a cura di Francesco Romani

Crittografia: l'algoritmo DES

Questo mese parliamo un po' di crittografia; in particolare presentiamo uno dei principali algoritmi di cifratura messo a punto dalla cosiddetta crittografia moderna. Tale algoritmo conosciuto, con il nome *Data Encryption Standard*, o più brevemente DES, rappresenta attualmente lo standard della crittografia a chiave segreta. L'implementazione del DES in *Mathematica* riportata nel seguito sicuramente non è efficiente ma si presta bene a sperimentazioni didattiche.

di Antonio Cisternino e Barbara Leporini

Cifrari a sostituzione ed a trasposizione

Il termine crittografia deriva da due termini di origine greca: *kryptos* (nascosto) e *graphia* (scrittura), da cui segue l'immediato significato di scrittura segreta, ovvero che può essere decifrata solo da chi sia a conoscenza del codice. Si indica con **cifratura** il procedimento con cui si passa dal testo in chiaro al crittogramma, e con **decifrazione** il procedimento inverso. Quando si parla di **cifrario** si fa riferimento al testo in chiaro, a quello cifrato, alla tecnica di cifratura e alla chiave crittografica. Nonostante che i primi metodi di cifratura abbiano origini assai antiche e che siano molto semplici ed elementari, essi costituiscono i concetti di base per la maggior parte degli algoritmi crittografici.

Un cifrario è detto **a sostituzione** quando ogni lettera del testo in chiaro viene sostituita con una lettera, o simbolo, dell'alfabeto cifrato. Il cosiddetto "Codice di Cesare", pur essendo alquanto banale, costituisce un caso significativo della cifratura a sostituzione. Con tale codice una determinata lettera viene cifrata con quella che si trova tre posti più avanti nell'alfabeto, cioè la A con la D, la B con la E, la C con la F, e così via. Quello che segue è un esempio di cifratura a sostituzione con il codice di Cesare:

```
ROSSO DI SERA BEL TEMPO SI SPERA
URVVR GL VHUD EHO WHPSR VL VSHUD
```

Si chiamano **cifrari a trasposizione** quei cifrari in cui le lettere del testo in chiaro non vengono sostituite, bensì spostate secondo una certa regola fissa detta appunto trasposizione. Un cifrario di questo tipo, molto importante, è il "Cifrario completo" che si basa su una suddivisione in blocchi del testo in chiaro, e sulla permutazione delle lettere stesse di ogni blocco. Un esempio di codifica con questo tipo di cifrario è il seguente:

```
ILMIO|REGNO|PERUN|CAVAL|LO--
MOILI|GOREN|RNPEU|VLCAA|-LO-
```

Vediamo come ottenere tale cifrario in *Mathematica*. Utilizziamo la funzione **transf** che rende gli elementi di una lista **l** ordinati secondo le posizioni contenute in un'altra lista **m**:

In[1]:=

```
transf[l_, m_] := Flatten[l][[#]]& /@ m;
```

Cifriamo un blocco della trasposizione precedente:

In[2]:=

```
StringJoin@@
 transf[Characters["ILMIO"],
  {3,5,1,2,4}]
```

Out[2]:=

```
MOILI
```

Il Data Encryption Standard

Il DES è l'algoritmo crittografico a chiave segreta più popolare e rappresenta uno *standard* crittografico da circa 20 anni. Il DES è un cifrario a blocchi, in quanto opera su sequenze di dati di 64 bit di testo mediante l'uso di una chiave segreta di 56 bit effettivi (64 con 8 bit di parità). L'algoritmo è strutturato in modo che in ingresso riceva un blocco di 64 bit di testo in chiaro, e in uscita restituisca un altro blocco di 64 bit di testo cifrato. È importante precisare che il DES è un sistema simmetrico, perché usa lo stesso algoritmo e la stessa chiave sia per la cifratura che per la decifrazione (è quindi possibi-

le adoperare lo stesso dispositivo per entrambe le operazioni). Per quanto riguarda la chiave segreta, che come detto è costituita da 56 bit, il DES effettua una trasformazione della stessa in 16 chiavi parziali di 48 bit, mediante un algoritmo di *Key Scheduling*, il quale usa i vari bit diverse volte.

