiţi că datele de intrare într-o funcţie hash sunt „absorbite" în faza de hashing şi

apoi „stoarse” pentru a produce hash, într-o construcţie de tip burete, stoarcerea mai rapidă produce un hash mai rapid, dar mai puţin sigur şi vice-versa, oferind flexibilitate.

Algoritmul nou este în general mai rapid decât predecesorul său şi prin urmare consumă mai puţină energie, afirmă Gilles Van Assche, unul dintre criptografii aflaţi în spatele algoritmului Keccak, care a fost dezvoltat de firmele de materiale semiconductoare STMicroelectronics din Geneva, Elveţia şi NXP din Eindhoven, Olanda.

Alţi experţi în securitate sunt mai puţin siguri de utilitatea unui nou algoritm. Chiar şi Bruce Schneier, creatorul unuia dintre finaliştii SHA-3, numit Skein, şi-a exprimat îndoiala. „Nu este vorba despre faptul că noile funcţii hash nu ar fi bune, ci despre faptul că nu prea avem nevoie de unul”, a scris Schneier pe blog-ul său, deşi după anunţul făcut de NIST el a adăugat că Keccak este o „alegere bună”.

„Din punct de vedere practic, nu se cunoaşte să fie ceva în neregulă cu algoritmii pe care oamenii îi folosesc în acest moment", afirmă Ross Anderson, un expert în securitate de la Universitatea din Cambridge. Având în vedere faptul că nu există nici o vulnerabilitate evidentă a SHA-2, sistemele pot decide să utilizeze în continuare acest algoritm, mai degrabă decât să îl schimbe, afirmă el. „SHA-2 este încă un algoritm bun, dar poate SHA-3 ar fi mai eficient în unele situaţii”, afirmă Van Assche.

Polk relatează că NIST nu încurajează pe nimeni să abandoneze SHA-2 în favoarea SHA-3. „Ei sunt într-adevăr cei doi algoritmi foarte buni, iar SHA-3 nu este mai bun decât SHA-2 pe toate planurile”.

Tema 14. Semnături digitale.

Semnătura digitală este o schemă matematică pentru demonstrarea autenticităţii unui mesaj sau document electronic. O semnătură digitală validă este un motiv de încredere pentru destinatarul mesajului că acest mesaj a fost creat de către un expeditor cunoscut, astfel încât el nu va putea nega faptul trimiterii mesajului (*autentificarea* şi *non-repudierea*) şi că mesajul nu a fost modificat în drum (*integritatea*). Semnăturile digitale sunt în general utilizate pentru distribuţia de software, în tranzacţiile financiare, precum şi în alte cazuri în care este importantă detectarea falsificării sau manipulării.

Semnătura electronica nu este o semnătură scanată, pictogramă, o poză sau o hologramа şi nici nu este un smart-card.

O schemă de semnătură digitală se bazează pe trei algoritmi:

- algoritmul de *selectare aleatoare a unei chei private* care se va asocia unei chei publice;
- algoritmul de *semnare* care, aplicat unei chei private şi unui document digital, generează semnătura digitală;
- algoritmul de *verificare* a semnăturii digitale, care aplicat cheii publice şi semnăturii digitale, acceptă sau respinge mesajul de conformitate.

Semnăturile digitale reprezintă echivalentul electronic al semnăturilor de mână, acest concept fiind introdus ca funcţionalitate adiţională a criptosistemelor cu cheie publică de către Diffie şi Hellman în 1976, în absenţa unei scheme criptografice pentru acest scop. Obiectivul principal de securitate pe care îl asigură semnăturile digitale îl reprezintă *non-repudierea*, şi anume faptul că o entitate odată ce a semnat o informaţie nu poate nega că a emis acea informaţie şi orice altă entitate neutră poate verifica acest lucru. Semnăturile digitale reprezintă deci o valoare numerică care leagă conţinutul unui mesaj de identitatea unei entităţi. În principiu, orice algoritm asimetric poate fi utilizat pentru crearea unei semnături digitale prin *inversarea rolului cheii publice cu cea privată*, iar primele propuneri de semnături digitale se găsesc în lucrările lui Rivest, Rabin şi ElGamal. Subliniem că şi algoritmi simetrici pot fi folosiţi pentru a crea semnături digitale de tip *one-time* dar acestea sunt rar utilizate în practică.

Pentru o semnătură digitală sunt necesare două proprietăţi de bază:

- În primul rând, o semnătură generată dintr-un mesaj fix şi o cheie fixă privată ar trebui să verifice autenticitatea acestui mesaj utilizând cheia publică corespunzătoare.
- În al doilea rând, ar trebui să fie imposibil de generat o semnătură validă pentru o entitate care nu posedă cheia privată.

Mai menţionăm câteva proprietăţi ale semnăturilor digitale:

- Trebuie să fie uşor de calculat doar de către cel care semnează mesajul (funcţia de semnare trebuie să fie uşor de calculat).
- Trebuie să fie uşor de verificat de către oricine (funcţia de verificare trebuie să fie uşor de calculat).
- Trebuie să deţină o durată de viaţă corespunzătoare, adică semnătura să nu poată fi falsificată până când nu mai este necesară scopului în care a fost creată.

Precizăm încă odată cele trei motive principale pentru care se recomanda folosirea semnăturilor digitale:

Autenticitatea. Deşi documentele digitale pot să includă informaţii despre identitatea celui care le-a emis, aceste informaţii pot să nu fie corecte. Semnătura digitală poate fi folosită pentru autentificarea sursei documentului. Atunci când dreptul de proprietate asupra unei semnături digitale aparţine unei anumite persoane, semnătura digitală arată că documentul a fost eliberat de către acea persoană. Importanţa acestui aspect apare în dovedirea autenticităţii documentelor digitale în context financiar-contabil. De exemplu actele contabile ale unei firme sunt trimise corect de către firma de contabilitate către administratorul firmei. Daca acesta va modifica aceste acte încercând să schimbe informaţiile financiare sau orice fel de informaţii transmise ca fişier PDF/A semnat electronic, pentru a putea obţine un credit de la o bancă, în momentul în care banca deschide acele documente, semnătura electronică va fi invalidată, deci firma de contabilitate nu va putea fi considerată responsabilă de conţinutul documentului, persoana care a transmis documentul modificat nu va putea fi urmărită în justiţie pentru fals şi uz de fals, iar banca va fi asigurată că nu cade în capcana acordării unui credit pe baza unor documente false.

Integritatea. Semnătura digitală aplicată unui document electronic reprezintă o garanţie a integrităţii documentului atunci când este validată

de o autoritate publică. Cheia aleatoare se creează pe baza unor criterii multiple care includ printre altele şi detalii despre conţinutul documentului. Orice modificare a conţinutului unui document digital înseamnă o cheie aleatoare nouă diferită de cheia aleatoare folosită pentru aplicarea semnăturii digitale. În clipa în care se solicită validarea documentului cele două chei aleatoare vor fi diferite, iar documentul se va putea considera ca având conţinutul alterat.

Imposibilitatea repudierii (non-repudiere). Odată ce este emis un document digital semnat electronic, atâta vreme cât semnătura electronică este validă pe documentul digital, autorul documentului nu îşi poate declina răspunderea pentru conţinutul documentului cu semnătura electronică validă. În plus, nu poate nega faptul că documentul a fost semnat personal, deoarece legislaţia în vigoare prevede faptul că deţinătorul unei semnături digitale nu are dreptul să înstrăineze sau împrumute *token*-ul (dispozitivul criptografic) pe care există certificatul digital calificat pe care îl deţine. Din acest punct de vedere, exemplul de mai sus în care firma de contabilitate transmite documentele financiare firmei pentru care le întocmeşte, dacă a comis greşeli în întocmirea acestor acte, rămâne responsabilă pentru datele prezentate, câta vreme documentul digital transmis poartă o semnătură validă.

Pentru o semnătură digitală a mesajului m de către entitatea A putem folosi notaţia $Sig_A(m)$.

