ISO 9979/0019

ALGORITHM REGISTER ENTRY

a) ISO Entry Name {ISO standard 9979 Cipher(19)}

b) Name of Algorithm CIPHERUNICORN-E

c) Intended Range of Application
1. Confidentiality
2. Hash Function - as detailed in ISO 10118-2
3. Authentication - as detailed in ISO 9798
4. Data Integrity - as detailed in ISO 9797

d) Cryptographic Interface Parameters

1. Input size	64 bits
2. Output size	64 bits
3. Key length:	128 bits
4. Round number	positive integer

e) Test Data

ROUND NUMBER	16
KEY	(0000 0000 0000 0000 0000 0000 0000 0000)$_{hex}$
INPUT DATA	(1234 5678 9abc def0)$_{hex}$
OUTPUT DATA	(b500 5b80 1083 0d37)$_{hex}$

f) Sponsoring Authority
Information-Technology Promotion Agency,
Japan(IPA)
Shuwa-Shibakoen 3-chome Bldg., 6F,
3-1-38 Shibakoen,
Minato-ku, Tokyo 105, JAPAN
Tel:
+81-3-3437-2301
Fax:
+81-3-3437-9421

Registration Requested by
NEC Corporation
C&C Media Research Laboratories

Contact for Information
Yukiyasu TSUNOO
Assistant Manager
NEC Corporation
C&C Media Research Laboratories
Security Technology Group
4-1-1 Miyazaki, Miyamae-ku,
Kawasaki 216-8555, JAPAN
Tel:
+81-44-856-2141
Fax:
+81-44-856-2235

g) Date of submission: 1998.3.10
Date of registration 6th July 1998

h) Whether the Subject of a National Standard: No.

i) Patent - License Restriction A patent applied for:
1. Japan,No.09-213274

For commercial use of CIPHERUNICORN-E, a license and fee is required.

j) References

k) Description of Algorithm

CIPHERUNICORN-E is a 64-bit block cipher algorithm with a 128-bit key.

CIPHERUNICORN-E has an option, which is called 'round number'. The round number specifies the number of internal iteration of data randomization.The round number is recommended to be at least 16.

l) Modes of operation Mode of operation as defined in ISO 8372 are applicable.

m) Other information A sample program is as follows:

ISO 9979/0019

```
/*********************************************
*                                            *
*     A Sample Program of CIPHERUNICORN-E    *
*                Coding: January 21 1998     *
*     Copyright (C) NEC Corporation 1998     *
*                                            *
*********************************************/
#include <stdlib.h>
#include <stdio.h>

#define ROUND 16
typedef unsigned int  uint;
typedef unsigned char uchar;

struct {
  uint  fk[ROUND][2];
  uint  sk[ROUND][2];
  uint  ik[(ROUND/2)+1][2];
} EK;
uchar sh[16][4] = {
  0,2,1,3 , 0,2,3,1 , 0,3,1,2 , 0,3,2,1,
  1,0,3,2 , 1,2,0,3 , 1,3,0,2 , 3,1,0,2,
  3,2,1,0 , 2,0,1,3 , 2,0,3,1 , 3,0,2,1,
  1,3,2,0 , 2,1,0,3 , 2,1,3,0 , 3,1,2,0
};
uchar S[4][256] = {
149,111,237,155, 21, 85,108, 76,236, 75,193, 84, 22,138, 89, 55,
 51,145, 13,153,148,163, 86, 59,204,175, 91,117,126, 70,144, 10,
248,146,201,  0, 97,208, 23,214,147,234, 66, 65,226, 57,210,224,
172, 40,154, 87,178,235,135,220,110,121, 96,  8,  9, 53,241,105,
143,169,182,139,112, 16,183, 67,233, 39,197, 74,166,218,231,242,
161,159,192, 37,177,228, 47,119, 14, 18,244, 56,  3,195,239,219,
 33,167, 26,180, 54, 61, 58,222,  4, 30,191, 34,107,249,142,150,
 95, 42,124, 25,232,181,120, 93,  5, 68,  6, 48,129, 41,104, 73,
188,165,212,160,250,141,123,216, 94,238, 81,202,  7,122,196, 17,
207,102,184,189,243, 72,206, 12,200,225,164,176,247,  1,  2,254,
 71,185,229,187,251,137, 69,168, 50, 24,171,173,158,221,127, 27,
252,114,152, 82,209, 38,203,128,215,213, 36,174,134,179, 90,118,
 80,246,253,125, 29, 44, 15,227, 98,205,255, 77,198,194,133,130,
 79,103, 78, 49, 19,140,109,211,223, 63, 64,151, 62,217,170, 83,
136, 45,115,199, 20, 46,190,240,132, 28,162,230,131,106, 32, 88,
157, 31, 43,156,113,186, 35,101, 52, 60, 11,100,116,245, 99, 92,

174,255,161,109,254, 40, 95, 67, 33,124,133, 58,224,238,129, 56,
137, 57,169, 87,221,220,163, 84, 14,239,171,138, 74,192, 66,104,
  8,250, 43,115,126, 88,212,103, 62, 82,143,  4,117,226, 28,155,
 65,156,139,183,235,125,217,116,111,237,157, 68,160,184,213,172,
170,132, 73,  2,  1,232, 92,249,136,106,175,  5,  9,140, 38,191,
 50,251, 85, 12, 27, 48, 46, 52,145, 78,168,159,100,188, 16,227,
 26,198,244,205,178, 72,142,162, 51,246,241,128,194,177,122, 20,
144, 49, 83,166,247,225, 11,  7,102,242,185, 18,150,165,121, 98,
 93,197, 70,151, 75,118,202,216,108,207, 15,112, 99, 35,101, 69,
 86, 61, 79,110, 13,218,149,  6,134, 29, 36,131,181,154,180,230,
 77,193,164, 17,211,  3,209,105, 94,206, 44, 19, 60,123, 10, 31,
130,195, 76,208, 54,252,219,203,199, 39,189, 80,167, 90, 32, 30,
233, 64,245,182,120,231,127, 47, 22,135, 55,114,234, 41, 21, 81,
173,223, 23,253,153, 25, 45,248, 97,179,186,119,200,146,187,210,
```