La struttura dell'algoritmo è molto semplice, perché è basata su una combinazione delle tecniche tradizionali per la cifratura, vale a dire la trasposizione e la sostituzione (ottenuta con l'operatore di or esclusivo). Ogni blocco di 64 bit del testo dato in ingresso all'algoritmo subisce dapprima una trasposizione iniziale, poi viene elaborato da una certa funzione **F** in 16 passate, e infine subisce un'ulteriore trasposizione che produce il testo in uscita. La funzione **F** è ottenuta mediante semplici operazioni aritmetiche e logiche, su gruppi di 64 bit. Questa scelta ha reso il DES facilmente implementabile anche alla fine degli anni '70, quando la tecnologia non era certo quella di oggi.

Permutazione iniziale

Inizialmente viene effettuata la trasposizione dei bit del blocco in entrata nel seguente modo:

In[1]:=
```
doTI[l_] := transf[l, tabTI];
```

dove la tabella **tabTI** è riportata nell'incorniciato. A questo punto il blocco trasposto $\mathbf{B_0}$, viene fornito in *input* alla prima iterazione delle 16 passate effettuate dall'algoritmo.

Round del DES

Il DES esegue 16 iterazioni, dette *round*, ognuna delle quali è costituita da più operazioni che insieme formano un'unica funzione **F**. Ogni iterazione esegue un'operazione di "confusione" e di "diffusione"; la prima legata alla chiave parziale $\mathbf{K_i}$ e a cassette di sicurezza dette S-*Box*, la seconda è indipendente dalla chiave. La robustezza del DES è praticamente basata sulla combinazione di tali operazioni. Il blocco elaborato in ogni fase viene suddiviso in due sottoblocchi di 32 bit ciascuno, indicati con $\mathbf{L_i}$ e $\mathbf{R_i}$ tali che $\mathbf{B_i == L_i <> R_i}$.

Visto che realizzeremo il DES manipolando liste di 0 e 1 sono necessarie alcune funzioni per manipolare queste liste.

In[2] :=
```
xor[l_, m_] := Mod[Plus[l, m], 2];
split[l_, sz_] := Partition[l, sz];
```

La funzione **xor** effettua la somma binaria tra due liste di bit di eguale lunghezza; la funzione **split** spezza la lista **l** in sottoliste di lunghezza **sz**.

Seguono ora le definizioni delle funzioni di conversione da lista di bit a numero decimale **binToDec** e da lista di bit a stringa di 0 e 1 **binToStr**:

In[3] :=
```
binToDec[l_] := Inner[Times, l,
 2^Reverse[Range[0,Length[l]-1]],Plus];
binToStr[l_] := StringJoin[
 FromCharacterCode /@
  binToDec[split[l, 8]]];
```

Le seguenti funzioni si occupano di inserire all'inizio della stringa o della lista di bit un numero opportuno di zeri:

In[4] :=
```
fixedSz[s_, sz_] := Module[{len},
 len = sz - Length[s];
 If[len>0, Table[0,{len}] ~Join~ s, s]];
fixedSzStr[s_, sz_] := Module[
  {l=Characters[s],len},
  len = sz - Length[l];
  If[len>0, l=Table["0",{len}] ~Join~ l];
  StringJoin @@ l];
```

La funzione **binBlock**, dato un blocco di 8 caratteri, restituisce la lista dei bit che ne costituiscono la rappresentazione:

In[5] :=
```
blkChar[ch_]:= Flatten[IntegerDigits[
  ToCharacterCode[ch],2]];
doByte[ch_] := fixedSz[blkChar[ch], 8];
binBlock[msg_String] :=
  Flatten[doByte /@
    Characters[msg]] /;
      StringLength[msg]==8;
```