Există două categorii distincte de semnături digitale:

Semnături digitale cu recuperarea mesajului. La utilizarea acestor semnături mesajul poate fi recuperat direct din semnătura digitală. Cel mai simplu exemplu de construcţie este prin inversarea rolului cheii publice şi private în cazul schemei RSA. Pentru a evita fraudarea lor este obligatorie aplicarea unei funcţii de redundanţă asupra mesajului după care semnătura propriu-zisă se aplică asupra mesajului redundant.

Semnături digitale cu anexă (sau cu *apendice*). Acestea sunt semnături digitale din care mesajul nu poate fi recuperat, drept care este trimis adiţional ca anexă la semnătura digitală. Se pot construi uşor prin aplicarea unei funcţii *hash* asupra mesajului şi semnarea hash-ului obţinut. Datorită eficienţei computaţionale în semnarea mesajelor de dimensiuni mari (deoarece se semnează efectiv doar hash-ul mesajului care are o lungime fixă pentru fiecare algoritm hash indiferent de lungimea mesajului), aceste semnături sunt cele mai utilizate în practică. Totodată orice semnătură

digitală cu recuperarea mesajului poate fi uşor convertită în semnătură cu anexă.

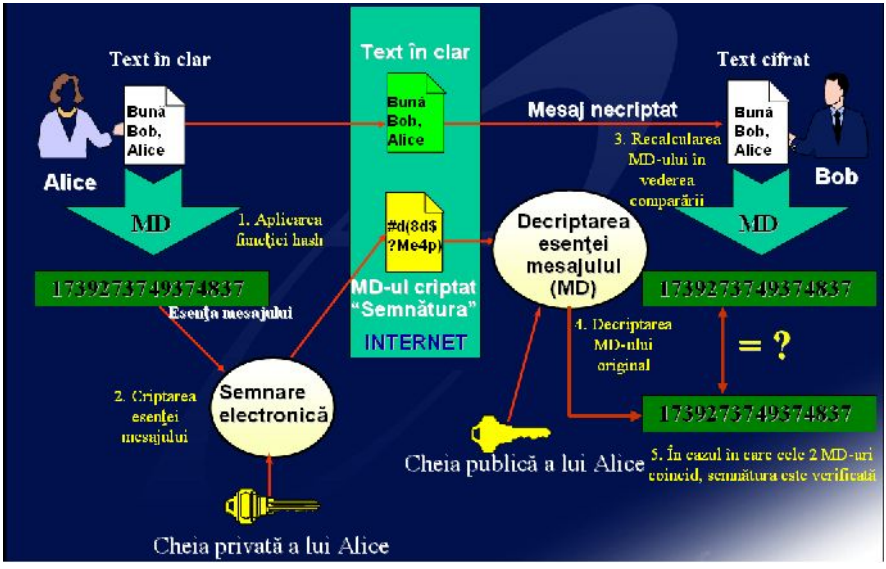


Figura 14.1. *Schema unei semnături digitale fără criptarea mesajului*

Semnăturile digitale de asemenea pot fi clasificate în *semnături deterministe* respectiv *randomizate* după cum algoritmul de semnare foloseşte valori aleatoare şi nu returnează aceeaşi semnătură pentru acelaşi mesaj de fiecare dată.

O altă clasificare a algoritmilor de semnătură mai poate fi în *algoritmi de unică folosinţă* (one-time) sau pentru *folosire multiplă* (multiple-time).

Să analizăm în continuare câteva dintre schemele de semnătură digitală aplicate în prezent.

Schema de semnătură RSA

1. *Generarea cheilor*. Entitatea *A* execută următoarele în conformitate cu algoritmul de generare a cheilor *RSA*:
   - Generează două numere prime mari p şi q;
   - Calculează $n = p \cdot q$ şi $\varphi(n) = (p-1) \cdot (q-1)$;
   - Selectează aleator un număr întreg e, $1 < e < \varphi(n)$, astfel încât *cmmdc* $(e, \varphi(n)) = 1$;

- Calculează numărul întreg d, 1< d < φ(n), astfel încât $e \cdot d = 1 \pmod{\varphi(n)}$;
- Cheia publică a entităţii A este perechea (***e***, ***n***); cheia privată este d sau (***d***, ***n***).

2. *Generarea semnăturii*. Entitatea A execută următoarele:
   - Calculează $S = [H(m)]^d \pmod{n}$, unde H este o funcţie hash;
   - Semnătura mesajului m este S.
3. *Verificarea semnăturii*. Pentru a verifica semnătura S a mesajului m, entitatea B execută următoarele:
   - Obţine cheia publică autentică (e, n) a entităţii A;
   - Calculează $H_1=S^e$ şi $H_2 = H(m) \pmod{n}$. În cazul în care $H_1 = H_2$ semnătura e verificată, în caz contrar înseamnă că s-a întâmplat ceva în canalul de comunicaţii sau A vrea să-l înşele pe B.

Exemplu.

- Fie $p = 53$ şi $q = 61$, două numere prime secrete ale lui A.
- Se calculează $n = 53 \cdot 61 = 3233$ şi $\varphi(n)=(p-1)\cdot(q-1)=52 \cdot 60=3120$.
- Se alege o cheie secreta $d = 71$ ($cmmdc(71, 3120) = 1$).
- Se calculează cheia publică $e = 791$ ($71 \cdot d = 1 \pmod{3120}$);
- Se consideră un document al cărui rezumat $S=H(m)=13021426$.
- Deoarece valoarea lui S depăşeşte mărimea modulului n =3233, se va sparge acest rezumat în două blocuri (1302 şi 1426), pe care A le va semna separat, folosind cheia privată d = 71:

  $(1302^{71}) \bmod 3233 = 1984;$

  $(1426^{71}) \bmod 3233 = 2927;$
- Semnătura electronică obţinută este S = 1984 2927;
- Se transmite S obţinut la pasul anterior şi M (textul clar – fără secretizare).
- B primeşte pachetul S şi M, după care calculează $H(M)$ în două moduri şi le va compara:
  - va calcula $H(M)$ cu cheia publică a lui A:

    $(1302^{71}) \bmod 3233 = 1984;$

    $(1426^{71}) \bmod 3233 = 2927;$

    deci H_1 = 1302 1426;

- va calcula $H(M)$ asupra mesajului primit (la fel cum a procedat A):

$$H_2 = H(M) = 1302\ 1426;$$

- Deoarece H_1 este identic H_2 atunci cu siguranţă semnătura este validă.

Schema de semnătură ElGamal

Schema de semnătură ElGamal a fost propusă de ElGamal împreună cu schema de criptare cu chei publice.

1. *Generarea cheilor*. Fiecare entitate generează cheia publică şi cheia privată corespunzătoare.

   Entitatea A execută următoarele:

   - Generează un număr prim mare p şi un generator α al grupului multiplicativ $\boldsymbol{Z}_p^*$;
   - Selectează aleator un număr întreg a astfel încât $1 \le a \le p-2$;
   - Calculează $y = \alpha^a \bmod p$.
   - Cheia publică a entităţii A este (p, α, y); cheia privată este a.
2. *Generarea semnăturii*. Entitatea A semnează un mesaj binar m de o lungime arbitrară.

   Pentru aceasta, entitatea A execută următoarele*:*

   - Selectează aleator un număr întreg secret k, $1 \le k \le p-2$, astfel încât $cmmdc(k, p-1) = 1$;
   - Calculează $r = \alpha^k \bmod p$ şi $k^{-1} \bmod (p-1)$;
   - Calculează $s = k^{-1}(H(m) - a \cdot r) \bmod (p-1)$, unde H este o funcţie hash;
   - Semnătura mesajului m este perechea $(\boldsymbol{r}, \boldsymbol{s})$.
3. *Verificarea semnăturii*.

   Pentru a verifica semnătura (r, s) a mesajului m, entitatea B execută următoarele:

   - Obţine cheia publică autentică (p, α, y) a entităţii A ;
   - Verifică dacă $1 \le r \le p-1$ (dacă această inegalitate nu are loc, semnătura (r, s) nu e validă);
   - Calculează $v_1 = y^r r^s \bmod p$;
   - Calculează $H(m)$ şi $v_2 = \alpha^{H(m)} \bmod p$, unde H este o funcţie hash;
   - Semnătura $(\boldsymbol{r}, \boldsymbol{s})$ este acceptată dacă şi numai dacă $v_1 = v_2$.