ISO 9979/0019

```
  0,228, 24,190,141,236, 63,201, 96,113,240,147,229, 91,107,214,
 89, 59,152,215,176,204,243,148, 42,158, 71, 34,222, 37,196, 53,

 37, 34,162,132,134,220, 91,143, 41, 45,229,247, 98,178, 68, 56,
212, 97, 70, 15, 58, 72,216,208, 14, 96,214,217,133,179, 28,154,
120,123, 83,100,235,  3,230,160,193,245,164,155,255,175, 79,148,
227,219, 23, 95,111, 11, 87,104,163,203,189, 29,156,173,211, 64,
157, 53,196, 89, 81,  4, 84, 16,192, 74, 13,181, 20,184, 57,183,
 90,119, 93,207, 38,131, 94, 60,116,  1,213,122,  5,101,144,117,
 75, 46,  8,172,170,152,231,210, 66, 54, 10,187,128,204, 12,102,
243,115,137,147,159,233, 59,221,253,112,165,198,105,222,234,153,
 43,201,121,180, 86,205,225,242,182, 55, 63,232,254, 44,  9, 21,
136, 65,114, 31, 40, 49,  0, 36,169, 22,249, 35, 62, 17,174,248,
158,151, 24, 50,176,108, 67,127,150, 18,  2,168,194,171,195,145,
 99, 25, 80,224, 33,200,197,118,161, 61,142, 77,190,209, 48,139,
238,206, 42,125,239,237, 52,223, 88,167, 26,130, 76,191,  7, 71,
215, 27,126,  6,251, 51,241,129,135,246,244,146, 32,177, 73, 82,
226,110, 78,186,240,141,166, 69,107, 85,103,149,250,109,202, 19,
113,140,138, 39,185,228,106, 47,252,199,188, 92,218, 30,236,124,

 24,252,144,121, 17, 42, 77,127,  2, 35,173, 21,129, 58,105,113,
112,229,185,189, 76,204,209, 87,  5, 96, 82, 99,133,140, 66, 64,
192,107,194,220, 16, 68,183,171,219, 51, 92, 13,152, 86,135,123,
 98,174,103,156,157, 59,145,155,158,  8,231,132, 83, 49, 23, 32,
 85, 69,251, 36,233,238,222,149, 37,248, 26, 18,125, 11,137,253,
 79, 52, 56, 95,241,187, 44,167,124,102,227,115,212,142,154, 93,
247,211, 33, 28, 67, 10,147,225,215,210,246,160,131, 73, 65, 57,
  1,182,180,199,207,126,216,224, 61, 81,202,196,146,188,119,128,
 50, 30, 91,161, 89, 12,195, 74,235,223,226,172,245,  7,218,159,
242,217,208, 38,163, 45, 39,  4, 62,136,104,179, 88,197,  6,  0,
141,190,243,214,109,162, 60,165,198,228,221,164,106,101,203,236,
143, 48,110, 80,176, 78,234,181, 97, 84, 20, 70, 29,168, 27, 72,
 71, 90,255, 19,254,114, 25,230, 47, 43,100,178, 40, 41,249,186,
150,205,184,201,139, 75, 54, 22, 63,244,108,175, 46,169,240,153,
151,116,122,232,166,117, 14, 94,111,206,237,177,200, 31,170,120,
213, 53,148, 15, 55,239,  3,191,134,250,193,  9,130,118,138, 34
};

void  UnicornScheduler(uint *);
void  SetIK(int,int,uint *);
void  SetSK(int,int,uint *);
void  SetFK(int,int,uint *);
void  UnicornEncode(uint *,uint *);
void  UnicornDecode(uint *,uint *);
void  L(uint *,uint *,uint,uint);
uint  F(int,uint);
uint  T(uint,int,uchar);
uint  Y(uint,int,int,int);
uint  K(uint,uchar,int);

void  main( void )
{
  uint  mkey[4];
  uint  p[2] , c[2];
  int   lp;

  mkey[0] = 0x00000000;
```