Durante ogni *round* si generano le chiavi parziali, spostando i bit della chiave e selezionandone 48. Dobbiamo calcolare ora le chiavi parziali per tutti e 16 i *round* del DES a partire dalla chiave data. Innanzitutto viene fatta una permutazione iniziale dei bit della chiave secondo la tabella **tabTK**

In[6] :=
```
doTK[k_] := transf[k, tabTK];
```

Successivamente si prende il risultato della trasposizione e si spezza in due blocchi da 28 bit ciascuno, ad ogni passo questi blocchi vengono fatti ruotare circolarmente verso sinistra di una quantità definita dalla tabella **rot**. La funzione che effettua la rotazione dei due blocchi è la seguente

In[7] :=
```
doRot[sd_,step_]:=
  {RotateLeft[sd[[1]], rot[[step]]],
   RotateLeft[sd[[2]], rot[[step]]]};
```

Dove **sd** è la chiave ottenuta dal passo precedente oppure dalla trasposizione **TK**. Ad ogni passo la chiave trasformata è composta da due liste di 28 bit. In ogni *round*, dopo avere trasformato la chiave mediante le rotazioni, si calcola una chiave di 48 bit che è quella effettivamente usata dall'algoritmo. Per fare questo dobbiamo effettuare una compressione a 48 bit della chiave di 56 con una trasposizione che scarta alcuni bit:

In[8] :=
```
doCK[l_]:=transf[l,tabCK];
```

La generazione della chiave da 48 bit al passo **i** è ottenuta ricorsivamente dalla seguente funzione (la ricorsione viene usata per creare una grossa lista contenente tutte le chiavi):

In[9] :=

```
keysGen[k_, i_] :=
  {doCK[Flatten[k]],
   keysGen[doRot[k,i],i+1]} /;
     i<17;
keysGen[k_,17]:=doCK[Flatten[k]];
```

La generazione delle chiavi è demandata alla seguente funzione che effettua la trasposizione iniziale, innesca la funzione **keysGen** e infine raggruppa le chiavi in 16 liste da 48 bit ciascuna:

In[10] :=

```
keysGen[k_] :=
 split[Flatten[keysGen[
   doRot[split[doTK[k],28],1],2]],
   48];
```

Successivamente si calcola una funzione **F** applicata ad un blocco di 32 bit. Questa funzione è realizzata in quattro fasi; innanzitutto espande a 48 bit questo blocco usando la seguente permutazione/espansione ottenuta trasponendo i bit e ripetendone alcuni (vedi la tabella **TE**).

In[11] :=

```
doExp[l_] := transf[l, tabE];
```

Si combinano poi i 48 bit tra loro mediante uno **xor** e il risultato si invia ad 8 S-*Box* che, mediante sostituzioni, producono 32 bit ancora permutati. Quindi ad una S-*Box* arrivano 6 bit che vengono utilizzati per produrne 4. La S-*Box* altro non è che una tabella 4x16; in cui il primo e l'ultimo bit identificano la riga nella tabella mentre i restanti quattro ne identificano una colonna. La tabella **tabZ** contiene numeri compresi tra 0 e 15 e quindi produce 4 bit. La funzione **doZ[l,i]** effettua una compressione dei 6 bit contenuti nella lista **l**, usando la S-*Box* numero **i**. La funzione **doCom** esegue la compressione.

In[12] :=

```
idx[l_]:= binToDec[transf[l, tabIdx]];
doZ[l_,i_]:= fixedSz[IntegerDigits[
  tabZ[[i]][[idx[l]+1]], 2], 4];
doCom[l_]:=Flatten[
  doZ[l[[#]],#]&/@Range[8]];
```

Una volta ottenuti **4*8 == 32** bit, applicando la compressione viene eseguita un'ulteriore permutazione su di essi

In[13] :=

```
doP[l_] := transf[l, tabP];
```

Queste 4 operazioni formano la funzione **F**:

In[14] :=

```
F[d_,k_]:=
 doP[doCom[split[xor[doExp[d],k],6]]];
```

La funzione **F** viene a sua volta combinata mediante uno **xor** con la parte sinistra **Li**, ottenendo così come risultato, la parte destra del passo successivo. La parte destra del passo attuale diventa invece quella sinistra nella fase successiva. Tutto questo viene ripetuto per 16 volte.