Exemplu.

- *A* generează numărul prim p=2357 şi generatorul α=2 al grupului $\boldsymbol{Z}^*_{2357}$;
- *A* alege cheia privată a = 1751 şi calculează $y = \alpha^a \bmod p = 21751 \bmod 2357 = 1185$.
- Cheia publică a lui *A* este (p = 2357, α = 2, y = 1185), iar cheia sa privată este a = 1751.
- Pentru a semna mesajul m cu $H(m)$ =1463, *A* selectează aleator un întreg k =1529, calculează
  $r = \alpha^k \bmod p = 2^{1529} \bmod 2357 = 1490$ şi
  $k^{-1} \bmod (p-1) = 245$;
- *A* calculează s=245·(1463–1751·1490) mod 2356=1777;
  semnătura mesajului m = 1463 este perechea (r =1490, s =1777);
- Pentru verificarea semnăturii, *B* calculează
  $v_1 = 1185^{1490} \cdot 1490^{1777} \bmod 2357 = 1072$,
  $H(m)$ =1463 şi
  $v_2 = 2^{1463} \bmod 2357 = 1072$;
- Entitatea *B* acceptă semnătura deoarece $v_1 = v_2$.

Standardul DSS (Digital Signature Standard) de semnătură digitală

DSA este algoritmul de semnătură digitală al standardului DSS, elaborat de NIST în august 1991. Este un standard foarte controversat în literatura de specialitate deoarece este destinat să înlocuiască standardul "de facto" al domeniului, RSA. Algoritmul SDA se bazează pe un aparat matematic derivat din metoda ElGamal, gardul de securitate al său fiind bazat la fel pe problema dificultăţii calculului logaritmilor într-un câmp finit.

1. *Generarea cheilor*. Fiecare entitate generează cheia publică şi cheia privată corespunzătoare.
   Entitatea *A* execută următoarele:
   - Generează un număr prim q astfel încât $2^{159} < q < 2^{160}$;
   - Generează un număr prim p astfel încât $2^{512} \le p < 2^{1024}$ şi $q|(p-1)$;
   - Selectează un generator α pentru grupul ciclic $\boldsymbol{Z}_p$ de ordin q;
   - Alege un element $g \in \boldsymbol{Z}_p^*$ şi calculează $\alpha = g^{(p-1)/q} \bmod p$;
     dacă α = 1, atunci alege alt element g;

- Se selectează un număr întreg a astfel încât $1 \le a \le q-1$;
- Se calculează $y = \alpha^a \bmod p$;
- *Cheia publică* a entităţii *A* este (***p***, ***q***, **α**, ***y***), iar *cheia privată* este ***a***.

2. *Generarea semnăturii*. Entitatea *A* semnează un mesaj m astfel:
   - Selectează aleator un număr întreg k astfel încât $0 < k < q$;
   - Calculează $r = (\alpha^k \bmod p) \bmod q$.
   - Calculează $k^{-1} \bmod q$;
   - Calculează $s = k^{-1} (H(m) + a \cdot r) \bmod q$, unde H este o funcţie hash;
   - Semnătura mesajului m este perechea (***r***, ***s***).
3. *Verificarea semnăturii*. Pentru a verifica semnătura (***r***, ***s***) a mesajului *m*, entitatea *B* execută următoarele:
   - Obţine cheia publică autentică (***p***, ***q***, **α**, ***y***) a entităţii *B*;
   - Verifică dacă $0 < r < q$ şi $0 < s < q$. Dacă aceste inegalităţi nu au loc, semnătura (***r***, ***s***) nu e validă;
   - Calculează $w = s^{-1} \bmod q$ şi $H(m)$;
   - Calculează $u_1 = w \cdot H(m) \bmod q$ şi $u_2 = r \cdot w \bmod q$;
   - Calculează $v = (\alpha^{u_1} y^{u_2} \bmod p) \bmod q$;
   - Semnătura (***r***, ***s***) a mesajului *m* este acceptată dacă şi numai dacă $v = r$.

Exemplu.

- Entitatea *A* generează numerele prime p=124540019 şi q=17389, astfel încât $q \mid p-1$;
- *A* selectează $g = 110217528 \in \boldsymbol{Z}_p^*$ şi calculează $\alpha = g^{7162} \bmod p = 10083255$;
- *A* selectează un număr întreg $a = 12496$ astfel încât $1 \le a \le q-1$ şi calculează

  $y = \alpha^a \bmod p = 10083255^{12496} \bmod 124540019 = 19946265$;
- Cheia publică a lui *A* este (***p***, ***q***, **α**, ***y***) = (124540019, 17389, 10083255, 119946265) ;
- Cheia privată a lui *A* este $a = 12496$.

  Pentru a semna un mesaj *m*:
- *A* alege aleator un număr întreg $k = 9557$ şi calculează $r = (\alpha^k \bmod p) \bmod q$;
- $r = (10083255^{9557} \bmod 124540019) \bmod 17389 =$

=27039929 mod 17389=34.

- *A* calculează
  k^{-1} mod q=7631,
  $H(m)$=5246 şi apoi
  s=7631·(5246+12496·34) mod q = 13049.
- Semnătura mesajului m este perechea (***r***, ***s***)= (**34**, 13049).

Pentru a verifica un mesaj, entitatea B:

- Obţine cheia publică autentică (***p***, ***q***, ***α***, ***y***) a entităţii *B*;
- Verifică dacă 0 < 34 < 17389 şi 0 < 13049 < 17389. Dacă aceste inegalităţi nu ar avea loc, semnătura (***r***, ***s***) nu ar fi fost validă;
- Calculează $w = s^{-1}$ mod q =1799;
- Calculează
  $u_1 = w \cdot H(m)$ mod q = 5246·1799 mod 17389 = 12716
  $u_2 = r \cdot w$ mod q = 34·1799 mod 17389=8999;
- *B* calculează $v = (\alpha^{u_1} y^{u_2} \bmod p) \bmod q$:
  (10083255 ^{12716} ·119946265 ^{8999} mod 124540019) mod 17389 = =27039929 mod 17389 = **34**.
- Deoarece $v = r$, *B* acceptă semnătura.

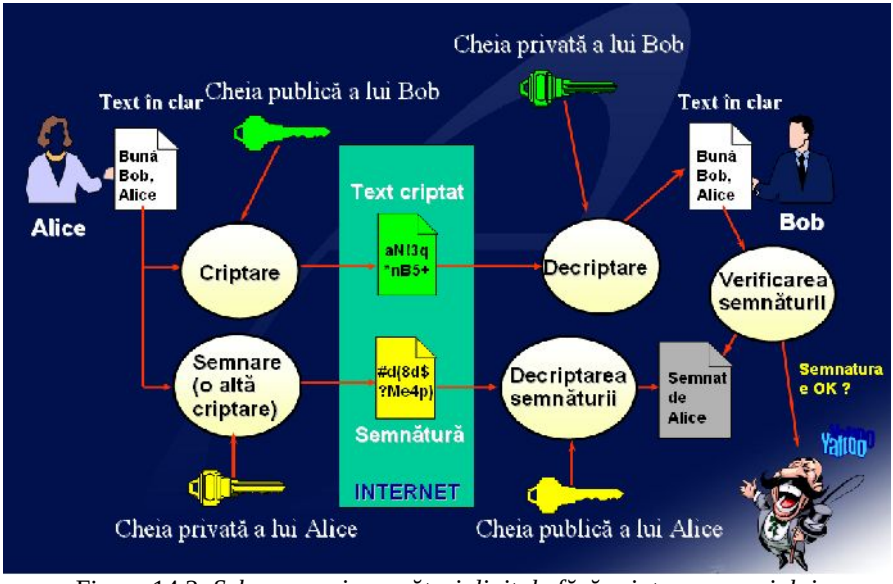


Figura 14.2. *Schema unei semnături digitale fără criptarea mesajului*

În exemplele de mai sus documentul a fost trimis fără a fi criptat, fiind necesară numai semnătura digitală. În cazul în care este necesar, textul clar este criptat cu unul dintre sistemele de criptare şi se trimite lui *B* textul cifrat împreună cu semnătura digitală, după care *B* îl descifrează cu cheia respectivă apoi efectuează verificarea semnăturii, având deja textul clar. În Figura 14.1 este prezentată schema aplicării semnăturii digitale fără criptarea mesajului, iar în Figura 14.2 – cu criptarea acestuia.

