Masagus Krisna Ismaliansyah
13503115

KRIPTANALISIS PADA TIGER

Abstraksi

Makalah ini membahas tentang studi dan kriptanalisis pada fungsi Tiger. Tiger adalah sebuah fungsi hash yang dirancang oleh Ross Anderson dan Eli Biham pada tahun 1996. Fungsi ini dirancang sebagai fungsi hash alternatif semenjak ditemukannya kolisi pada fungsi hash MD4 dan turunannya, karena fungsi ini memiliki mekanisme yang berbeda dalam pembuatan nilai hashnya.
Kriptanalisis pada fungsi hash Tiger dirancang oleh John Kelsey dan Stefan Lucks. Mereka melakukan serangan kolisi terhadap fungsi hash ini. Tetapi, serangan kolisi yang baru berhasil hanyalah pada fungsi hash Tiger yang hanya memiliki 16 putaran. Sampai saat ini, belum diumumkan adanya serangan kolisi pada fungsi hash dengan putaran penuh.
Pada pembuatan makalah ini, penulis menggunakan referensi implementasi yang didapatkan dari situs resmi pembuat fungsi Tiger untuk melakukan perbandingan terhadap hasil nilai hash dari implementasi fugnsi tiger.

Kata Kunci : fungsi hash, kotak S, kolisi, kriptanalisis, Tiger.

I. Pendahuluan

Pada bab ini akan dijelaskan tentang latar belakang, tujuan, lingkup masalah, dan batasan masalah dalam pembuatan makalah ini.

I.1. Latar Belakang

Latar belakang pemilihan fungsi hash Tiger sebagai bahan makalah ini adalah untuk menambah wawasan tentang fungsi hash, selain fungsi-fungsi hash yang dipelajari pada saat kuliah. Selain itu, pemilihan fungsi hash ini untuk membandingkan kekuatannya dengan fungsi hash MD5 dan SHA.

I.2. Tujuan Pembuatan Makalah

Tujuan pembuatan makalah ini adalah untuk memperdalam pengetahuan tentang fungsi hash Tiger, baik dari segi algoritma, kekuatan dan keamanannya.

I.3. Lingkup Bahasan

Makalah ini melingkupi tentang teori singkat fungsi Tiger, algoritma fungsi Tiger, kriptanalisis pada fungsi Tiger, dan kekuatan dari fungsi Tiger.

I.4. Batasan Masalah

Makalah ini penulis batasi pada :

1. Algoritma fungsi hash Tiger, ditulis dalam bahasa algoritmik, dan diimplementasikan pada bahasa C.
2. Maksud kriptanalisis pada fungsi Tiger adalah serangan kolosi pada fungsi Tiger yang telah dimodifikasi yaitu hanya memiliki 16 putaran, dan menggunakan teknik yang dikembangkan oleh John Kelsey dam Stefan Lucks.

II. Teori Singkat Fungsi Tiger

Fungsi hash Tiger diciptakan pada tahun 1996 oleh dua orang ilmuan dari Universitas Cambridge yaitu Ross Anderson dan Eli Biham. Mereka merancang fungsi Tiger untuk dapat bekerja pada mesin 64 bit tetapi fungsi ini juga dapat bekerja pada mesin 32 bit.
Alasan utama pembuatan fungsi Tiger adalah menciptakan sebuah fungsih hash yang aman dan dapat bekerja pada prosesor 64 bit, mengingat banyak fungsi hash yang dirancang untuk prosesor 32 bit.

Selain itu, alasan lain dari pembuatan fungsi ini adalah karena ditemukannya kolisi pada fungsi hash MD4, padahal banyak fungsi hash yang berdasarkan fungsi MD4, termasuk keluarga SHA. Sehingga timbul spekulasi bahwa fungsi hash yang lain tinggal menunggu waktu untuk dapat dipecahkan. Karena itu, Ross Anderson dan Eli Biham ingin menciptakan fungsi hash yang tidak berdasarkan fungsi MD4. Walaupun demikian, fungsi Tiger tetap menggunakan prinsip penambahan panjang pesan dan pembagian pesan seperti pada fungsi MD4.

