Sicurezza su Reti **Anno Acc. 1997-98**

PROF. ALFREDO DE SANTIS

Implementazione dello SKIPJACK

a cura di Galdo Stefano *27 Novembre 1998*

Contents

1 Introduzione

Nell' Escrowed Encryption Standard (EES) [1] si specifica l'uso dell'algoritmo di cifratura SKIPJACK e del Law Enforcement Access Field (LEAF) Creation Method i quali vanno implementati in dispositivi elettronici protetti contro modifiche e usi non autorizzati. Questi dispositivi sono contenuti in moduli crittografici per poi essere integrati in prodotti per la cifratura e la decifratura di telecomunicazioni.

Questi dispositivi devono essere testati e convalidati dal NIST per essere dichiarati conformi a questo standard e con il FIPS 140-1e FIPS 81.

2 Specifiche dell'algoritmo SKIPJACK

Le operazioni usate nell' algoritmo sono le seguenti

- X$\oplus$Y: Or esclusivo bit a bit tra X e Y.
- X$\|$Y: Concatenazione del byte X con il byte Y, X$\|Y = X * 2^8 + Y$.

L'algoritmo di cifratura/decifratura SKIPJACK [2] cifra quattro parole da 2 byte ognuna $w_i^0, 1 \leq i \leq 4$, alternando l'uso di due regole: Regola A e Regola B. Si inizializza un contatore: *counter*=1, e si parte con otto iterazioni usando la Regola A poi altre otto usando la Regola B, si torna alla Regola A per altre otto iterazioni e si conclude con otto applicazioni della Regola B. Ad ogni iterazione *counter* viene incrementato. Si giunge quindi all'output $w_1^{32}w_2^{32}w_3^{32}w_4^{32}$. L'apice indica il numero dell'iterazione corrente.

Questo procedimento é schematizzato nella seguente figura:

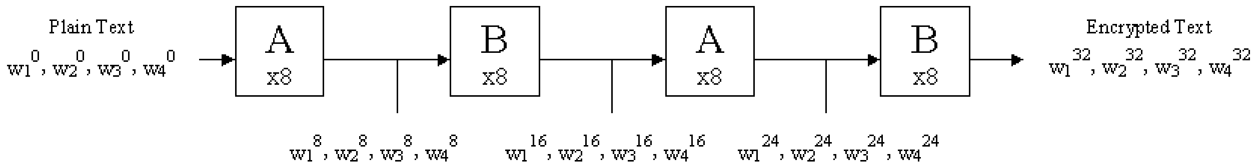

Figure 1: Cifratura

Le Regole A e B per la (k+1)-esima iterazione sono cosí definite:

CIFRATURA

Regola A	Regola B
$w_1^{k+1} = G^k(w_1^k) \oplus w_4^k \oplus counter^k$	$w_1^{k+1} = w_4^k$
$w_2^{k+1} = G^k(w_1^k)$	$w_2^{k+1} = G^k(w_1^k)$
$w_3^{k+1} = w_2^k$	$w_3^{k+1} = w_1^k \oplus w_2^k \oplus counter^k$
$w_4^{k+1} = w_3^k$	$w_4^{k+1} = w_3^k$

Schematizziamo le due regole nelle Figure 2 e 3.