Tema 15. Atacuri criptografice

Aşa cum nu există bine fără rău, sau noapte fără zi aşa şi nu avem criptografie fără criptanaliză.

Criptanaliza este ştiiţa spargerii cifrurilor şi ce se ocupă de obţinerea valorii iniţiale a informaţiei criptate fără a avea acces la informaţia secretă, adică la cheia necesară pentru acest lucru.

Persoana care se ocupă cu criptanaliza se numeşte *Criptanalist*.

Rezultatul criptanalizei unui cifru concret se numeşte *atac criptografic* asupra cifrului dat. Atacurile criptografice sunt concepute pentru a submina securitatea algoritmilor de criptare şi utilizate pentru a încerca decriptarea datelor fără a acces prealabil la cheie.

Scopul metodelor de criptanaliză este descoperirea mesajelor în clar şi/sau a cheii din mesajul criptat. Orice cifru este creat în scopul de a asigura confidențialitatea informației protejate și, reieșind din acest concept, întotdeauna se vor găsi oamnei care doresc să obţină accesul la informaţia dată.

Orice încercare de obținere a textului în clar din textul criptat, fără a deține cheia secretă, este considerată drept atac criptanalitic. Analiza criptografică studiază metode de atac pornind de la informaţii minimale despre cheile de criptare, algoritmii utilizați, protocoalele de autentificare, segmente de text clar şi segmentele corespondente din textul criptat, sau doar pe baza unuia sau a unui set de texte criptate utilizând același algoritm.

În esență, se încearcă determinarea unui punct vulnerabil al algoritmului, care să poată fi exploatat folosind metode pentru care timpul de căutare să fie considerabil mai mic decât timpul necesar verificării tuturor combinațiilor de chei posibile (atac forţa brută).

Nu există încă un sistem criptografic despre care să se poată afirma că este pe deplin sigur, dar pot fi considerate sigure acele criptosisteme pentru care atacurile cunoscute necesită un timp mult prea îndelungat pentru a putea fi considerate practice. În continuare sunt descrise pe scurt, cele mai cunoscute atacuri, demonstrate și verificate de matematicieni, informaticieni și criptanaliști.

Se cunosc mai multe tipuri de atacuri criptografice. O categorie aparte sunt atacurile bazate pe *forţă*. Acestea sunt atacurile ce se realizează prin forţă brută (*brute force*), adică aplică o metodă exhaustivă de căutare prin

încercarea tuturor combinaţiilor posibile fie de chei de criptare, fie de simboluri din text pentru deducerea textului în clar (de exemplu, la metodele de criptare prin substituţia sau transpoziţia literelor din mesaje de tip text). Complexitatea acestui atac este în funcţie de cantitatea tuturor variantelor posibile.

Acest tip de atac este unul genereal, adică poate fi aplicat la orice algoritm de criptare. Din acest motiv în elaborarea sistemelor de criptare autorii încearcă să obţină ca acest atac să fie cel mai eficient, comparativ cu celelalte metode de spargere. Sistemul se proeictează astfel încât forţa brută să aibă un spaţiu al soluţiilor suficient de voluminos pentru ca rezultatul aplicării forţei brute să nu fie obţinut pe percursul a câtorva ani, sau uneori şi secoole.

În baza complexităţii aplicării forţei brute se face evaluarea securităţii sistemului. În particular, cifrul se consideră sigur dacă nu există o metodă de spargere semnificativ mai rapidă decât forţa brută.

Atacurile criptografice bazate pe forţa brută sunt cele mai universale, dar şi cele mai îndelungate. În legătură cu aceasta există deja şi se mai elaborează încontinuu posibilităţi de optimizare a metodei forţei brute. Printre aceste metode optimizate se numără următoarele:

- Metoda ramifică şi mărgineşte (*Branch and Bound method*);
- Metoda calculelor paralele (*Parallel calculation method*);
- Метод радужных таблиц (*Rainbow tables method*).

Metoda *ramifică şi mărgineşte* reprezintă un algorim de căutare a soluţiilor optine pentru diverse probleme. Esenţa lui constă în separarea submulţimii de soluţii admisibile care nu conţine soluţii optime. Metoda a fost propusă pentru prima dată de către A. H. Land şi A. G. Doig în 1960 pentru programarea discretă.

Calculul paralel este execuția în paralel pe mai multe procesoare a aceloraşi instrucțiuni, sau şi a unor instrucțiuni diferite, cu scopul rezolvării mai rapide a unei probleme, de obicei special adaptată sau subdivizată. Ideea de bază aici constă în divizarea mulţimii soluţiilor în *N* submulţimi, fiecare din ele fiind mult mai „mică” decât originalul, astfel realizarea „forţei brute” va necesita de *n* ori mai puţin timp, în funcţie de numărul de submulţimi ale partiţiei făcute. Problema fundamentală aici constă în determinarea şi divizarea mulţimii soluţiilor. „Forţa brută” se aplică până când un procesor nu a găsit soluţia (cheia) necesară.

Ideia metodei *Rainbow tables* constă în determinarea prealabilă a spaţiului cheilor într-o formă comprimată. Acest spaţiu comprimat va conţine o colecție de chei cu proprietăţi de „checkpoint” (punct de control). Atunci când determinăm spațiul comprimat al cheilor, selectând punctele de control, se identifică o colecție relativ mică de chei (câteva miliarde), despre care se ştie că conţin cheia căutată. Pentru restabilirea cheii la valoarea dată a funcţiei hash se palică o funcţie de reducere şi se face căutare aîn tabel. Dacă nu sunt găsite coincidenţe se aplică din nou funcţia hash şi funcţia de reducre. Procesul continuă până când nu se găseşte o coincidenţă. După găsirea acestei coincidenţe se restabileşte şirul care o conţine pentru a determina valoarea omisă care şi este cheia căutată. Aşadar căutarea cheii cu „tabele curcubeu” constă din două etape; în primul rând, începem căutarea pentru a găsi un „punct de control”, și apoi, în al doilea rând, căutam cheile definite de punctul de control pentru a găsi cheia căutată.

Avantajul faţă de *forţa brută* este că pot fi create mai multe „puncte de control”, și, prin urmare, procesul de căutare va nimeri unul dintre aceste puncte de control mult mai repede decât am aștepta până când nimerim peste cheia țintă singuratică. Desigur aici un dezavantaj este faptul că „punctele de control” trebuie să fie pre-calculate și stocate. Deci trebuie găsit un compromis între timpul de căutare şi memoria de stocare - timpul de căutare se reduce prin mărirea numărului de „puncte de control” însă cu preţul măririi costului de pre-calcul și de stocare a tabelelor.

„Tabelele curcubeu” pot fi folosite de asemena şi la spargerea diverselor parole transformate cu ajutorul funcţiilor hash dificeil de inversat.

În tabelul 15.1 este prezentată dependenţa timpului de spargere prin forţa brută de lungimea cheii (sau a parolei). În acest tabel este arătat timpul aproximativ pentru verificarea tuturor variantelor posibile ale unei chei ce poate fi alcătuită din 36 caractere (26 litere şi 10 cifre) cu viteza de prelucrare de 100000 de variante pe secundă. Dacă însă se vor adăuga litere din alţi regiştri timpul va creşte de câteva ori. Din tabel putem observa că cheile de lungime mai mică decât 8 sunt foarte vulnerabile la forţa brută.

Lungime cheie (caractere)	Variante posibile	Timpul de atac
1	36	< 1 s
2	1296	< 1 s
3	46656	< 1 s
4	1679616	17 s
5	60466176	26 s
6	2176782336	6 ore
7	78364164096	9 zile
8	$2{,}8211099 \cdot 10^{12}$	11 luni
9	$1{,}0155995 \cdot 10^{14}$	32 ani
10	$3{,}6561584 \cdot 10^{15}$	1162 ani
11	$1{,}3162170 \cdot 10^{17}$	41823 ani
12	$4{,}7383813 \cdot 10^{18}$	1505615 ani

Tabelul 15.1. Timpul necesar pentru atacul în forţă brută

De rând cu forţa brută se mai aplică şi *metodele statistice* de atac. Aceste metode se divizează în două subcategorii:

- metode de criptanaliză a proprietăţilor statistice ale gamei de criptare;
- metode de criptanaliza a complexităţii şirului.

