2

Coduri detectoare/corectoare de erori. Criptarea informaţiei

1. Prezentare teoretică

În cadrul acestei lucrări de laborator se vor prezenta algotimii CRC şi Reed-Solomon folosiţi la detectarea şi corectarea erorilor care pot apărea într-o transmisie de date. Algoritmii RSA şi IDEA prezentaţi sunt uzual folosiţi pentru criptarea informaţiei şi se bazează pe chei publice. Implementările hardware ale altor algoritmi de criptare, care se bazează pe metode tradiţionale (de exemplu algoritmul de criptare DES), pot fi studiate la http://www.csit-sun.pub.ro/resources.

Sume de control

Scopul unei tehnici de detecţie a erorilor este acela de a pune la dispoziţia receptorului unui mesaj, transmis printr-un canal cu zgomote (pasibil de introducere de erori), o metodă de a determina dacă mesajul a fost corupt sau nu. Pentru a face posibil acest lucru, emiţătorul construieşte o valoare numită sumă de control care este o funcţie de mesaj şi o anexează acestuia. Receptorul poate să folosească aceeaşi funcţie pentru a calcula suma de control pentru mesajul primit, iar apoi să o compare cu suma de control anexată (concatenată mesajului) pentru a vedea dacă mesajul a fost receptat corect.

Exemplu Să se aleagă o funcţie care are ca rezultat (sumă de control) suma octeţilor din mesaj modulo 256:

$$f(x) = \sum(octeti \quad mesaj) \bmod 256 \qquad (1)$$

Considerând toate valorile în zecimal, se obţine:

mesaj	:	7 24 3
mesaj cu suma de control	:	7 24 3 34
mesaj după transmisie	:	7 28 3 38

Al doilea octet al mesajului a suferit o modificare în timpul transmisiei, de la 24 la 28. Cu toate acestea, receptorul poate determina prezenţa unei erori comparând suma de control transmisă (34) cu cea calculată (38 = 7 + 28 + 3).

Dacă însăşi suma de control este coruptă, un mesaj transmis corect poate fi (incorect) interpretat drept unul eronat. Acesta nu este însă un eşec periculos. Un eşec periculos are loc atunci când atât mesajul cât şi suma de control se modifică astfel încât rezultă într-o transmisie consistentă intern (interpretată ca neavând erori).

Din păcate, această posibilitate nu poate fi evitată şi cel mai bun lucru care se poate realiza este de a minimiza probabilitatea ei de apariţie prin creşterea cantităţii de informaţie din suma de control (de exemplu, lărgind dimensiunea ei la doi octeţi în loc de unul).

Coduri CRC

Ideea de bază pentru algoritmii CRC este de a trata mesajul drept un număr reprezentat în binar, de a-l împărţi la un alt număr binar fixat şi de a considera restul drept sumă de control. La primirea mesajului, receptorul poate efectua aceeaşi împărţire şi poate compara restul cu suma de control primită (restul transmis).

Exemplu Considerând că mesajul care trebuie transmis este alcătuit din 2 octeţi (6, 23), el este reprezentat în baza 16 ca numărul 0617 şi în baza 2 ca 0000_0110_0001_0111. Se presupune folosirea unei sume de control de l octet şi a unui împărţitor constant 1001. Atunci suma de control va fi restul împărţirii 0000_0110-0001_0111 : 1001 = ... 0000010101101, rest 0010. Mesajul transmis de fapt va fi: 06172, unde 0617 este mesajul iniţial (informaţia utilă), iar 2 este suma de control (restul).

Aritmetica binară fără transport

Toate calculele executate în cadrul algoritmilor CRC sunt realizate în binar, fără transport. Deseori se foloseşte denumirea de aritmetică polinomială, dar în continuare se va folosi denumirea de aritmetică CRC deoarece la implementarea cu polinoame s-a renunţat.

Adunarea a două numere în aritmetica CRC, aşa cum se poate observa în figura 1, este asemănătoare cu adunarea binară obişnuită, însă nu există transport. Aceasta înseamnă că fiecare pereche de biţi corespondenţi determină bitul corespondent din rezultat, fără nici o referinţă la alt bit din altă poziţie (aşa cum se poate observa din exemplul prezentat în figura 2 a).).