Si osservi che alla 16-esima iterazione i due sottoblocchi L_{16} e R_{16} non vengono scambiati tra loro, bensì subito riuniti a

Le tabelle

```
tabTF={40, 8, 48, 16, 56, 24, 64, 32, 39, 7, 47,
       15, 55, 23, 63, 31, 38, 6, 46, 14, 54, 22,
       62, 30, 37, 5, 45, 13, 53, 21, 61, 29, 36,
       4, 44, 12, 52, 20, 60, 28, 35, 3, 43, 11,
       51, 19, 59, 27, 34, 2, 42, 10, 50, 18, 58,
       26, 33, 1, 41, 9, 49, 17, 57, 25};
tabTI={58, 50, 42, 34, 26, 18, 10, 2, 60, 52, 44,
       36, 28, 20, 12, 4, 62, 54, 46, 38, 30, 22,
       14, 6, 64, 56, 48, 40, 32, 24, 16, 8, 57,
       49, 41, 33, 25, 17, 9, 1, 59, 51, 43, 35,
       27, 19, 11, 3, 61, 53, 45, 37, 29, 21, 13,
       5, 63, 55, 47, 39, 31, 23, 15, 7};
tabE={32, 1, 2, 3, 4, 5, 4, 5, 6, 7, 8, 9, 8, 9,
       10, 11, 12, 13, 12, 13, 14, 15, 16, 17, 16,
       17, 18, 19, 20, 21, 20, 21, 22, 23, 24, 25,
       24, 25, 26, 27, 28, 29, 28, 29, 30, 31, 32,
       1};
tabP={16, 7, 20, 21, 29, 12, 28, 17, 1, 15, 23, 26,
       5, 18, 31, 10, 2, 8, 24, 14, 32, 27, 3, 9,
       19, 13, 30, 6, 22, 11, 4, 25};
tabZ1={14, 4, 13, 1, 2, 15, 11, 8, 3, 10, 6, 12, 5,
       9, 0, 7, 0, 15, 7, 4, 14, 2, 13, 1, 10, 6,
       12, 11, 9, 5, 3, 8, 4, 1, 14, 8, 13, 6, 2,
       11, 15, 12, 9, 7, 3, 10, 5, 0, 15, 12, 8,
       2, 4, 9, 1, 7, 5, 11, 3, 14, 10, 0, 6, 13};
tabZ2={15, 1, 8, 14, 6, 11, 3, 4, 9, 7, 2, 13, 12,
       0, 5, 10, 3, 13, 4, 7, 15, 2, 8, 14, 12, 0,
       1, 10, 6, 9, 11, 5, 0, 14, 7, 11, 10, 4,
       13, 1, 5, 8, 12, 6, 9, 3, 2, 15, 13, 8, 10,
       1, 3, 15, 4, 2, 11, 6, 7, 12, 0, 5, 14, 9};
tabZ3={10, 0, 9, 14, 6, 3, 15, 5, 1, 13, 12, 7, 11,
       4, 2, 8, 13, 7, 0, 9, 3, 4, 6, 10, 2, 8, 5,
       14, 12, 11, 15, 1, 13, 6, 4, 9, 8, 15, 3,
       0, 11, 1, 2, 12, 5, 10, 14, 7, 1, 10, 13,
       0, 6, 9, 8, 7, 4, 15, 14, 3, 11, 5, 2, 12};
tabZ4={7, 13, 14, 3, 0, 6, 9, 10, 1, 2, 8, 5, 11,
       12, 4, 15, 13, 8, 11, 5, 6, 15, 0, 3, 4, 7,
       2, 12, 1, 10, 14, 9, 10, 6, 9, 0, 12, 11,
       7, 13, 15, 1, 3, 14, 5, 2, 8, 4, 3, 15, 0,