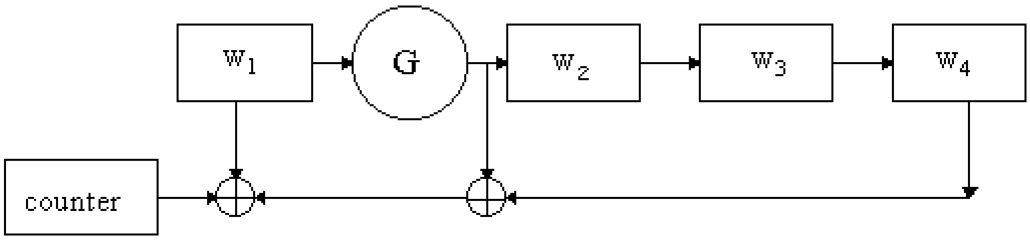

Figure 2: Diagramma della Regola A

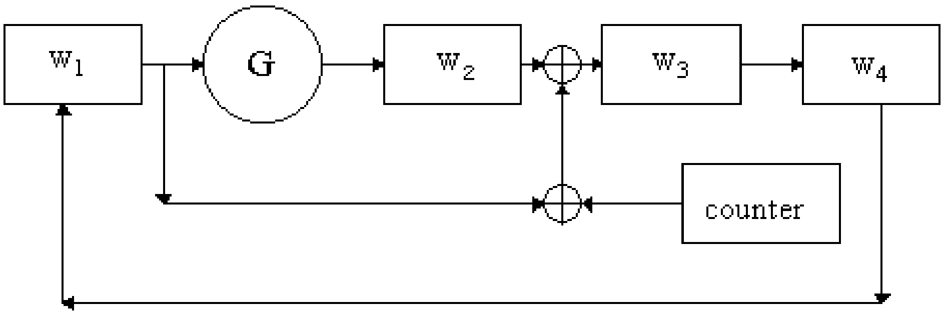

Figure 3: Diagramma della Regola B

2.1 La funzione G

La funzione G prende in input 2 byte e ne restituisce altrettanti in output. Nella sua struttura possiamo distinguere quattro differenti round in ognuno dei quali viene fatta una sostituzione sfruttando una tabella (FTable) di valori prefissati; inoltre ogni round utilizza anche uno dei 10 byte della chiave *cv*. Definiamo $G^k(g_1 \| g_2) = g_5 \| g_6$, dove $g_i = F(g_{i-1} \oplus cv_{4k+i-3}) \oplus g_{i-2}$; k é il numero dell'iterazione, F é la funzione che attua la sostituzione dei valori dalla FTable, infine cv_{4k+i-3} é il $(4k+i-3)$-esimo byte della chiave schedulata. Quindi esplicitando i quattro round si ha:

1. $g_3 = F(g_2 \oplus cv_{4k}) \oplus g_1$
2. $g_4 = F(g_3 \oplus cv_{4k+1}) \oplus g_2$
3. $g_5 = F(g_4 \oplus cv_{4k+2}) \oplus g_3$
4. $g_6 = F(g_5 \oplus cv_{4k+3}) \oplus g_4$

La funzione G é illustrata nella figura seguente:

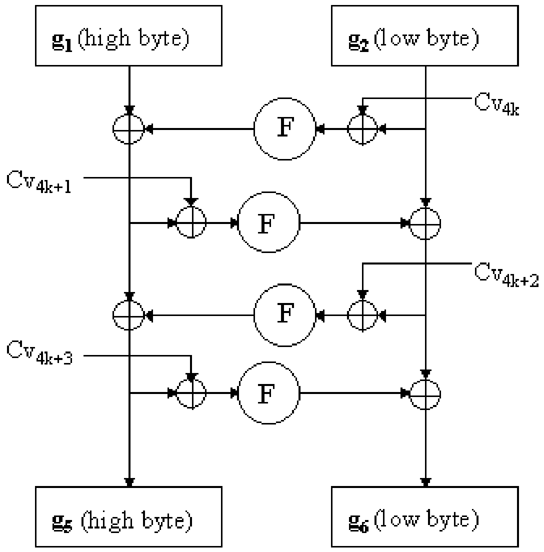

Figure 4: Diagramma della G

2.2 Schedulazione della chiave

La chiave, che é lunga 10 byte, é vista come un vettore circolare, dove il decimo byte precede il primo. Quello che in pratica si fa é applicare all'indice del vettore chiave la riduzione in modulo; ad esempio se si fa riferimento al byte cv_{4k+i-3} il valore $4k+i-3$ é da considerare modulo 10.

2.3 Funzione F e FTable

La FTable é una tavola di valori esadecimali prefissati di dimensione 16x16 :