Definiţia operaţiei de scădere este identică cu operaţia de adunare şi poate fi observată în figura 1, iar un exemplu este prezentat în figura 2 b).

Se poate concluziona că atât adunarea cât şi scăderea în aritmetica CRC sunt echivalente cu operaţia SAU EXCLUSIV (XOR), iar operaţia XOR este propria sa inversă. Acest fapt reduce operaţiile primului nivel de putere (adunare, scădere) la una singură, care este propria sa inversă (o proprietate foarte convenabilă a acestei aritmetici).

a	b	a + b
0	0	0
0	1	1
1	0	1
1	1	0

a	b	a - b
0	0	0
0	1	1
1	0	1
1	1	0

Figura 1: Definirea operaţiilor de adunare/scădere.

Pe baza adunării, se poate defini şi înmulţirea, care se realizează natural, fiind suma dintre primul număr deplasat corespunzător şi cel de-al doilea număr (se foloseşte adunarea CRC). Un exemplu pentru această operaţie este prezentat în figura 2 c).

Pentru realizarea operaţiei de împărţire, este nevoie să se cunoască când *un număr este cuprins în altul.* De aceea, se va considera următoarea definiţie: *X* este mai mare decât sau egal cu *Y* dacă poziţia celui mai semnificativ bit l al lui X este mai mare sau aceeaşi cu poziţia celui mai semnificativ bit l al lui Y. Un exemplu complet este prezentat în figura 2 d).

Transmisia - recepţia datelor folosind CRC

Aşa cum s-a arătat până acum, calculul CRC este de fapt o simplă împărţire. Pentru realizarea unui calcul CRC este nevoie de un divizor, denumit în limbaj matematic *polinom generator.* Lungimea polinomului uzuală este de 16 sau 32 de biţi, CRC-16, CRC-32, şi aceste dimensiuni sunt folosite în calculatoarele digitale moderne. *Lungimea unui polinom - W-* este de fapt poziţia celui mai semnificativ bit l (lungimea polinomului 10011 este 4).

La transmiţător, înainte de calculul CRC, se adaugă *W* biţi cu valoarea 0 la sfârşitul mesajului care va fi împărţit folosind aritmetica CRC la polinom, astfel încât toţi biţii mesajului să participe la calculul CRC. Un exemplu este prezentat în figura 2 d). Împărţirea produce un cât, care nu va fi ignorat şi un rest, care este suma de control

calculată (CRC-ul). În mod uzual CRC-ul este apoi adăugat mesajului, iar rezultatul este trimis către receptor, în acest caz se transmite 11010110111110.