Prima subcategorie studiază şirurile la ieşirea algoritmilor de criptare. În acest caz criptanalistul cu ajutorul diverselor teste statistice încearcă să găsească valoarea următorului bit al şirului cu o probabilitate mai mare decât probabilitatea alegerii aleatoare.

În cazul al doilea criptanalistul încearcă să genereze şiruri analogice cu gama însă aplicând metode mult mai simple.

Din diversitatea de tipuri şi metode cel mai cunnoscute de atac criptografic, demonstrate, verificate şi aplicate de matematicieni, informaticieni și criptanaliști punem menţiona atacurile cu text clar sau text cifrat:

- ***Atac cu text cifrat*** (*ciphertext-only attack*) interceptat, prin analiza căruia se încearcă găsirea textului original sau a cheii de criptare. Acest atac se bazează pe informaţii referitoare la secvenţe de text cifrat și este una dintre cele mai dificile metode criptografice din

cauza informaţiei sumare pe baza căreia trebuie să se deducă informaţii referitoare la textul clar sau la chei.

- ***Atac cu text clar cunoscut*** (*known plaintext attack*), este un atac în care criptanalistul are acces nu doar la textul cifrat, ci şi la textul clar corespunzător. El încearcă să descopere o corelație între cele două pentru a găsi cheia de criptare sau pentru a crea un algoritm care îi va permite să descifreze orice mesaje criptate cu această cheie. Textele în clar necesare pentru acest atac pot fi obţinute prin diverse metode, de exemplu dacă se cunoaşte că se trimte un fişier cifrat cu un nume ştiut, atunci din extensia fişierului se pot face concluzii despre conţinutul anumitor fragmente ale fişierelor, de exemplu a heeder-ului. Acest atac este mai puternic decât atacul cu text cifrat.
- ***Atac cu text cifrat ales*** (*chosen ciphertext attack*), este un atac în care criptanalistul alege un text cifrat și încearcă să găsească textul clar potrivit. Acest lucru se poate face cu o decriptare oracul (o mașină care decriptează fără a demasca cheia). Atacul este aplicat la criptarea cu cheie publică – se începe cu un text cifrat şi căutări de potrivire de date ale textului clar postate public.
- ***Atac cu text clar ales*** (*chosen plaintext attack*), este un atac în care criptanalistul poate cripta un text clar la alegerea sa și studia textul cifrat rezultat. Scopul criptanalistului este acelaşi cala atacul cu text clar cunoscut: de a afla cheia de criptare sau de a găsi o altă metodă pentru descifrarea mesajelor cifrate cu aceeaaşi cheie. Însă criptanalistul trebuie să mai aibă posibilitatea de a alege câteva texte în clar şi să obţină rezultatul cifrării lor. Obţinerea textului cifrat respectiv pentru textul clar dat uneori se poate face prin crearea şi transmiterea unui mesaj necifrat în numele unia din utiliaztoriicare folosesc criptarea. În cazul coincidenţii unor factori acest mesaj poate fi cifrat şi retransmis înapoi. Acest atac este cel mai des utilizat în criptografia asimetrică, în cazul în care criptanalistul are acces la o cheie publică.
- ***Atac cu text clar ales adaptiv*** (a*daptive chosen plaintext attack*), este un caz particular, mai confortabil, al atacului cu text clar ales. Confortul atacului constă în faptul că pe lângă posibilitatea alegerii textului clar, criptanalistul poate lua decizia de a cifra un careva

text clar în baza operaţiilor de cifrare deja efectuate. Cu alte cuvinte la realizarea atacului cu text clar ales criptanalistul alege doar un bloc mare al textuli clar pentru a fi cifrat şi apoi, în baza datelor obţinute începe spargerea sistemului. În cazul organizării atacului cu text clar ales adaptiv criptanalistul poate obţine rezultatul cifrării oricărui bloc al textului clar pentru a acumula datele care îl interesează şi care vor fi luate în seamă la alegerea ulterioară a blocurilor textului clar etc. Anume adaptiviattea (posibilitatea feedback-ului) îi dă un avantaj atacului cu text calr ales adaptiv.

- ***Atac cu text cifrat ales adaptiv*** (*Adaptive Chosen Ciphertext Attack*), este un atac analogic atacului cu text clar ales adaptiv.

În continuare vom menţiona alte tipuri şi metode de tac utilizate în prezent.

Atacul cu chei alese (*Chosen key attacks*) în care atacatorul nu posedă informaţii complete despre chei ci doar câteva informaţii disparate despre relaţiile dintre nişte chei. Acest tip de atac se bazează pe faptul că criptanalistul poate analiza activitatea algoritmului de criptare, care utilizează mai multe chei. Criptanalistul iniţial nu cunoaşte valoarea exactă a cheilor, dar el ştie unele relaţii matematice care relaţionează cheile. Un exemplu poate fi cazul în care criptanalistul a constatat că în ultimii 80 de biţi a tuturor cheilor sunt aceleaşi, deşi valorile acestor biţi pot fi necunoscute. Este un atac foarte puţin folosit şi aproape impracticabil.

Atacul de tip dicţionar (*dictionary attack*) este o metodă criptanalitică în care atacatorul pregăteşte şi memorează un tabel cu corespondenţe text clar - text criptat de tipul perechilor $(P_i\, C_i = E_{Ki}(P), K_i)$ sortate după C_i. Ulterior, atacatorul monitorizează comunicaţia şi în momentul în care va găsi un text criptat C_j care se regăseşte în tabelul său va găsi imediat cheia de criptare K_j.

Atacul zilei de naştere (*Birthday attack*), se bazează pe cunoscutul paradox al „zilei de naştere” şi a variantelor sale (într-un grup de 23 de persoane, probabilitatea să existe două dintre ele născute în aceeaşi zi din an este peste 0,5; în general, într-un grup de $\sqrt{k}$ numere aleatoare având k valori posibile, probabilitatea ca cel puţin două să fie egale este în jur de 0,5). Problema poate fi astfel generalizată: dacă o funcţie f: $A \rightarrow B$ poate lua oricare din cele n valori din mulţimea B cu probabilităţi egale, atunci

după calculul funcţiei pentru $\sqrt{n}$ valori diferite este foarte posibil să găsim o pereche de valori x_1 şi x_2 astfel încât $f(x_1) = f(x_2)$. Evenimentul reprezintă o coliziune (Bellare şi Kohno, 2004), iar pentru funcţii cu distribuţie impară, coliziunea poate apărea şi mai devreme. Semnătura digitală este susceptibilă de a fi supusă unui astfel de atac.

Atac cu întâlnire la mijloc (*Meet-in-the-middle attack*) este similar cu atacul zilei de naştere, cu excepţia faptului că în acest caz analistul are o flexibilitate mai mare. În loc să aştepte coincidenţa a două valori într-o singură mulţime de date, analistul poate căuta o intersecţie a două mulţimi.

Presupunem că atacatorul cunoaşte o mulţime de texte în clar ***P*** şi texte criptate ***C*** cu cheile k_1 şi k_2. Atunci el poate calcula $E_K(P)$ pentru toate cheile posibile K şi să memoreze rezultatele, apoi poate calcula $D_K(C)$ pentru fiecare K şi să compare cu rezultatele memorate - dacă va găsi o coincidenţă este ca şi cum ar fi găsit cele două chei şi poate verifica direct pe textul în clar şi cel criptat. Dacă dimensiunea cheii este n, atacul va folosi doar 2^{n+1} criptări în contrast cu un atac clasic, care ar avea nevoie de 2^{2n} criptări.

Atacul omului din mijloc (*Man-in-the-middle attack*) descrie situaţia când un atacator are posibilitatea să citească şi să modifice mesajele schimbate între doi corespondenţi fără ca cele două părţi să sesizeze faptul că metoda de comunicare între ei a fost compromisă.