       6, 10, 1, 13, 8, 9, 4, 5, 11, 12, 7, 2,
       14};
tabZ5={2, 12, 4, 1, 7, 10, 11, 6, 8, 5, 3, 15, 13,
       0, 14, 9, 14, 11, 2, 12, 4, 7, 13, 1, 5, 0,
       15, 10, 3, 9, 8, 6, 4, 2, 1, 11, 10, 13, 7,
       8, 15, 9, 12, 5, 6, 3, 0, 14, 11, 8, 12, 7,
       1, 14, 2, 13, 6, 15, 0, 9, 10, 4, 5, 3};
tabZ6={12, 1, 10, 15, 9, 2, 6, 8, 0, 13, 3, 4, 14,
       7, 5, 11, 10, 15, 4, 2, 7, 12, 9, 5, 6, 1,
       13, 14, 0, 11, 3, 8, 9, 14, 15, 5, 2, 8,
       12, 3, 7, 0, 4, 10, 1, 13, 11, 6, 4, 3, 2,
       12, 9, 5, 15, 10, 11, 14, 1, 7, 6, 0, 8,
       13};
tabZ7={4, 11, 2, 14, 15, 0, 8, 13, 3, 12, 9, 7, 5,
       10, 6, 1, 13, 0, 11, 7, 4, 9, 1, 10, 14, 3,
       5, 12, 2, 15, 8, 6, 1, 4, 11, 13, 12, 3, 7,
       14, 10, 15, 6, 8, 0, 5, 9, 2, 6, 11, 13, 8,
       1, 4, 10, 7, 9, 5, 0, 15, 14, 2, 3, 12};
tabZ8={13, 2, 8, 4, 6, 15, 11, 1, 10, 9, 3, 14, 5,
       0, 12, 7, 1, 15, 13, 8, 10, 3, 7, 4, 12, 5,
       6, 11, 0, 14, 9, 2, 7, 11, 14, 1, 9, 12,
       14, 2, 0, 6, 10, 13, 15, 3, 5, 8, 2, 1, 14,
       7, 4, 10, 8, 13, 15, 12, 9, 0, 3, 5, 6,
       11};
tabZ={tabZ1, tabZ2, tabZ3, tabZ4, tabZ5, tabZ6,
       tabZ7, tabZ8};
tabIdx={1, 6, 2, 3, 4, 5};
tabTK={57, 49, 41, 33, 25, 17, 9, 1, 58, 50, 42,
       34, 26, 18, 10, 2, 59, 51, 43, 35, 37, 19,
       11, 3, 60, 52, 44, 36, 63, 55, 47, 39, 31,
       23, 15, 7, 62, 54, 46, 38, 30, 22, 14, 6,
       61, 53, 45, 37, 29, 21, 13, 5, 28, 20, 12,
       4};
tabCK={14, 17, 11, 24, 1, 5, 3, 28, 15, 6, 21, 10,
       23, 19, 12, 4, 26, 8, 16, 7, 27, 20, 13, 2,
       41, 52, 31, 37, 47, 55, 30, 40, 51, 45, 33,
       48, 44, 49, 39, 56, 34, 53, 46, 42, 50, 36,
       29, 32};
rot={1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2,
       1};
```

formare il blocco B_{16} che viene fornito in ingresso alla trasposizione finale:

In[15] :=

```
doTF[l_] := transf[l, tabTF];
```

Ciò serve soprattutto per realizzare la simmetria del DES, cioè poter effettuare la cifratura e la decifratura con lo stesso algoritmo.