	$x0$	$x1$	$x2$	$x3$	$x4$	$x5$	$x6$	$x7$	$x8$	$x9$	xA	xB	xC	xD	xE	xF
$0x$	$a3$	$d7$	09	83	$f8$	48	$f6$	$f4$	$b3$	21	15	78	99	$b1$	af	$f9$
$1x$	$c7$	$2d$	$4d$	$8a$	ce	$4c$	ca	$2e$	52	95	$d9$	$1e$	$4c$	38	44	28
$2x$	$0a$	df	02	$a0$	17	$f1$	60	68	12	$b7$	$7a$	$c3$	$e9$	fa	$3d$	53
$3x$	96	84	$6b$	ba	$f2$	63	$9a$	19	$7c$	ae	$c5$	$f5$	$f7$	16	$6a$	$a2$
$4x$	39	$b6$	$7b$	$0f$	$c1$	93	81	$1b$	cc	$b4$	$1a$	ca	$d0$	91	$2f$	$b8$
$5x$	55	$b9$	da	85	$3f$	41	bf	$c0$	$5a$	58	80	$5f$	66	$0b$	$d8$	90
$6x$	35	$d5$	$c0$	$a7$	33	06	65	69	45	00	94	56	$6d$	98	$9b$	76
$7x$	97	fc	$b2$	$c2$	$b0$	fe	db	20	$e1$	cb	$d6$	$c4$	dd	47	$4a$	$1d$
$8x$	42	ed	$9e$	$6e$	49	$3c$	cd	43	27	$d2$	07	$d4$	de	$c7$	67	18
$9x$	89	cb	30	$1f$	$8d$	$c6$	$8f$	aa	$c8$	74	dc	$c9$	$5d$	$5c$	31	$a4$
Ax	70	88	61	$2c$	$9f$	$0d$	$2b$	87	50	82	54	64	26	$7d$	03	40
Bx	34	$4b$	$1c$	73	$d1$	$c4$	fd	$3b$	cc	fb	$7f$	ab	$e6$	$3e$	$5b$	$a5$
Cx	ad	04	23	$9c$	14	51	22	$f0$	29	79	71	$7e$	ff	$8c$	$0c$	$e2$
Dx	$0c$	ef	bc	72	75	$6f$	37	$a1$	cc	$d3$	$8e$	62	$8b$	86	10	$e8$
Ex	08	77	11	be	92	$4f$	24	$c5$	32	36	$9d$	cf	$f3$	$a6$	bb	ac
Fx	$5e$	$6c$	$a9$	13	57	25	$b5$	$c3$	bd	$a8$	$3a$	01	05	59	$2a$	46

La funzione F riceve in input un byte che viene diviso in due componenti di 4 bit, parte alta e parte bassa. La parte alta dell'input va ad indicizzare la riga della FTable mentre la parte bassa indicizza la colonna, individuando cosí un elemento nella tabella. Ad esempio, F(7A)=d6.

2.4 Decifratura

La fase di decodifica, qui descritta, ha una struttura analoga a quella vista per la codifica. In luogo delle relgole A e B troviamo le loro rispettive inverse la Regola A^{-1} e la Regola B^{-1} descritte di seguito:

DECIFRATURA

Regola A^{-1}

$$
\begin{aligned}
w_1^{k-1} &= [G^{k-1}]^{-1}(w_2^k) \\
w_2^{k-1} &= w_3^k \\
w_3^{k-1} &= w_4^k \\
w_4^{k-1} &= w_1^k \oplus w_2^k \oplus counter^{k-1}
\end{aligned}
$$

Regola B^{-1}

$$
\begin{aligned}
w_1^{k-1} &= [G^{k-1}]^{-1}(w_2^k) \\
w_2^{k-1} &= [G^{k-1}]^{-1}(w_2^k) \oplus w_3^k \oplus counter^{k-1} \\
w_3^{k-1} &= w_4^k \\
w_4^{k-1} &= w_1^k
\end{aligned}
$$

L'algoritmo ha come input il testo cifrato $w_1^{32}w_2^{32}w_3^{32}w_4^{32}$, in questo caso k parte da 32; *counter* é inizializzato a 32. Si inizia con otto iterazioni usando la Regola B^{-1} poi otto usando la Regola A^{-1}, si ritorna alla Regola B^{-1} per altre otto iterazioni per poi concludere con le ultime otto usando nuovamente la Regola A^{-1}. A differenza della fase di cifratura, *counter* viene decrementato ad ogni passo. Alla fine dell'algoritmo si ottiene il testo in chiaro $w_1^0, w_2^0, w_3^0, w_4^0$ come mostrato nella seguente figura:

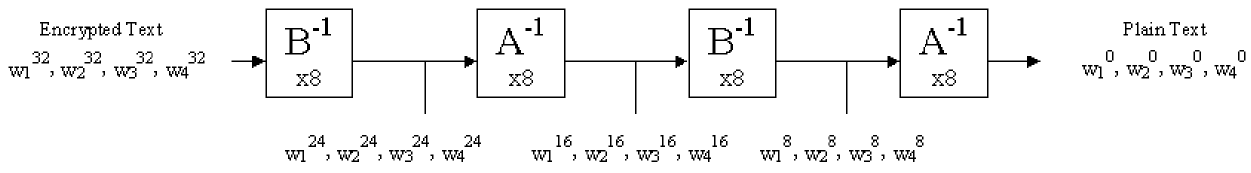

Figure 5: Decifratura

Con G^{-1} si intende indicare l' inversa della funzione G, che é cosí definita:

$$[G^k]^{-1}(g_5 \| g_6) = g_1 \| g_2 \text{ dove } g_{i-2} = F(g_{i-1} \oplus cv_{4k+i-3}) \oplus g_i$$

che possiamo anche rappresentare schematicamente:

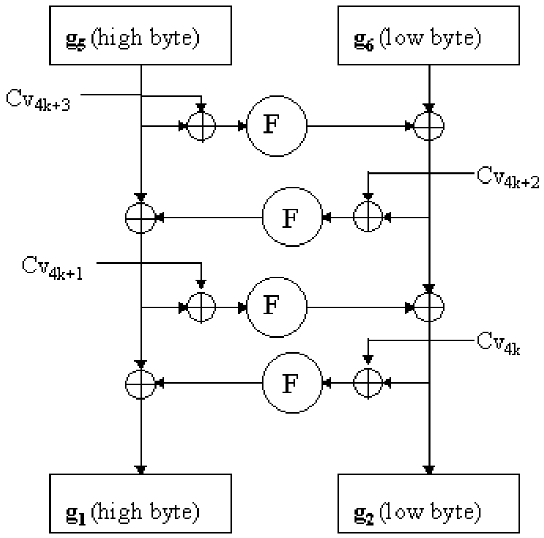

Figure 6: Diagramma della G^{-1}

3 Modalitá operative dello SKIPJACK

Le modalitá operative dello SKIPJACK sono un sottoinsieme di quelle del DES [3].