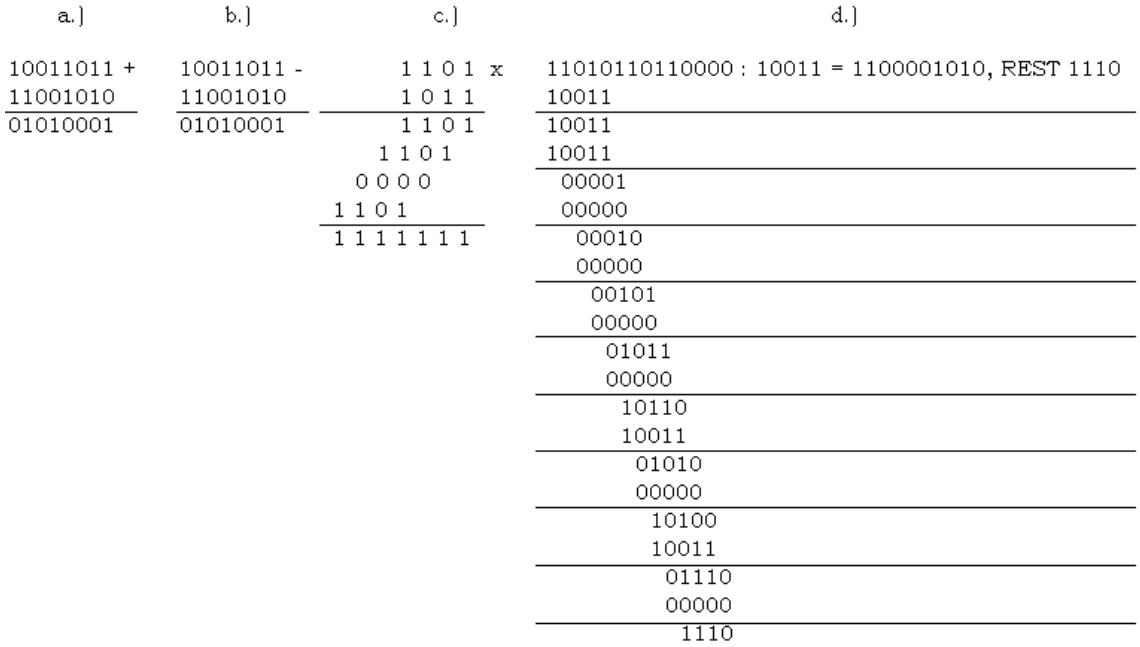


Figura 2: Exemplificarea operaţiilor binare fără transport

Receptorul calculează suma de control pentru întreg mesajul primit (fără adăugare de zerouri) şi compară restul cu 0. Realizarea acestei operaţii este motivată de faptul că mesajul transmis *T* este multiplu de polinomul folosit drept divizor.

Implementarea directă

CRC-ul se poate calcula utilizând noţiunile teoretice prezentate până acum. Algoritmul, implementarea Verilog precum şi rezultatele simulării pot fi observate în figura 3.

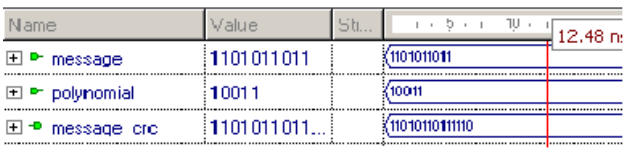
Implementarea bazată pe tabelă

Acest algoritm este o variantă îmbunătăţită a algoritmului anterior, el fiind foarte eficient deoarece implică doar o deplasare, o operaţie SAU, o operaţie SAU EXCLUSIV şi un acces la memorie pentru fiecare octet. Algoritmul precum şi rezultatele implementării sale în Verilog sunt prezentate în figura 4.

```
module crc_simple(message, polynomial, message_crc);
        /* Parametrii: */
        parameter crc_width = 4;  /* lungimea CRC-ului; */
        parameter msg_width = 10; /* lungimea mesajului; */
        /*Intrari, iesiri si resurse fizice interne*/
        /* -> masajul initial: */
        input  [msg_width - 1 : 0] message;
        wire   [msg_width - 1 : 0] message;
        /* -> polinomul generator: */
        input  [crc_width : 0] polynomial;
        wire   [crc_width : 0] polynomial;
        /* -> mesajul cu CRC-ul adaugat: */
        output [msg_width + crc_width - 1 : 0] message_crc;
        reg    [msg_width + crc_width - 1 : 0] message_crc;
        reg    [crc_width - 1 : 0] crc; /* the CRC register; */
        reg    msb_crc;/*cel mai semnificativ bit al registrului CRC; */
        /* resurse logice: */
        integer count; /* numarator; */
        /* Calculeaza CRC-ul si atasare: */
        always @(message | polynomial) begin
                /* Initializare: */
                message_crc = message;
                message_crc = message_crc << crc_width;
                crc = 'b0;
                /* Calculeaza CRC-ul: */
                for (count = msg_width + crc_width - 1; count >= 0; count = count - 1)
                 begin
                        msb_crc = crc[crc_width - 1];
                        crc = crc << 1;
                        crc[0] = message_crc[count];
                        if (msb_crc == 1) begin
                                crc = crc ^ polynomial;
                        end
                end
                /* Adauga CRC-ul mesajului: */
                message_crc[crc_width - 1 : 0] = crc;
        end
endmodule
```

Algoritm:

1. încarcă registrul cu biți 0
2. adaugă la sfârşitul mesajului W biți 0.
3. cât timp (mesajul mai are biți) deplasează registrul stânga cu un bit, introducând următorul bit din mesaj în poziția 0
4. dacă (un bit 1 a fost scos din registru) registru = registru XOR polinom
5. registrul conține restul