ISO 9979/0019

```
  mkey[1] = 0x00000000;
  mkey[2] = 0x00000000;
  mkey[3] = 0x00000000;

  UnicornScheduler(mkey);

  printf("Master Key = 0x%08x 0x%08x 0x%08x 0x%08x\n\n"
    ,mkey[0],mkey[1],mkey[2],mkey[3]);
  for(lp = 0 ; lp < ROUND; lp++)
    printf("EK.fk[%2d][0] = 0x%08x , EK.fk[%2d][1] = 0x%08x\n"
      ,lp,EK.fk[lp][0],lp,EK.fk[lp][1]);
  for(lp = 0 ; lp < ROUND; lp++)
    printf("EK.sk[%2d][0] = 0x%08x , EK.sk[%2d][1] = 0x%08x\n"
      ,lp,EK.sk[lp][0],lp,EK.sk[lp][1]);
  for(lp = 0 ; lp < (ROUND/2)+1 ; lp++)
    printf("EK.ik[%2d][0] = 0x%08x , EK.ik[%2d][1] = 0x%08x\n"
      ,lp,EK.ik[lp][0],lp,EK.ik[lp][1]);

  p[0] = 0x12345678;
  p[1] = 0x9abcdef0;

  UnicornEncode(p,c);

  printf("\nP = 0x%08x 0x%08x -> C = 0x%08x 0x%08x",p[0],p[1],c[0],c[1]);

  UnicornDecode(c,p);

  printf(" -> P = 0x%08x 0x%08x\n",p[0],p[1]);

  return;
}

void  UnicornScheduler( uint *mkey )
{
  uint  x[4] , xl , xr;
  int   lp , num = 0;
  int   ik = 0 , sk = 0 , fk = 0;

  x[0] = mkey[0];
  x[1] = mkey[1];
  xl = x[2] = mkey[2];
  xr = x[3] = mkey[3];

  for(lp = 0 ; lp < 4 ; lp++)
  {
    xl += T(xr,num%4,(uchar)(xr >> (24-(num%4)*8)));
    xr += T(xl,(num+1)%4,(uchar)(xl >> (24-((num+1)%4)*8)));
    xl = x[0] ^= xl;
    xr = x[1] ^= xr;
    xl += T(xr,(num+2)%4,(uchar)(xr >> (24-((num+2)%4)*8)));
    xr += T(xl,(num+3)%4,(uchar)(xl >> (24-((num+3)%4)*8)));
    xl = x[2] ^= xl;
    xr = x[3] ^= xr;
    num++;
  }

  for(lp = 0 ; lp < (ROUND/4)-1 ; lp++)
```

```
{
  SetIK(num++,ik++,x);
  SetSK(num++,sk,x);
  sk += 2;
  SetFK(num++,fk,x);
  fk += 2;
}

 {
  SetIK(num++,ik++,x);
  SetSK(num++,sk,x);
  sk += 2;

  xl = x[2] , xr = x[3];
  xl += EK.fk[fk][0] = T(xr,num%4,(uchar)(xr >> (24-(num%4)*8)));
  xr += EK.fk[fk+1][0] = T(xl,(num+1)%4,(uchar)(xl >> (24-((num+1)%4)*8)));
  xl = EK.fk[fk][1] = x[0] ^= xl;
  xr = EK.fk[fk+1][1] = x[1] ^= xr;
  xl += T(xr,(num+2)%4,(uchar)(xr >> (24-((num+2)%4)*8)));
  xr += EK.ik[ik][0] = T(xl,(num+3)%4,(uchar)(xl >> (24-((num+3)%4)*8)));
  num++;
  fk += 2;

  xl = EK.fk[fk][1] = x[2] ^= xl;
  xr = EK.fk[fk+1][1] = x[3] ^= xr;
  xl += T(xr,num%4,(uchar)(xr >> (24-(num%4)*8)));
  xr += EK.ik[ik][1] = T(xl,(num+1)%4,(uchar)(xl >> (24-((num+1)%4)*8)));
  xl = x[0] ^= xl;
  xr = x[1] ^= xr;
  xl += EK.sk[sk][1] = T(xr,(num+2)%4,(uchar)(xr >> (24-((num+2)%4)*8)));
  xr += EK.sk[sk+1][1] = T(xl,(num+3)%4,(uchar)(xl >> (24-((num+3)%4)*8)));
  xl = x[2] ^= xl;
  xr = x[3] ^= xr;
  num++;
  ik++;

  xl += EK.sk[sk][0] = T(xr,num%4,(uchar)(xr >> (24-(num%4)*8)));
  xr += EK.sk[sk+1][0] = T(xl,(num+1)%4,(uchar)(xl >> (24-((num+1)%4)*8)));
  xl = x[0] ^= xl;
  xr = x[1] ^= xr;
  xl += EK.fk[fk][0] = T(xr,(num+2)%4,(uchar)(xr >> (24-((num+2)%4)*8)));
  xr += EK.fk[fk+1][0] = T(xl,(num+3)%4,(uchar)(xl >> (24-((num+3)%4)*8)));
  xl = x[2] ^= xl;
  xr = x[3] ^= xr;
  num++;
  sk += 2;
  fk += 2;

  SetIK(num++,ik++,x);
 }

 for(lp = 0 ; lp < (ROUND/4)-1 ; lp++)
 {
   SetSK(num++,sk,x);
   sk += 2;
   SetFK(num++,fk,x);
   fk += 2;
```

```
      SetIK(num++,ik++,x);
    }

    return;
  }

  void  SetIK( int line , int n , uint *x)
  {
    uint  xl , xr;

    xl = x[2] , xr = x[3];
    xl += T(xr,line%4,(uchar)(xr >> (24-(line%4)*8)));
    xr += EK.ik[n][0] = T(xl,(line+1)%4,(uchar)(xl >> (24-((line+1)%4)*8)));
    xl = x[0] ^= xl;
    xr = x[1] ^= xr;
    xl += T(xr,(line+2)%4,(uchar)(xr >> (24-((line+2)%4)*8)));
    xr += EK.ik[n][1] = T(xl,(line+3)%4,(uchar)(xl >> (24-((line+3)%4)*8)));
    x[2] ^= xl;
    x[3] ^= xr;

    return;
  }

  void  SetSK( int line , int n , uint *x )
  {
    uint  xl , xr;

    xl = x[2] , xr = x[3];
    xl += EK.sk[n][1] = T(xr,line%4,(uchar)(xr >> (24-(line%4)*8)));
    xr += EK.sk[n+1][1] = T(xl,(line+1)%4,(uchar)(xl >> (24-((line+1)%4)*8)));
    xl = x[0] ^= xl;
    xr = x[1] ^= xr;
    xl += EK.sk[n][0] = T(xr,(line+2)%4,(uchar)(xr >> (24-((line+2)%4)*8)));
    xr += EK.sk[n+1][0] = T(xl,(line+3)%4,(uchar)(xl >> (24-((line+3)%4)*8)));
    x[2] ^= xl;
    x[3] ^= xr;

    return;
  }

  void  SetFK( int line , int n , uint *x )
  {
    uint  xl , xr;

    xl = x[2] , xr = x[3];
    xl += EK.fk[n][1] = T(xr,line%4,(uchar)(xr >> (24-(line%4)*8)));
    xr += EK.fk[n+1][1] = T(xl,(line+1)%4,(uchar)(xl >> (24-((line+1)%4)*8)));
    xl = x[0] ^= xl;
    xr = x[1] ^= xr;
    xl += EK.fk[n][0] = T(xr,(line+2)%4,(uchar)(xr >> (24-((line+2)%4)*8)));
    xr += EK.fk[n+1][0] = T(xl,(line+3)%4,(uchar)(xl >> (24-((line+3)%4)*8)));
    x[2] ^= xl;
    x[3] ^= xr;

    return;
  }
```