Seperti algoritma SHA, fungsi Tiger dapat menerima masukan berupa pesan dengan ukuran maksimum 264 bit, tetapi message digest yang dihasilkan memiliki panjang 192 bit, lebih panjang dari message digest yang dihasilkan oleh SHA maupun MD5. Selain itu, semua komputasi yang ada pada fungsi Tiger dalam 64 bit yang direpresentasi dengan little-endian.

Fungsi Tiger menggunakan kotak S untuk mengubah panjang bit dari 8 bit menjadi 64 bit. Kotak S pada fungsi tiger berupa larik yang bertipe word dengan panjang 64 bit dengan banyak anggota sebanyak 1024 buah. Selain menggunkaan kotak S, operasi yang ada pada fungsi tiger adalah operasi penambahan dan pengurangan 64 bit, perkalian 64 bit dengan konstanta yang kecil (5, 7, dan 9), operasi logika seperti XOR dan NOT, serta operasi pergeseran bit. Untuk lebih jelasnya, algoritma fungsi tiger akan dijabarkan pada bab III.

III. Algoritma Fungsi Tiger

Secara garis besar, langkah-langkah pembuatan nilai hash pada fungsi tiger tidak berbeda dengan fungsi SHA-0, adapun langkah-langkah tersebut adalah sebagai berikut :

1. Penambahan bit-bit pengganjal.
2. Penambahan nilai panjang pesan semula.
3. Inisialisasi penyangga MD.
4. Pengolahan pesan dalam blok berukuran 512 bit.

III.1. Penambahan Bit-Bit Pengganjal

Pesan ditambah dengan sejumlah bit pengganjal sedemikian sehingga panjang pesan (dalam satuan bit) kongruen dengan 448 modulo 512. Adapun penambahan bit-bit pengganjal adalah antara 1 sampai dengan 512 bit.

Adapun bit-bit pengganjal tersebut terdiri dari sebuah bit 1 diikuti dengan sisanya bit 0.

Contoh :

a. Pesan semula berukuran 40 bit, dengan nilai ABCDE, representasi bitnya adalah

```
01000001 01000010
01000011 01000100
01000101
```

maka jumlah bit pengganjal yang ditambahkan adalah sebesar 408 bit, sehingga panjang pesan menjadi 448 bit dengan representasi bit sebagai berikut :

```
01000001 01000010
01000011 01000100
01000101 00000001
00000000 00000000
00000000 00000000
00000000 00000000
00000000 00000000
00000000 00000000
00000000 00000000
00000000 00000000
00000000 00000000
00000000 00000000
00000000 00000000
00000000 00000000
00000000 00000000
00000000 00000000
00000000 00000000
00000000 00000000
00000000 00000000
00000000 00000000
00000000 00000000
00000000 00000000
00000000 00000000
00000000 00000000
00000000 00000000
00000000 00000000
00000000 00000000
```

b. Pesan semula berukuran 448 bit, maka jumlah bit pengganjal yang ditambahkan adalah sepanjang 512 bit. Sehingga, panjang pesan semula menjadi 960 bit.

III.2. Penambahan Nilai Panjang Pesan Semula

Pesan yang telah diberi bit-bit pengganjal selanjutnya ditambah lagi dengan 64 bit yang menyatakan panjang pesan semula. Sehingga, setelah ditambah dengan nilai panjang pesan semula, panjang pesan sekarang adalah kelipatan 512 bit.