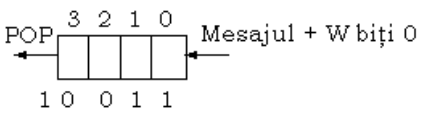


Figura 3: Implementare directă

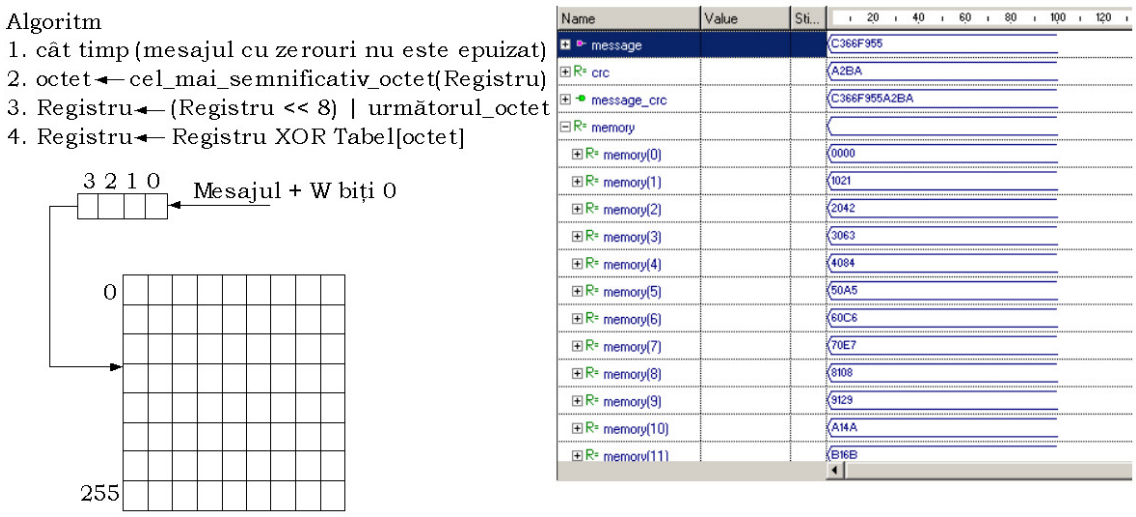


Figura 4: Implementarea bazată pe tabelă

Coduri Reed-Solomon

Codurile Reed-Solomon (RS) sunt coduri corectoare de erori în bloc inventate în 1960 de Irving Reed şi Gustave Solomon. Aceste coduri au început să fie utilizate începând cu 1990, atunci când progresele tehnologice au făcut posibilă trimiterea datelor în cantităţi mari si la viteze ridicate. Actualmente aceste coduri sunt utilizate într-o gamă largă de echipamente electronice cum sunt:

- dispozitivele pentru stocarea datelor (CD, DVD, hard-disk);
- telefoanele mobile;
- echipamentele folosite în comunicaţiile prin satelit;
- televiziunea digitală;
- modemurile de mare viteză (ADSL, xDSL).

Realizarea unei transmisii folosind codurile RS presupune ca, codificatorul RS să preia un bloc de date şi să adauge o informaţie suplimentară caracteristică. Una dintre caracteristicile importante ale codului RS constă în faptul că acest cod va codifica grupuri de simboluri de date.

Decodificatorul RS procesează fiecare bloc şi încearcă să corecteze erorile apărute şi să recupereze datele trimise original.

Un cod RS este specificat ca *RS(n, k)* cu simboluri de *s* biţi. Această descriere semnifică faptul că, codificatorul preia *k* simboluri de paritate astfel încât să rezulte un cuvânt de cod de *n* simboli. Sunt *n - k* simboluri de paritate, de câte *s* biţi fiecare. Un decodificator RS poate corecta până la *t* simboluri ce conţin erori, cu $2t = n - k$.

Un cod RS este obţinut împărţind mesajul original în blocuri de lungime fixă. Fiecare bloc este apoi împărţit în simboluri de *m* biţi. Fiecare simbol are lungime fixă (între 3 si 8 biţi). Natura liniară a acestui cod asigură faptul că fiecare cuvânt de *m* biţi este valid pentru codificare astfel încât se pot transmite date binare sau text.

Exemplu Un cod des folosit este RS(255, 233) cu simboluri de 8 biţi. Fiecare cuvânt de cod conţine 255 de simboluri din care 233 sunt de date şi 22 sunt de paritate. Pentru acest cod se pot stabili următoarele relaţii: $n = 255$, k — 233, $s = 8$, $t = 16$.