Atacul începe de obicei cu ascultarea canalului şi se termină cu încercarea criptanalistului de a înlocui mesajul interceptat, extragerea informaţiilor utile din el, redirecţionarea mesajului la unele resurse externe. Fie că subiectul *A* planifică transmiterea spre subiectul *B* a unei informaţii oarecare. Subiectul *C* posedă cunoştinţe despre structura şi proprietăţile metodei de transmitere a datelor, precum şi a însuşi faptului transmiterii acelei informaţii pe care intenţionează să o intercepteze (de exemplu cheia privată). Pentru săvârşirea atacului *C* se „prezintă" lui *A* drept *B*, iar lui *B* drept*A*. Subiectul *A* consideră eronat că transmite informaţia lui *B* şi o trimite lui *C*, care la rândul său efectuează unele operaţii cu ea (o copiază sau o modifică în scopuri personale) şi o transmite lui *B*. Ultimul consideră că informaţia a fost primită direct de la *A*.

Posibilitatea unui astfel de atac rămâne o problemă serioasă pentru sistemele bazate pe chei publice.

Atacul în reluare (*replay atack*) este un atac în care atacatorul memorează o sesiune de comunicare în ambele sensuri (mesajele schimbate de ambii corespondenţi) sau bucăţi din sesiune (Schneier, 1996) şi (Menezes şi colab., 1996). Ideea atacului nu este de a decripta o sesiune de comunicare, ci de a crea confuzii şi mesaje false.

Atacul cu chei relaţionate (*related keys attack*). În acest caz atacatorul descoperă o relaţie între un set de chei şi are acces la funcţiile de criptare cu astfel de chei relaţionate. Scopul declarat este de a găsi chiar cheile de criptare (Knuth 1998; Biham, 1994). Algoritmi ca IDEA, GOST, RC2 şi TEA au prezentat slabiciuni cand au fost supuse atacului (Kelsey şi colab., 1996; 1997).

Atacul prin alunecare (*slide attack*) poate fi văzut ca o variantă a atacului cu chei relaţionate în care relaţiile sunt definite pe aceeaşi cheie. Atacul este eficient în cazul unor procese iterative sau recursive (algoritmi simetrici de tip şir sau bloc) care prezintă grade de similitudine între cicluri succesive ale procesului iterativ (Biryukov şi Wagner, 1999; 2000). Complexitatea atacului este independentă de numărul de cicluri ai algoritmului. Slăbiciuni în cazul acestui atac au fost relevate în algoritmul Feistel şi chiar în cazul SHA-1 (Saarinen, 2003).

Atacul de corelaţie (*correlation attack*) se efectuează asupra generatorului de filtrare din cifrurile şir bazate pe generatoare de tip LFSR (Linear Feedback Shift Register), în două faze: întâi se determină o funcţie între şirul de biţi cheie generat şi biţii tregistrului de deplasare, după care şirul de chei este interpretat ca o versiune afectată de zgomot a şirului generat de LFSR.

Siegenthaler a dezvoltat versiunea originală a atacului de corelaţie care presupune o căutare exhaustivă aplicată în toate fazele LFSR-ului pentru a găsi cel mai înalt grad de corelaţie (Seigenthaler, 1985). Meier şi Staffelbach au arătat ulteriorcă tehnicile de reconstrucţie iterative sunt mult mai rapide, în special când funcţia de combinare este un polinom de grad mic (Meier şi Staffelbach, 1988), iar Mihaljevic şi Golic stabilesc condiţiile în care acest tip de atac rapid de corelaţie converge (Mihajevici şi Golic, 1992).

Atacul de corelaţie rapidă (*fast correlation attack*) se aplică generatoarelor de chei bazate pe LFSR, ca şi atacul de corelaţie, dar sunt mai rapide şi exploatează existenţa unei corelaţii între şirul de chei şi

ieşirea unui LFSR, numit LFSR ţintă, a cărui stare iniţială depinde de anumiţi biţi ai cheii secrete (Meier şi Staffelbach, 1988).

Atacurile de corelaţie rapidă evită examinarea tuturor iniţializărilor posibile ale LFSR-ului ţintă folosind anumite tehnici eficiente de corectare a erorii. Astfel, descoperirea stării iniţiale a LFSR-ului constă în decodarea subşirului de chei relativ la codul FSR-ului (Johansson şi Jonsson, 1999; 2000).

Atacul prin interpolare (*interpolation attack*) este o tehnică de atac asupra cifrurilor simetrice bloc construite din funcţii algebrice simple. Dacă textul criptat este scris ca un polinom, funcţie de elementele textului clar şi gradul polinomului este suficient de mic, atunci un număr limitat de perechi de text clar/criptat sunt suficiente pentru determinarea funcţiei de criptare. Acest lucru permite atacatorului să cripteze sau să decripteze blocuri de date fără a recupera propriu zis cheia de criptare. Atacul a fost introdus în 1997 şi aplicat prima dată pe o variantă a algoritmului SHARK (Jakobsen şi Knudsen, 1997), un predecesor al algoritmului Rijndael. Acest tip de atac poate fi generalizat, astfel că ideea interpolării poate fi aplicată şi în cazul unor polinoame probabilistice (Jakobsen, 1998).

Atacul „divide şi cucereşte” (*divide and conquer attack*). Atacurile din această categorie încearcă diviziunea cheilor în bucăţi mai mici, pentru a face posibilă căutarea exhaustivă. Acest tip de atac este eficient în măsura în care este posibil să determinăm bucăţi separate din chei. Problema constă în validarea sau invalidarea unui segment de cheie, fără a avea informaţii despre restul cheii.

Atacul temporal (*timing attack*). Durata de execuţie a unui echipament hardware de criptare poate furniza informaţii despre parametrii implicaţi astfel încât analiza atentă şi măsurarea timpului de execuţie poate duce, în anumite condiţii, la recuperarea cheilor secrete (Kocher, 1996). Pentru a putea realiza un astfel de atac, atacatorul are nevoie de un set de mesaje împreună cu durata lor de procesare pe echipamentul criptografic. Metodele de măsurare a timpului sunt diverse: monitorizarea activităţii procesorului, măsurarea timpului într-o secvenţă de interogare/răspuns etc. Atacul a fost aplicat algoritmilor RSA (Schindler şi colab., 2001) şi RC5 (Handschuh şi Heys, 1998), dar şi unor protocoale de internet de tipul SSL (Canvel şi colab. 2003).

În continuare sunt expuse două tehnici de criptaniliză - *criptanaliza liniară* şi *criptanaliza diferneţială* - care la momnetul actual sunt unele dintre cele mai răspândite metode de spargere a cifrurilor bloc.

Criptanaliza liniară (*linear cryptanalisis*) este o tehnică introdusă de Matsui si Yamagishi în 1991 (*A new Method for known plain text attack of FEAL cipher*) şcare încearcă să exploateze apariţiile cu probabilitate mare ale expresiilor liniare ce implică biţi de text clar, biţi de text criptat şi biţi ai subcheilor. În acest caz se presupune că atacatorul cunoaşte un set aleator de texte clare, precum şi textele criptate corespunzătoare. Se aplică algoritmilor simetrici, de tip bloc (Matsui, 1993) şi a fost utilizată cu succes în criptanaliza algoritmului DES (Matsui, 1994). Sunt elaborate atacuri pentru cifrurile bloc şi cele de tip flux. Descoperirea criptanalizei liniare a constituit un imbold pentru elaborarea noilor scheme de criptare.

Criptanaliza liniară este o tehnică prin care se urmăreşte construcţia unui sistem de ecuaţii liniare între biţii textului clar, ai textului cifrat şi ai cheii. Rezolvarea acestui sistem de ecuaţii duce la aflarea cheii de cifrare. Sistemul de ecuaţii ce se construieşte poate fi chiar un sistem probabilist, în sensul că o ecuaţie este verificată cu o anumită probabilitate.

Criptanaliza se face în două etape. Prima - construirea relațiilor dintre textul clar, textul cifrat și cheie, care sunt adevărate cu o probabilitate mare. A doua - utilizarea acestor relații, împreună cu perechile cunoscute de text clar şi text cifrat pentru obţinerea biților cheii.