ISO 9979/0019

```
void  UnicornEncode( uint *p , uint *c )
{
  uint    xl = p[0] , xr = p[1];
  int   r;

  L(&xl,&xr,EK.ik[0][0],EK.ik[0][1]);

  for(r = 0 ; r < ROUND; r += 2)
  {
    xl ^= F(r,xr);
    xr ^= F(r+1,xl);
    L(&xl,&xr,EK.ik[r/2+1][0],EK.ik[r/2+1][1]);
  }

  c[0] = xl;
  c[1] = xr;

  return;
}

void    UnicornDecode( uint *c , uint *p )
{
  uint    xl = c[0] , xr = c[1];
  int   r;

  L(&xl,&xr,EK.ik[ROUND/2][0],EK.ik[ROUND/2][1]);

  for(r = ROUND-1 ; r > 0 ; r -= 2)
  {
    xr ^= F(r,xl);
    xl ^= F(r-1,xr);
    L(&xl,&xr,EK.ik[r/2][0],EK.ik[r/2][1]);
  }

  p[0] = xl;
  p[1] = xr;

  return;
}

void  L( uint *x0 , uint *x1 , uint k0 , uint k1 )
{
  uint  w0 = *x0 , w1 = *x1;

  *x0 = w0 ^ (w1 & k1) ^ (w0 & k0 & k1);
  *x1 = w1 ^ (w0 & k0) ^ (w1 & k1 & k0);

  return;
}

uint  F( int r , uint x )
{
  uint  w32 = x , k32;
  uchar wk1 , wk2 , wk3;

  w32 += EK.fk[r][0];
  k32  = EK.sk[r][0] + w32;
```