Codurile RS pot fi scurtate dacă la codificator se fac anumiţi biţi zero, nu se transmit dar sunt adăugaţi la decodificator. Spre exemplu, codul RS(255, 233) poate fi scurtat la (200, 168). Operaţiile realizate de codificator sunt următoarele:

- se preia un bloc de 168 de biţi de date;
- se adaugă virtual 55 de biţi de zero creând astfel un cod (255, 233);
- se transmit doar 168 biţi de date şi 32 biţi de paritate.

Un decodificator RS poate corecta un număr de *t* erori şi până la *2t* ştersături. La decodificarea unui cuvânt RS pot apărea următoarele variante:

- dacă $2s + r < 2t$ atunci codul original transmis poate fi corectat în întregime;
- decodificatorul indică faptul că nu poate reface codul original;
- decodificatorul va genera un cuvânt decodat cu erori şi nu va fi semnalat acest lucru.

Arhitectura decodorului poate fi urmărită în figura 5.

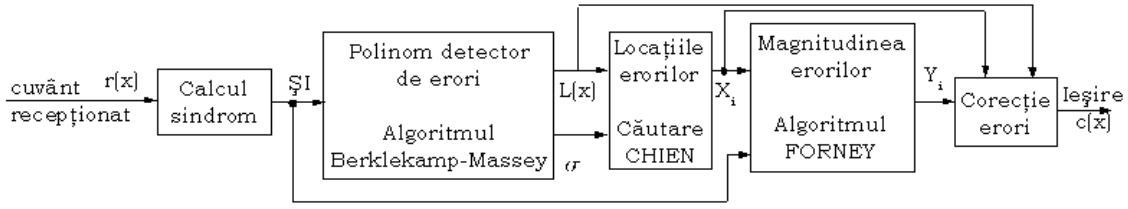


Figura 5: Arhitectura unui decodificator RS.

Algoritmul de criptare RSA

Algoritmul RSA este un sistem criptografic ce utilizează chei publice şi a fost creat de un grup de cercetători de la MIT (Massachusetts Institute of Technology) cu scopul de a asigura securitatea datelor schimbate prin intermediul Internet-ului.

Metodele tradiţionale de criptare (spre exemplu algoritmul DES - implementările hardware şi JAVA precum şi simulările acestor implementări pot fi vizualizate la http://www.csit-sun.pub.ro) folosesc un număr de $\frac{n \cdot (n-1)}{2}$ chei, în timp ce algoritmii bazaţi pe chei publice utilizează un număr de cel mult *n* chei publice.

O altă deosebire constă în faptul că în sistemele tradiţionale de criptare, cheia de criptare trebuie ţinută secretă deoarece ea trebuie utilizată în cadrul procesului de decriptare. În cazul criptării cu chei publice, cheia de criptare/decriptare nu mai este trimisă receptorului, deci canalul de comunicaţie dintre transmiţător şi receptor poate să nu fie securizat.

Utilizarea algoritmului RSA implică crearea a două chei de către transmiţător: *una publică* şi una *privată.* Cheia publică este trimisă oricărui destinatar la care trebuie trimis mesajul criptat. Cheia privată sau secretă este utilizată pentru decriptarea mesajului criptat cu ajutorul cheii publice.

Modalitatea de realizare a unei comunicaţii criptate cu ajutorul algoritmului RSA este prezentată în figura 6.

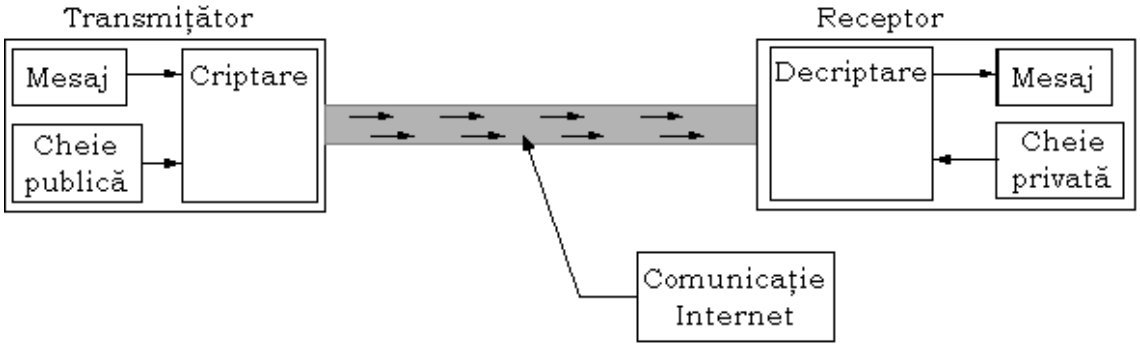


Figura 6: Arhitectura unui decodor RS.