Il DES su un blocco è quindi dato dalla seguente funzione:

In[16] :=

```
blockDES[bl_, keys_] :=
 Module[{sd,ret},
  sd = split[doTI[binBlock[bl]],32];
  For[i=1, i<17, i++,
    sd = {sd[[2]],
          xor[sd[[1]],
          F[sd[[2]],
          keys[[i]]]]}];
 binToStr[doTF[Join[sd[[2]], sd[[1]]]]];
```

La cifratura e la decifrazione del DES

In realtà il DES non si limita a giustapporre i blocchi cifrati, usa una tecnica detta *cypher chaining*, per confondere ulteriormente il messaggio. Se M_i sono i blocchi da 64 bit che formano il messaggio allora i blocchi cifrati C_i sono così ottenuti:

C_1 = blockDES[M_1];

C_i = blockDES[M_i C_{i-1}];

La chiave verrà assegnata con la seguente funzione:

In[1] :=

```
setKey[k_]:= subkeys=keysGen[DESKey = k];
```

La funzione **prepBlocks** prende in ingresso il messaggio da cifrare/decifrare e lo suddivide in blocchi da 8 caratteri ciascuno:

In[2] :=

```
prepBlocks[msg_] := Module[
  {len = StringLength[msg]},
  len += Mod[8 len - len, 8];
  StringJoin /@
    split[Characters[
     fixedSzStr[msg, len]], 8]];
```

Siamo quindi pronti per cifrare e decifrare in DES con le seguenti funzioni:

In[3] :=

```
encryptDES[msg_String] :=
 Module[
 {DESblocks=prepBlocks[msg], ret, tmp, i},
ret=c = blockDES[DESblocks[[1]], subkeys];
  Do[tmp = binToStr[xor[
     binBlock[DESblocks[[i]]],
              binBlock[c]]];
     ret = StringJoin[
        ret,c=blockDES[tmp, subkeys]],
    {i,2,Length[DESblocks]}];
  ret];
decryptDES[msg_String] :=
 Module[
  {DESblocks = prepBlocks[msg], ret,
  subK = Reverse[subkeys], tmp, tmp1, i},
  ret = blockDES[DESblocks[[1]], subK];
Do[tmp = blockDES[DESblocks[[i]], subK];
     tmp1 = binToStr[
       xor[binBlock[tmp],
       binBlock[DESblocks[[i - 1]]]]];
     ret = StringJoin[ret, tmp1],
  {i,2,Length[DESblocks]}];
  ret];
```

Si osservi come la cifratura e la decifrazione siano praticamente identiche, l'unica differenza è data dal fatto che il *cypher chaining* viene applicato inversamente e che le sottochiavi vengono applicate in ordine inverso. Ecco infine una prova del funzionamento:

In[4] :=

```
key={0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0,
1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 0,
0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0,
0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0,
1, 0, 1, 1, 1, 1, 0, 0, 0, 1};
setKey[key];
```

In[5] :=

```
mess =
"Questo è un testo crittato in DES      ";
encryptDES[mess]//InputForm
```

Out[5] :=

```
"\035„ªìgh$\(Paragraph)ÚÍîõ)ô≥\013C2ü\(Cr
oss)6ÍUvçÎÛñG«\020&fÕ>zªH/9"
```

Si è usata la **InputForm** in quanto il codice cifrato contiene caratteri non stampabili che avrebbero appunto messo in crisi la stampa del testo (!)

In[6] :=

```
decryptDES[%]//InputForm
```

Out[6] :=

```
"Questo è un testo crittato in DES      "
```

MC

Maggiori dettagli e una copia elettronica del codice si possono trovare alla URL: http://www.di.unipi.it/~romani/romani.html

Bibliografia

C. Giustozzi, **Il Data Encryption Standard,** Mcmicrocomputer n. 136 (gennaio 1994).

A.Sgarro, **Crittografia: tecniche di protezione dei dati riservati,** F. Muzzio & C., 1993.

B. Schneier, **Applied Cryptography**, J. Wiley & Sons, 1996.

The Mathematica Book, 3rd ed. (Wolfram Media/Cambridge University Press, 1996)