ISO 9979/0019

```
k32  = Y(k32,3,8,16);
k32  = T(k32,0,(uchar)(k32 >> 24));
k32 += EK.sk[r][1];
k32  = Y(k32,7,9,13);
k32  = T(k32,0,(uchar)(k32 >> 24));
k32  = T(k32,1,(uchar)(k32 >> 16));
wk1  = (uchar)(k32 >> 28);
wk2  = (uchar)k32;
wk3  = (uchar)(k32 >>  8);

w32  = T(w32,0,(uchar)(w32 >> 24));
w32  = T(w32,1,(uchar)(w32 >> 16));
w32  = T(w32,2,(uchar)(w32 >>  8));
w32  = T(w32,3,(uchar)w32);
w32 += EK.fk[r][1];
w32  = T(w32,sh[wk1][0],(uchar)(w32 >> (24-(sh[wk1][0]*8))));
w32  = T(w32,sh[wk1][1],(uchar)(w32 >> (24-(sh[wk1][1]*8))));
w32  = T(w32,sh[wk1][2],(uchar)(w32 >> (24-(sh[wk1][2]*8))));
w32  = T(w32,sh[wk1][3],(uchar)(w32 >> (24-(sh[wk1][3]*8))));
w32  = K(w32,wk2,24-(sh[wk1][0]*8));
w32  = T(w32,sh[wk1][0],(uchar)(w32 >> (24-(sh[wk1][0]*8))));
w32  = K(w32,wk3,24-(sh[wk1][1]*8));
w32  = T(w32,sh[wk1][1],(uchar)(w32 >> (24-(sh[wk1][1]*8))));

 return(w32);
}

uint  T( uint x , int n , uchar in )
{
  uchar wx[4];

  wx[(n+1)%4]  = S[0][in];
  wx[(n+2)%4]  = S[1][in];
  wx[(n+3)%4]  = S[2][in];
  wx[n]        = S[3][in] ^ in;

  return(x^(wx[0] << 24)^(wx[1] << 16)^(wx[2] << 8)^wx[3]);
}

uint  Y(uint x , int s1 , int s2 , int s3 )
{
  uint  wx = x;

  wx += wx << s1;
  wx += wx << s2;
  wx += wx << s3;

  return(wx);
}

uint  K( uint x , uchar k , int s )
{
  return(x^(k << s));
}
```