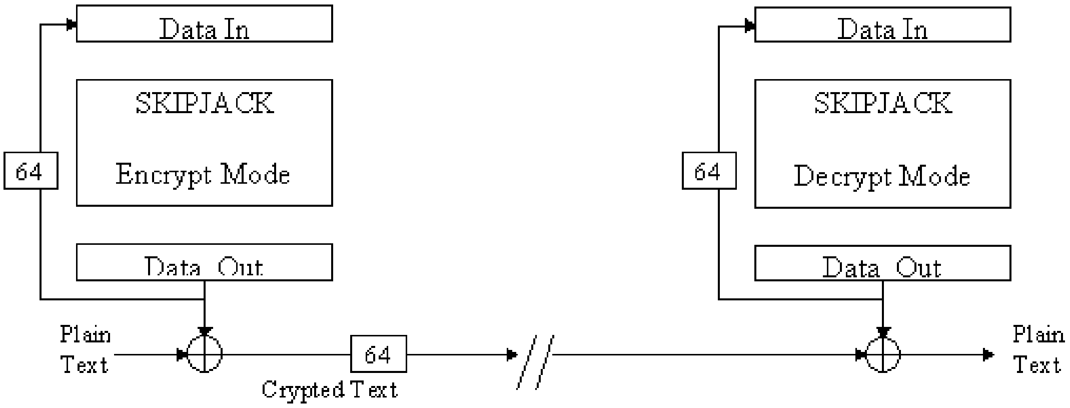

Figure 7: Output Feed-Back

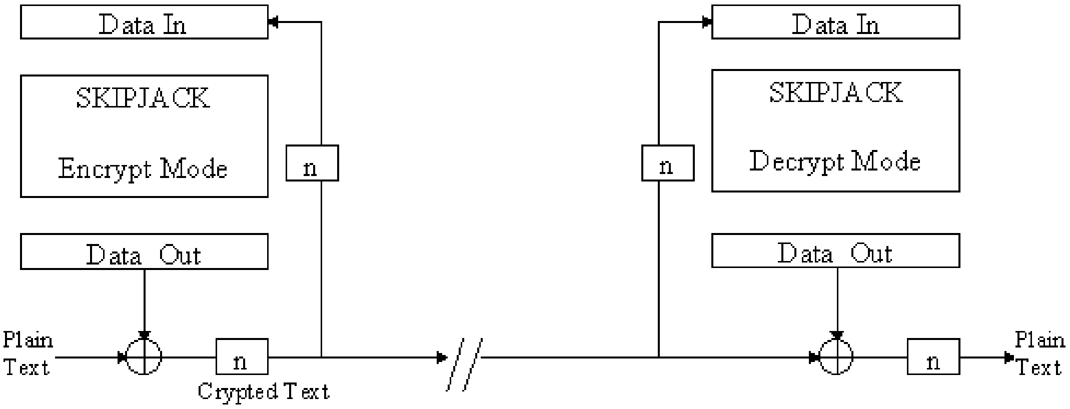

Figure 8: Cipher Feed-Back

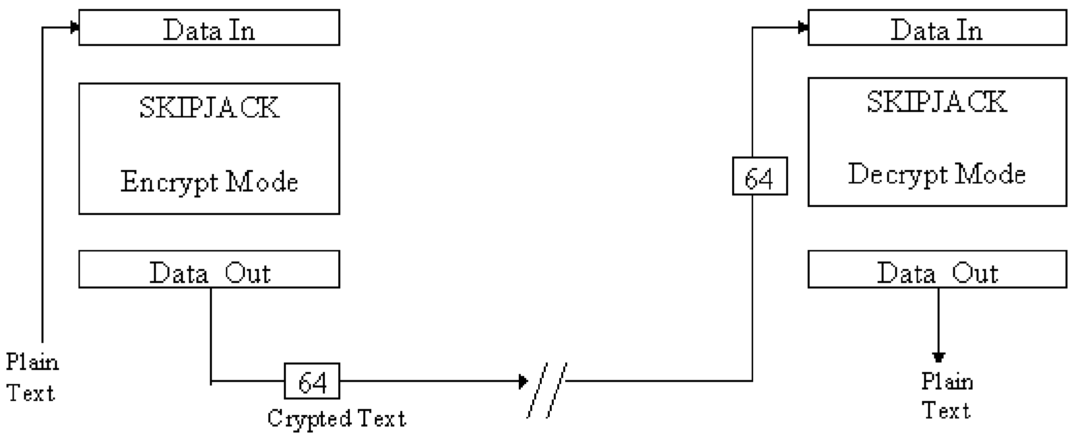

Figure 9: Codebook

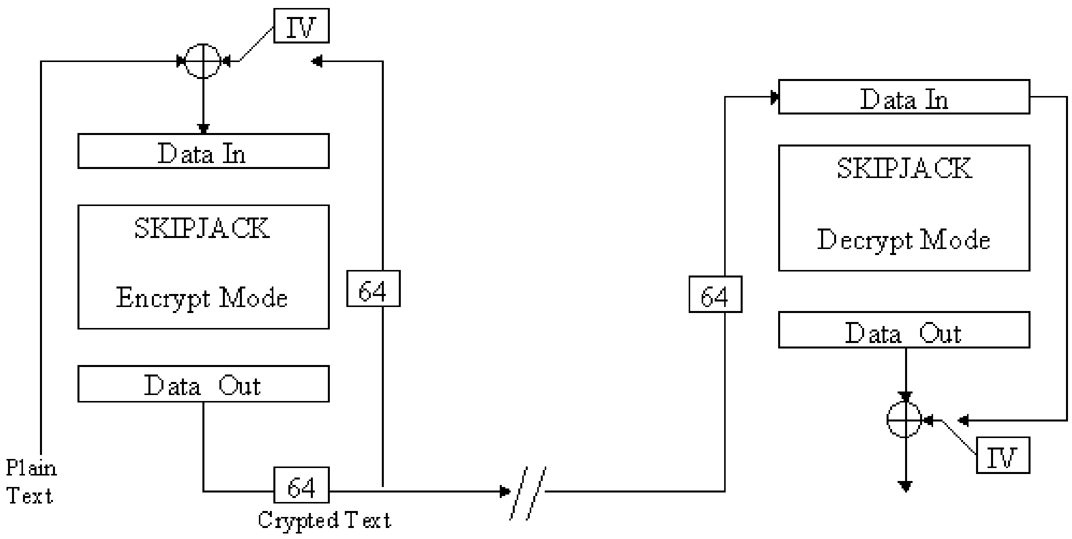

Figure 10: Cipher-Block Chaining

4 Il codice

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>

const unsigned int  FTable[16][16]={
0xa3,0xd7,0x09,0x83,0xf8,0x48,0xf6,0xf4,0xb3,0x21,0x15,0x78,0x99,0xb1,0xaf,0xf9,
0xe7,0x2d,0x4d,0x8a,0xce,0x4c,0xca,0x2e,0x52,0x95,0xd9,0x1e,0x4e,0x38,0x44,0x28,
0x0a,0xdf,0x02,0xa0,0x17,0xf1,0x60,0x68,0x12,0xb7,0x7a,0xc3,0xe9,0xfa,0x3d,0x53,
0x96,0x84,0x6b,0xba,0xf2,0x63,0x9a,0x19,0x7c,0xae,0xe5,0x75,0xf7,0x16,0x6a,0xa2,
0x39,0xb6,0x7b,0x0f,0xc1,0x93,0x81,0x1b,0xee,0xb4,0x1a,0xea,0xd0,0x91,0x2f,0xb8,
0x55,0xb9,0xda,0x85,0x3f,0x41,0xbf,0xe0,0x5a,0x58,0x80,0x5f,0x66,0x0b,0xd8,0x90,
```

```
0x35,0xd5,0xc0,0xa7,0x33,0x06,0x65,0x69,0x45,0x00,0x94,0x56,0x6d,0x98,0x9b,0x76,
0x97,0xfc,0xb2,0xc2,0xb0,0xfe,0xdb,0x20,0xe1,0xeb,0xd6,0xe4,0xdd,0x47,0x4a,0x1d,
0x42,0xed,0x9e,0x6e,0x49,0x3c,0xcd,0x43,0x27,0xd2,0x07,0xd4,0xde,0xc7,0x67,0x18,
0x89,0xcb,0x30,0x1f,0x8d,0xc6,0x8f,0xaa,0xc8,0x74,0xdc,0xc9,0x5d,0x5c,0x31,0xa4,
0x70,0x88,0x61,0x2c,0x9f,0x0d,0x2b,0x87,0x50,0x82,0x54,0x64,0x26,0x7d,0x03,0x40,