Transmisia folosind algoritmul RSA necesită parcurgerea a două etape importante:

1. Generarea cheilor - se generează două chei una publică şi una privată. Pentru aceasta trebuie parcurşi următorii paşi:

1. se aleg două numere prime p şi q cu aceeaşi magnitudine (lungime) şi se generează numărul $n = p \cdot q$;

2. se determină $\Phi = (p-1) \cdot (q-1)$;

3. se alege e ca fiind un număr prim în raport cu Φ, deci cel mai mare divizor comun (notat gcd(e, Φ)) al celor două numere trebuie să fie 1. În implementările practice valoarea lui e este aleasă ca fiind un număr prim Fermat (3, 5, 17, 65537,...);

4. se determină valoarea d care reprezintă inversiunea modulară a lui e şi Φ:

$$d = rest\left(e - \frac{1}{\Phi}\right)$$

Cheia publică este alcătuită din perechea *(n, e),* cât timp cheia privată este formată din perechea *(n, d).* Implementarea hardware a celui mai mare divizor comun se realizează cu ajutorul algoritmului lui Euclid.

```
Algoritm EuclidExtins(a, b)
     if b = 0 then
          return (a, 1, 0)
     else
          (d', x', y') = EuclidExtins(b, rest(a/b))
     return (d', y', x' - a/b · y')
```

2. Transmisia informaţiei - În cadrul acestei etape, atât transmiţătorul cât şi receptorul trebuie să execute câteva operaţii distincte. Transmiţătorul realizează următoarele operaţii:
   a. obţine cheia publică *(n, e)* de la receptor;
   b. converteşte mesajul într-o mulţime de întregi pozitivi;
   c. calculează textul criptat conform relaţiei: $c = m^e \ mod$ n;
   d. transmite mesajul c la receptor.

   Receptorul realizează următoarele operaţii:
   a. utilizează cheia privată *(n, d)* pentru a calcula $m = c^d \ mod \ n$;
   b. extrage textul din colecţia de numere întregi *m.*

Algoritmul de criptarea IDEA

IDEA este un algoritm bazat pe chei publice care criptează blocuri de câte 64 de biţi folosind o cheie de criptare de lungime 128 de biţi. Criptarea şi decriptarea presupun utilizarea aceluiaşi algoritm. Implementarea acestui algoritm impune utilizarea a trei operaţii: *XOR, adunarea modulo 65536* şi *înmulţirea modulo 65537* care operează pe sub-blocuri de dimensiune 16 biţi.

Funcţionarea algoritmului constă în parcurgerea a opt paşi. Blocul de date de dimensiune 64 de biţi este împărţit în 4 părţi X_0, X_1, X_2 şi X_3, fiecare parte având dimensiunea de 16 biţi. În fiecare pas, între cele 4 sub-blocuri se realizează o operaţie *XOR,* de adunare sau de înmulţire, împreună cu 6 subchei de dimensiune 16 biţi fiecare.

Între paşii 2 şi 3, sub-blocurile sunt interschimbate, iar în final cele 4 sub-blocuri sunt combinate împreună cu 4 subchei pentru a forma ieşirea. În cadrul fiecărui pas al algoritmului se execută următoarea succesiune de operaţii:

- se înmulţeşte X_0 cu prima subcheie;
- se adună X_1 la a doua subcheie;
- se adună X_2 la a treia subcheie;
- se înmulţeşte X_3 cu a patra subcheie;
- XOR între rezultatele paşilor l şi 3;
- XOR între rezultatele paşilor 2 şi 4;
- se înmulţeşte rezultatul pasului 5 cu subcheia numărul 5;
- se adună rezultatele obţinute în cadrul paşilor 6 şi 7;
- se înmulţeşte rezultatul de la pasul 8 cu subcheia numărul 6;
- se adună rezultatele obţinute la paşii 7 şi 9;
- XOR între rezultatele paşilor l şi 9;
- XOR între rezultatele paşilor 3 şi 9;
- XOR între rezultatele paşilor 2 şi 10;

- XOR între rezultatele paşilor 4 şi 10.