Contoh :

a. Pesan semula berukuran 40 bit, dengan nilai ABCDE, setelah pesan ditambahkan dengan bit-bit pengganjal seperti pada contoh 1 butir III.1, pesan yang telah ditambahkan dengan bit-bit pengganjal tersebut ditambahkan lagi dengan nilai panjang pesan semula sebesar 64 bit, yang berisi nilai :

```
00000000 00000000
00000000 00000000
00000000 00000000
00000000 00101000
```

(00101000 merupakan representasi bit dari nilai 40), sehingga representasi bit dari pesan tersebut sekarang adalah :

```
01000001 01000010
01000011 01000100
01000101 00000001
00000000 00000000
00000000 00000000
00000000 00000000
00000000 00000000
00000000 00000000
00000000 00000000
00000000 00000000
00000000 00000000
00000000 00000000
00000000 00000000
00000000 00000000
00000000 00000000
00000000 00000000
00000000 00000000
00000000 00000000
00000000 00000000
00000000 00000000
00000000 00000000
00000000 00000000
00000000 00000000
00000000 00000000
00000000 00000000
00000000 00000000
00000000 00000000
00000000 00000000
00000000 00000000
00000000 00000000
00000000 00000000
00000000 00000000
00000000 00000000
00000000 00000000
00000000 00000000
00000000 00000000
00000000 00101000.
```

b. Pesan semula berukuran 448 bit, maka 64 bit terakhir yang berisi nilai panjang semula memiliki representasi bit :

```
00000000 00000000
00000000 00000000
00000000 00000000
00000001 11000000.
```

III.3. Inisialisasi Penyangga MD

Tiger hanya membutuhkan 3 buah penyangga yang masing-masing panjangnya 64 bit. Total panjang penyangga adalah 3 x 64 = 192 bit. Ketiga penyangga ini menampung hasil antara dan hasil akhir.

Ketiga penyangga ini diberi nama a, b, dan c. Setiap penyangga diinisialisasi dengan nilai-nilai (dalam notasi HEX) sebagai berikut :

a = 0x0123456789ABCDEF
b = 0xFEDCBA9876543210
c = 0xF096A5B4C3B2E187

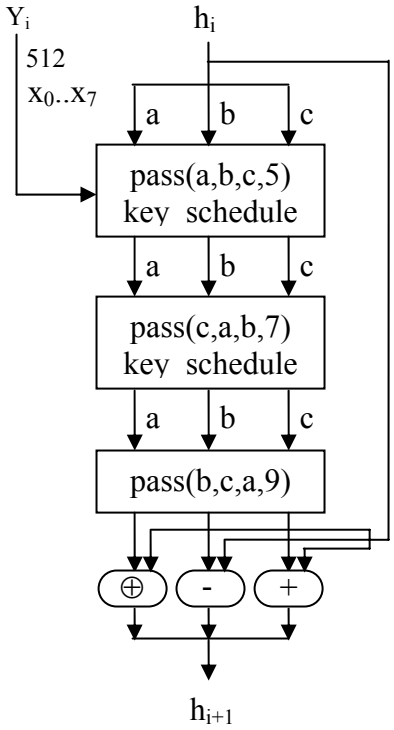
Ketiga penyangga ini merupakan nilai dari h_0.

III.4. Pengolahan Pesan dalam Blok Berukuran 512 bit.

Pesan dibagi menjadi L buah blok yang masing-masing panjangnya 512 bit (Y0 sampai YL-1). Setiap blok 512 bit diproses bersama dengan penyangga MD menjadi keluaran 192 bit, dan ini disebut nilai hi.

Blok pesan yang panjangnya 512 bit dibagi menjadi 8 buah word yang panjangnya 64 bit, x0, x1, x2, …, x7.

Selanjutnya, setiap blok pesan diproses seperti diperlihatkan pada Gambar 1.