Sensul a algoritmului este de a obține o relație de forma:

$$P_{i1} \oplus P_{i2} \oplus ... \oplus P_{im} \oplus C_{j1} \oplus C_{j2} \oplus ... \oplus C_{jn} = P_{l1} \oplus K_{l2} \oplus ... \oplus K_{lq} \quad (*),$$

Unde P_n, C_n, K_n – sunt biţii de ordin n respecriv ai textului clar, textului cifrat şi ai cheii.

Aceste relaţii sunt numite aproximări liniare. Pentru biții aleşi aleator ai textului clar, textului cifrat și ai chei probabilitatea p ca această relaţie să fie justă este aproximativ egală 0,5. Pentru spargerea algoritmului alegem astfel de relaţii care au probailitatea respectivă semnificativ diferită de 0,5

Mai întâi criptanalistul găsește o relaţie oarecare pe o singură rundă pe care încearcă să o extindă asupra întregului algoritm. Sunt elaboraţi algoritmi pentru găsirea relaţiilor utile de acest fel. Doi algoritmi de acest fel au fost descrişi de către Mitsuru Matsui, alții mai târziu .

În cifrurile bloc analiză se concentrează în principal pe S-boxe, deoarece acestea sunt parte a non-liniarităţii cifrului. Cea mai eficientă

relaţie pe o rundă pentru algoritmul DES utilizează proprietatea boxei S_5. Al doilea bit de intrare al boxei biţi este rezultatul XOR dintre toţi biţii de ieşire cu probabilitatea de 3/16 (cu o deplasare de 5/16 de la 1/2). Iar pentru DES-ul complet este cunoscută o relaţie care se îndeplineşte cu probabilitatea $1/2 + 2^{-24}$.

Criptanaliza liniară posedă o caracteristică foarte utilă - în anumite condiţii, este posibil să se reducăraportul dintre (*) la o ecuaţie deforma:

$$C_{j1} \oplus C_{j2} \oplus ... \oplus C_{jn} = P_{l1} \oplus K_{l2} \oplus ... \oplus K_{lq}.$$

Aici nu sunt biţii textului clar, deci este posibil să se construiască atacul numai cu textul cifrat. Acest atac este cel mai practic.

Experimentele efctuate de Mitsuru Matsui cu atacul cu text clar (calculele au fost efectuate pe HP9750 66MHz) au dat următoarele rezultate:

- *DES* cu 8 runde se sparge cu 2^{21} texte în clar cunoscute. Matsui a avut nevoie de 40 secunde.
- *DES* cu 12 runde se sparge cu 2^{33} texte în clar cunoscute. Au fost necesare 50 de ore.
- Pentru un *DES* 16 runde a fost necesare 2^{47} texte în clar cunoscute. Acest atac nu este, de obicei, practic. Cu toate acestea, metoda este mai rapidă decât căutarea exhaustivă pentru cheia de 56 biţi.

Tehnologiile moderne ale calculatoarelor sunt capabile să spargă cifrul dat mult mai rapid.

Deseori metoda de criptanaliza liniară este utilizată împreună cu *atacul forţei brute* un “brute force" - după ce un anumit număr de biţi ai cheii au fost găsiţi de criptanaliza liniara se face o căutare exhaustivă pentru toate valorile posibile ale celorlalte biţi. Pentru a sparge DES în acest fel sunt necesare 2^{43} texte în calr.

La momentul actual pentru orice cifru nou este necesar de demonstrat că el este rezistent la criptanaliza liniară.

Criptanaliza diferenţială (*diferential cryptanalysis*) este o tehnică introdusa de Biham şi Schamir prima dată în 1991 şi concretizată pentru DES în „Diferential Cryptanalysis of the Data Encryption Standard” publicat în 1993. Tehnica face parte din categoria atacurilor cu text clar ales. În general dandu-se perechea (D, D'), numită caracteristică, tehnica constă în generarea de texte clare (P_1, P_2) cu $D = P_1 - P_2$ astfel incat $D' = C_1 - C_2$ şi aflarea unei părţi a cheii, restul fiind cautata exhaustiv.

Operatia „–” este operaţia de grup care, spre exemplu în cazul cifrurilor bloc, constă în adunarea cheii de runda.

Criptanaliza diferenţială exploatează apariţiile cu mare probabilitate a diferenţelor din textele clare, precum şi a diferenţelor apărute în ultimul ciclu al criptorului în care atacatorul poate selecta intrările şi examina ieşirile în încercarea de a deduce cheia.

Sunt cunoscute cîteva modele de criptanaliză diferentială:

1. Criptanaliza diferenţială clasica;
2. Criptanaliza diferenţialelor trunchiate;
3. Criptanaliza diferenţialelor imposibile;
4. Criptanaliza diferenţialelor de ordin superior;
5. Criptanaliza diferenţial-liniara;

Criptanaliza diferentiala clasică permite pe baza perechilor de text clar/text cifrat ;i a diferentialelor acestora, aflarea ultimei chei de runda. Pentru diferentiala (α, β) în runda (r – 1) se realizează paşii (Figura 15.1, **URNG** - Uniformly Random Number Generator):

1. Selectează aleatoriu textul clar $X(1)$ şi calculează $X^*(1)$ astfel încat $\Delta X(1) = X(1) + X^*(1) = \alpha$. Se cifreaza $X(1)$ şi $X^*(1)$.
2. Presupunând că $\Delta Y(r–1) = \beta$, căutăm toate valorile subcheii $z(r)$ consistente cu $\Delta Y(r\text{-}1)$ şi textele cifrate $Y(r)$ şi $Y^*(r)$. Se incrementează frecvenţa de apariţie a acestor subchei.
3. Se repetă (1) şi (2) până când o anumită cheie are o frecvenţă semnificativ mai mare decât a celorlalte. Aceasta subcheie $z(r)$ este cheia ultimei runde.

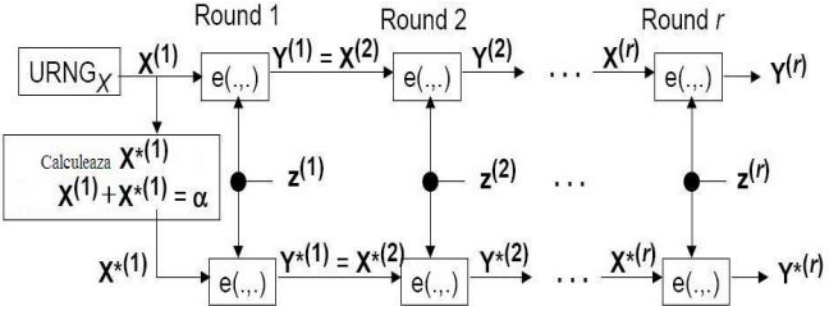


Figura 15.1. Schema criptanalizei diferenţiale

Criptanaliza diferenţialelor trunchiate (*truncated differential cryptanalysis*) este o generalizare a criptanalizei diferențiale. Lars Knudsen a dezvoltat tehnica în 1994. În timp ce criptanaliza diferentiala clasică analizează diferența totală între două texte, varianta trunchiată consideră diferențe care sunt determinate doar parțial. Prin urmare, acest atac poate prezice doar unele valori ale biților şi nu a întregului bloc. Aceasta tehnica a fost aplicata la cifrurile bloc SAFER, IDEA, Skipjack, E2, Twofish, Camellia, CRYPTON, şi chiar la cifrul flux Salsa20.

Criptanaliza diferenţialelor imposibile (*impossible differential cryptanalysis*) este o formă de criptanaliza diferențială pentru cifrurilor bloc. În timp ce criptanaliza diferentială clasică urmăreşte diferențe care se propagă prin cifru cu o probabilitate mai mare decât se aştepta, criptanaliza diferențialelor imposibile exploatează diferențele care sunt imposibile (cu probabilitatea 0) la o stare intermediară a algoritmului de criptare.