Cele patru rezultate sunt sub-blocurile obţinute în urma paşilor 11, 12, 13 si 14. Se inter-schimbă cele două sub-blocuri din mijloc şi astfel se obţine intrarea pentru următorul pas. Excepţie face ultimul pas în care nu se mai execută interschimbarea celor două sub-blocuri din mijloc. După pasul opt se execută următoarea secvenţă de operaţii pentru a determina rezultatul final:

- se înmulţeşte X_0 cu prima subcheie;
- se adună X_1 la a doua subcheie;
- se adună X_2 la a treia subcheie;
- se înmulţeşte X_3 cu a patra subcheie.

În final cele patru sub-blocuri se vor concatena pentru a forma blocul criptat de lungime 64 de biţi.

Algoritmul utilizează 52 de subchei: 6 subchei pentru fiecare pas şi 4 subchei pentru pasul final. Generarea subcheilor porneşte de la cheia de lungime 128 de biţi care se împarte în opt subchei. Acestea reprezintă primele opt subchei utilizate în algoritm. La pasul următor cheia este deplasată la stânga 25 de poziţii şi apoi împărţită în opt părţi. Acest proces de generare a subcheilor este continuat până se generează toate cele 52 de subchei necesare funcţionării algoritmului.

Schema generală a algoritmului de criptare ***IDEA*** este prezentată în figura 7.

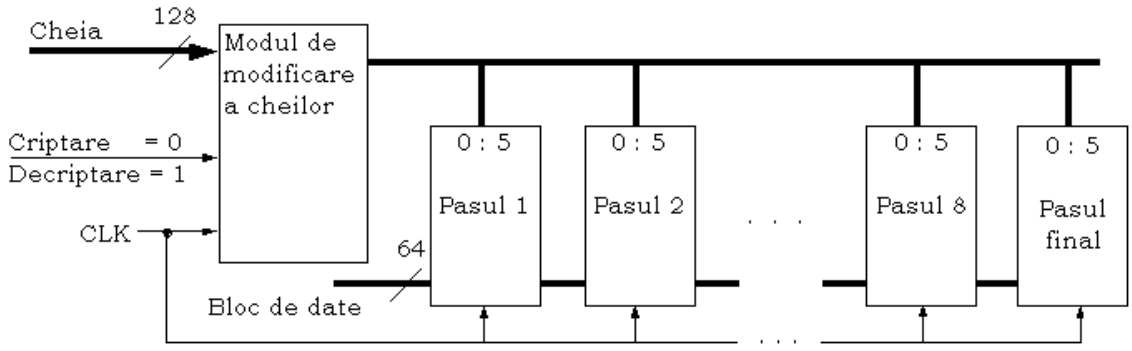


Figura 7: Schema generală a algoritmului de criptare cu chei publice IDEA.

2. Desfăşurarea lucrării

Se va proiecta în Verilog utilizând Xilinx WebPACK ISE 10.1 şi se va simula un circuit, care implementează algoritmul IDEA. Se va folosi schema generală prezentată în figura 7.

3. Probleme propuse

1. Să se proiecteze în Verilog utilizând Xilinx WebPACK ISE 10.1 şi să se simuleze un circuit, care implementează algoritmul CRC bazat pe tabelă.

2. Să se proiecteze în Verilog utilizând Xilinx WebPACK ISE 10.1 şi să se simuleze un circuit, care implementează algoritmul de criptare RSA.

Indicaţii

- Este bine să se calculeze o tabelă de conversie pentru fiecare dintre cele 256 valori de intrare posibile. Pentru a cripta mesajul va fi necesar doar accesul la o memorie locală care memorează tabela determinată. La decriptare se va utiliza acelaşi artificiu.

- Pentru a putea implementa în hardware expresia $m^e \ mod \ n$ se va utiliza următorul algoritm:

```
res = m;
for (i = 2; i<= e; i = i + 1) begin
  res = res * m;
  if (res > m) begin
        res = res % n;
  end
end
cypher(m, n, e) = res;
```