0x34,0x4b,0x1c,0x73,0xd1,0xc4,0xfd,0x3b,0xcc,0xfb,0x7f,0xab,0xe6,0x3e,0x5b,0xa5,
0xad,0x04,0x23,0x9c,0x14,0x51,0x22,0xf0,0x29,0x79,0x71,0x7e,0xff,0x8c,0x0e,0xe2,
0x0c,0xef,0xbc,0x72,0x75,0x6f,0x37,0xa1,0xec,0xd3,0x8e,0x62,0x8b,0x86,0x10,0xe8,
0x08,0x77,0x11,0xbe,0x92,0x4f,0x25,0xc5,0x32,0x36,0x9d,0xcf,0xf3,0xa6,0xbb,0xac,
0x5e,0x6c,0xa9,0x13,0x57,0x25,0xb5,0xe3,0xbd,0xa8,0x3a,0x01,0x05,0x59,0x2a,0x46
};

const unsigned int cv[10]={0x00,0x99,0x88,0x77,0x66,0x55,0x44,0x33,0x22,0x11};

unsigned int h1,h2,h3,h4;
unsigned int w1,w2,w3,w4;
int counter,k;


// funzione int F(int) :Prende in input un numero a 8 bit lo divide in due
numeri da 4 bit e li
// usa come indice di riga (parte alta) e di colonna (parte bassa) e
restituisce il valore punta
// to della FTable (Vedi sopra)

unsigned int F(unsigned char x) {
unsigned char l,h;

h=x>>4;
l=x<<4;
l=l>>4;
h=h;

return FTable[h][l];
}

// Funzione int MOD (int,int) :Calcola il resto della divisione a/b per
sottazioni successive
int mod (int a, int b) {
while (a>=b) a=a-b;
return a;
}

// Funzione int G(int) Prende in input un numero a 16 bit e lo divide in due
da otto g1, g2
// effettua poi delle trasformazione usando la funzione F
unsigned int IG(unsigned int w) {
   unsigned char g1,g2,g3,g4,g5,g6;
   unsigned int  h;
   g5=w>>8;
```