Gambar 1. Pengolahan blok 512 bit

Keterangan :

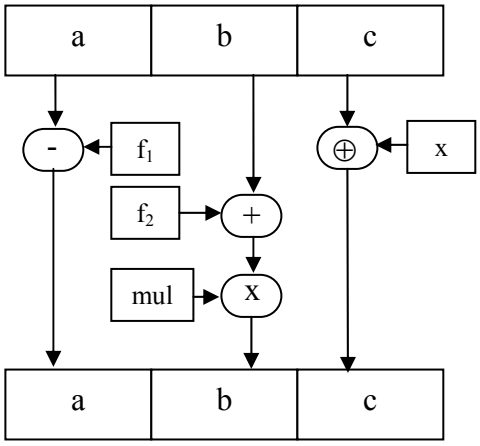
1. pass(a,b,c,mul) terdiri dari 8 buah putaran, yaitu :
   round(a,b,c,x0,mul)
   round(b,c,a,x1,mul)
   round(c,a,b,x2,mul)
   round(a,b,c,x3,mul)
   round(b,c,a,x4,mul)
   round(c,a,b,x5,mul)
   round(a,b,c,x6,mul)
   round(b,c,a,x7,mul)

2. round(a,b,c,x,mul) adalah
   c = c ⊕ x
   f1 = t1[c_0] ⊕ t2[c_2] ⊕ t3[c_4] ⊕ t4[c_6]
   f2 = t4[c_1] ⊕ t3[c_3] ⊕ t2[c_5] ⊕ t1[c_7]
   a = a – f1
   b = b + f2
   b = b * mul

   yang dalam hal ini,
   c_i = byte ke-i dari word c. Nilai c_i adalah antara 0..255.
   t_1 = anggota ke-0 s.d. ke-255 pada kotak S.
   t2 = anggota ke-256 s.d. ke-511 pada kotak S.
   t3 = anggota ke-512 s.d. ke-767 pada kotak S.
   t_4 = anggota ke-768 s.d. ke-1023 pada kotak S.

   Adapun isi dari kotak S akan dituliskan pada Bab IV bagian implementasi fungsi tiger.

   Proses round(a,b,c,x,mul) diilustrasikan pada Gambar 2.



Gambar 2. Operasi round(a,b,c,x,mul)

3. key_schedule adalah proses mengubah nilai $x_0..x_7$. Adapun proses tersebut adalah sebagai berikut :

   x0 = x0 – (x7 ⊕ 0xA5A5A5A5A5A5A5A5)
   x1 = x1 ⊕ x0
   x2 = x2 + x1
   x3 = x3 – (x2 ⊕ ((~x1)<<19))
   x4 = x4 ⊕ x3
   x5 = x5 + x4
   x6 = x6 – (x5 ⊕ ((~x4)>>23))
   x7 = x7 ⊕ x6
   x0 = x0 + x7
   x1 = x1 – (x0 ⊕ ((~x7)<<19))
   x2 = x2 ⊕ x1
   x3 = x3 + x2
   x4 = x4 – (x3 ⊕ ((~x2)>>23))
   x5 = x5 ⊕ x4
   x6 = x6 + x5
   x7 = x7 – (x6 ⊕ 0x0123456789ABCDEF)

yang dalam hal ini, $<<_x$ dan $>>_x$ menyatakan operasi pergeseran bit ke kiri atau ke kanan sebesar x bit.