Lars Knudsen pare a fi primul care a folosit o forma a acestui atac, într-o lucrare în 1998, unde a prezentat DEAL – candidatul său la AES. Prima prezentare care a atras atenția comunității criptografice a fost mai târziu în același an, la conferinţa CRYPTO'98, în care Eli Biham, Alex Biryukov, şi Adi Shamir a introdus definiţia „diferențiale imposibile" şi au folosit tehnica dată pentru a sparge cifrul *IDEA* şi *Skipjackc* cu un număr redus de runde. Tehnica a fost aplicată mai tîrziu şi la multe alte cifruri: Khufu şi Khafre, E2, variante ale Serpent, MARS, Twofish, Rijndael, CRYPTON, Zodiac, Hierocrypt-3, TEA, XTEA, Mini-AES, ARIA, Camellia, and SHACAL–2.

Biham, Biryukov şi Shamir a prezentat, de asemenea, o metodă specială relativ eficientă pentru a găsi diferențiale imposibile care au numit-o atac *miss-in-the-middle*. Aceasta constă în a găsi „două evenimente de probabilitate unu, ale căror condiții nu pot fi îndeplinite concomitent".

Criptanaliza diferenţialelor de ordin superior este o generalizare a criptanalizei diferențiale şi este de fapt aplicarea recursiva a diferentialei. Dezvoltată în 1994 de catre Lars Knudsen, tehnica a fost aplicată asupra unui şir de cifruri. În timp ce criptanaliza diferentială clasică analizează diferențele dintre cele două texte, varianta de ordin superior analizează diferențele dintre diferențe, etc. În unele cazuri această tehnică s-a dovedit a fi mai puternică decât un atac de ordinul întâi (vezi cifrul KN).

Criptanaliza diferenţial-liniara este o metodă hibridă ce utilizează criptanaliza diferenţială şi cea liniară.

Atacul de tip „bumerang” (*boomerang attack*) foloseşte flexibilitatea criptografiei diferenţiale și permite utilizarea a două caracteristici necorelate pentru a ataca cele două jumătăţi ale unui cifru bloc. Metoda măreşte potenţialul criptanalizei diferenţiale prin folosirea unor caracteristici ce nu se propagă prin întreg algoritmul criptografic. Rezultatele se produc doar în prezenţa ambelor caracteristici, întrucât metoda nu poate lucra independent, pentru fiecare caracteristică.

Atacul de consistenţă liniară (*linear consistency attack*) aplică o tehnica de tipul divide și cucereşte pentru o secvenţe de text clar cunoscute. A fost introdus în 1989 și aplicat algoritmilor de tip şir, asupra generatoarelor de şiruri de chei (Zeng și colab., 1989), dar şi asupra algoritmului E0 folosit în Bluetooth (Fluhrer și Lucks, 2001).

În final trebuie de subliniat că la elaborarea sistemelor de criptare trebuie să fie luate în consideraţie performanţele criptanalizei pentru a diminua riscurile posibile.

Bibliografie

1. Ben-Aroya I., Biham E., *Differential Cryptanalysis of Lucifer*, Journal of Cryptology 9(1), pp. 21–34, 1996
2. Biham E. and Biryukov A., *An Improvement of Davies' Attack on DES*, J. Cryptology 10(3): 195–206 (1997)
3. Biham E. and Shamir A., *Differential Cryptanalysis of the Full 16-Round DES*, Advances in Cryptology, Proceedings of CRYPTO '92, 1992, pp. 487–496
4. Daemen J., Rijmen V., *The Design of Rijndael: AES - The Advanced Encryption Standard*, Springer-Verlag, 2002
5. *Data Encryption Standard*, National Bureau of Standards, Federal Information Standard 46, USA; 1977
6. Devours C., Kahn D., Kruh L., Mellen G., Winkel B., *Cryptology: Machines, history and Methods*, Artech House, 1989
7. Devours C., Kruh L., *Machine Cryptography and Modern Cryptanalysis*, Artech House, 1985
8. Diffie W. and Hellman M. E., *New Directions in Cryptography*. IEEE Transactions on Information Theory, vol. IT-22, Nov. 1976, pp. 644–654
9. El Gamal T., *A Public Key Cryptosystem and Signature Scheme Based on Discrete Logarithms*, IEEE Transactions in Information Theory 31 (4), 1985, pp. 469–472
10. Ferguson N. and Schneir B., *Practical Cryptography*, Wiley N. Y. (2003)
11. FIPS PUB 197, The official AES standard
12. FIPS PUB 46-3, DES, (Federal Information Processing Standards Publication). NIST, 1999.
13. Friedman W. F., *Military Cryptanalysis – Part I, Monoalphabetic Substitution Systems*, United States Government Printing Office, Washington, 1939
14. Friedman W. F., *Military Cryptanalysis – Part II*: *Simpler Varieties of Polyalphabetic Substitution Systems*, United States Government Printing Office, Washington, 1938
15. Kahn D., *The Codebreakers - The Story of Secret Writing*, abridget ed. New York, NY: Signet, 1973

16. Katz J. & Lindell Y., *Introduction to Modern Cryptography*, Chapman & Hall/CRC (2008)
17. Konheim Alan G., *Computer Security and Cryptography*, John Wiley & Sons, Inc., 2007
18. Menezes A., Van Oorschot P. C., Vanstone S. A., *Handbook of applied cryptography*, CRC Press, 1997
19. Mogollon M., *Cryptography and Security Services: Mechanisms and Applications*, New York, Cybertech Publishing, 2008
20. National Bureau of Standards, *Data Encryption Standard*, FIPS PUB 46 (Jan. 1977)
21. National Bureau of Standards, *DES Modes of Operation*, FIPS PUB 81 (Dec. 1980)
22. National Institute of Standards and Technology, *Advanced Encryption Standard* (*AES*), FIPS PUB 197 (Nov. 2001)
23. NIST, Modes of operations for symmetric block ciphers (available at: http://csrc.nist.gov/CryptoToolkit/modes/)
24. Rivest R. L., Shamir A., Adleman L., *A method for obtaining digital signatures and public key cryptosystems*, Comm ACM 21, 1978, pp. 120–126
25. Rivest R. L., *The RC5 encryption algorithm*, In: Fast software encryption–Leuven 1994. LNCS, vol. 1008. Berlin: Springer; 1995, pp. 86–96
26. Rueppel R. A., *Stream ciphers*, In: G. J. Simmons, Editor, Contemporary Cryptology–The Science of Information Integrity, IEEE Press, New York, 1992, pp. 65–134
27. Salomaa A., *Criptografie cu chei publice*, Ed. Militară, 1994
28. Schneier B., *Applied Cryptography*, Second Edition. John Wiley & Sons, 1996
29. Schneier B., *The Blowfish Encryption Algorithm*. *One Year Later*, Dr. Dobb's Journal, 20(9), p. 137, September 1995.
30. Scripcariu Luminiţa, *Bazele reţelelor de calculatoare*", Ed. Cermi Iaşi, 2005.
31. Shannon C. E., *A Mathematical Theory of Communication*, Bell System Technical Journal. v. 27, n. 4, 1948, pp. 379-426
32. Shannon C. E., *Communication Theory of Secrecy Systems*, Bell System Technical Journal. v. 28, n. 4, 1949, pp. 656-715

33. Shannon C. E., *Predication and Entropy in Printed English*, Bell System Technical Journal. v. 30, n. 1, 1951, pp. 50-64
34. Smith, Laurence D. „*Substitution Ciphers”*. *Cryptography the Science of Secret Writing*, Dover Publications. pp. 81. 1943
35. Sorkin A., *LUCIFER: a cryptographic algorithm*, Cryptologia, 8(1), 22-35, 1984
36. Vernam G., *Cipher Printing Telegraph Systems For Secret Wire and Radio Telegraphic Communications*, Journal of the IEEE, Vol 55, pp. 109-115 (1926)
37. Аграновский А. В., Хади Р. А., *Практическая криптография, алгоритмы и их программирование*, Москва, 2009
38. Гутер Р.С., Полунов Ю.Л., *Чарльз Бэббедж* (1792-1871), Москва, Знание, 1973
39. Мао В., *Современная криптография* (*теория и практика*), М.: Вильямс, 2005
40. Шнайер Б., *Секреты и ложь. Безопасность данных в цифровом мире*, Москва, 2003
41. Щербаков А.Ю. Современная компьютерная безопасность. Теоретические основы. Практические аспекты. — М.: Книжный мир, 2009. — 352 с.