```
   h=w<<8;
   g6=h>>8;
   g4=F(g5 ^ cv[mod(4*(k)+3,10)]) ^ g6;
   g3=F(g4 ^ cv[mod((4*(k)+2),10)]) ^g5;
   g2=F(g3 ^ cv[mod((4*(k)+1),10)]) ^g4;
   g1=F(g2 ^ cv[mod((4*(k)),10)]) ^g3;
   w=g1;
   w=w<<8;
   w=w+g2;
   return w;
}

// funzione G
unsigned int G(unsigned int w) {
   unsigned int h;
   unsigned char g1,g2,g3,g4,g5,g6;
   g1=w>>8;
   h=w<<8;
   g2=h>>8;
   if (k==2) {
   }
   g3=F(g2 ^ cv[mod(4*k,10)]) ^ g1;
   g4=F(g3 ^ cv[mod((4*k+1),10)]) ^g2;
   g5=F(g4 ^ cv[mod((4*k+2),10)]) ^g3;
   g6=F(g5 ^ cv[mod((4*k+3),10)]) ^g4;
   w=g5;
   w=w<<8;
   w=w+g6;

   return w;
}
void  RuleA (){

   h1=G(w1) ^ w4 ^ counter;
   h2=G(w1);
   h3=w2;
   h4=w3;

   w1=h1;
   w2=h2;
   w3=h3;
   w4=h4;
}
void RuleB () {

   h1=w4;
   h2=G(w1);
   h3=w1^w2^counter;
   h4=w3;
```

```
   w1=h1;
   w2=h2;
   w3=h3;
   w4=h4;

}
void IRuleA() {

   h1=IG(w2);

   h2=w3;
   h3=w4;
   h4=w1^w2^(counter);
   w1=h1;
   w2=h2;
   w3=h3;
   w4=h4;
}
void IRuleB() {
   h1=IG(w2);

   h2=IG(w2)^w3^(counter);

   h3=w4;
   h4=w1;

   w1=h1;
   w2=h2;
   w3=h3;
   w4=h4;
}
// Funzione di Codifica
void Encrypt() {
   int i;
   counter=1;
   k=0;

   for (i=1;i<=8;i++) {
      RuleA();
      printf ("%x%x  %x%x\n",w1,w2,w3,w4);
      counter++;
      k++;

   }

   for (i=1;i<=8;i++) {
      RuleB();
      printf ("%x%x  %x%x\n",w1,w2,w3,w4);
      counter++;
      k++;
```

```
   }
   for (i=1;i<=8;i++) {
      RuleA();
      printf ("%x%x  %x%x\n",w1,w2,w3,w4);
      counter++;
      k++;
   }
   for (i=1;i<=8;i++) {
      RuleB();
      printf ("%x%x  %x%x\n",w1,w2,w3,w4);
      counter++;
      k++;
   }


}
// Funzione di Decodifica
void Decrypt() {
   int i;
   counter=32;
   k=32;
   printf ("Counter=%d k=%d\n",counter,k);
   for(i=1;i<=8;i++) {
      k--;
      IRuleB();
      counter--;

   }
   for (i=1;i<=8;i++) {
       k--;
      IRuleA();
      counter--;

   }
   for (i=1;i<=8;i++) {
      k--;
      IRuleB();
      counter--;

   }
   for (i=1;i<=8;i++) {
       k--;
      IRuleA();
      counter--;

   }
}
```

```
void main() {
   int i;

   // Testo da cifrare
   w1=0x3322;
   w2=0x1100;
   w3=0xddcc;
   w4=0xbbaa;
   printf ("PLAIN : w1=%x w2=%x w3=%x w4=%x\n",w1,w2,w3,w4);
   Encrypt();
   printf ("ENCRYPTED TEXT: w1=%x w2=%x w3=%x w4=%x\n",w1,w2,w3,w4);
   Decrypt();
   printf ("DECRYPTED TEXT:w1=%x w2=%x w3=%x w4=%x\n",w1,w2,w3,w4);


}
```

Riferimenti

1. US DEPARTMENT OF COMMERCE Technology Administration/National Institute of Standards and Technology FIPS PUB 185, 9 February 1994. http://jya.com/fips.htm

2. SKIPJACK and KEA Algorithm Specifications, Version 2.0, 29 May 1998.
   http://www.jya.com/skipjack-spec.htm

3. US DEPARTMENT OF COMMERCE Technology Administration/National Institute of Standards and Technology, DES MODES OF OPERATION, FIPS PUB 81, 2 December 1980.