III.5. Implementasi Fungsi Tiger

```
/* File : Tiger.c */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

typedef long long int word64;

/********** PROTOTYPE *********/
unsigned long int
getPanjangPesan(char * pesan);
//Mengembalikan nilai panjang
  dari pesan

char * getBagianPesan(char *
pesan, int x, int n);
//Mengambil bagian pesan
  sebanyak n dari posisi x

word64 getX(char * pesan, int
n);
//Mengambil pesan sebanyak 64
  bit dari posisi (n * 8)

int getC_i(word64 c, int n);
//Mengambil 1 byte dari word c
  dari posisi n

word64 stringToWord64(char * s);
//Mengubah string s menjadi
  representasi word 64 bit

char * wordToString(word64 w);
//Menghasilkan nilai string dari
  word w

void inisiasiX(char * pesan);
//I.S. sembarang
//F.S. nilai x0..x7 diinisiasi
  dengan nilai-nilai dari pesan

void inisiasiMD(char * newMD);
//I.S. sembarang
//F.S. nilai MD menjadi newMD

char * paddingPesan(char *
pesan);
//Menghasilkan pesan yang telah
  ditambahkan bit-bit pengganjal
  dan panjang pesan semula
  string yang dihasilkan
  totalnya 128 byte

void round(word64 * a, word64 *
b, word64 * c, word64 x, int
mul);
//I.S. a,b,c,x terdefinisi
//F.S. nilai a,b,c,x merupakan
  hasil satu loop dari fungsi
  tiger

void pass(word64 * a, word64 *
b, word64 * c, int mul);
//I.S. a,b,c terdefinisi
//F.S. melakukan 8 buah loop,
  fungsi round

void key_schedule();
//I.S. x0..x7 telah diinisiasi
//F.S. mengubah nilai x0..x7

char * hash(char * pesan);
//Menerima masukan bagianPesan
  yang panjang karakternya 512
  bit dan mengolah masukan
  tersebut menjadi nilai hash
  sementara

char * tiger(char * pesan);
//Menerima masukan pesan berupa
  String dan mengolah masukan
  tersebut sehingga menjadi
  nilai hash tiger

void tulisHasil(char * hasil);
//I.S. hasil terdefinisi
//F.S. menuliskan hasil ke layar

/******** KAMUS GLOBAL ********/
//Penyangga MD
word64 MD[3];

//Variabel x0..x7
word64 x[8];

//Kotak S
extern word64 kotakS[1024];

/********** BODY **********/
unsigned long int
getPanjangPesan(char * pesan)
//Mengembalikan nilai panjang
  dari pesan
{
   /* Kamus */
   unsigned long int panjang;
```

```
    /* Algoritma */
    Panjang = 0;
    while(pesan[panjang] != '\0')
    {
       panjang++;
    };
    return panjang;
}

char * getBagianPesan(char *
pesan, int x, int n)
//Mengambil bagian pesan
  sebanyak n dari posisi x
{
    /* Kamus */
    char * hasil;
    int i;

    /* Algoritma */
    hasil = (char *) malloc
      ((n + 1) * sizeof(char));

    for(i = 0; i < n; i++)
    {
          hasil[i] =
              pesan[x + i];
    }

    hasil[i] = '\0';
    return hasil;
}

word64 getX(char * pesan, int n)
//Mengambil pesan sebanyak 64
  bit dari posisi (n * 8)
{
    return(stringToWord64(
      getBagianPesan(pesan,
        n * 8, 8)));
}

int getC_i(word64 c, int n)
//Mengambil 1 byte dari word c
  dari posisi n
{
    /* Kamus */

    /* Algoritma */
    return ((c >> (n * 8)) &
       0xFF);
}

word64 stringToWord64(char * s)
//Mengubah string s menjadi
  representasi word 64 bit
{
    return (*(word64 *) s);
}

char * wordToString(word64 w)
//Menghasilkan nilai string dari
  word w
{
    /* Kamus */
    char * hasil;
    int i;

    /* Algoritma */
    hasil = (char *) malloc (9 *
       sizeof(char));

    for(i = 0; i < 8; i++)
    {
          hasil[i] = (w >>
             (i * 8)) & 0xFF;
    }

    hasil[8] = '\0';
    return hasil;
}

void inisiasiX(char * pesan)
//I.S. sembarang
//F.S. nilai x0..x7 diinisiasi
  dengan nilai-nilai dari pesan
{
    /* Kamus */
    int i;

    /* Algoritma */
    for (i = 0; i < 8; i++)
    {
        x[i] = getX(pesan, i);
    }
}

void inisiasiMD(char * newMD)
//I.S. sembarang
//F.S. nilai MD menjadi newMD
{
    /* Kamus Lokal */

    /* Algoritma */
    MD[0] = getX(newMD, 0);
    MD[1] = getX(newMD, 1);
    MD[2] = getX(newMD, 2);
}

char * paddingPesan(char *
pesan)
//Menghasilkan pesan yang telah
  ditambahkan bit-bit pengganjal
  dan panjang pesan semula
  string yang dihasilkan
```

```
  totalnya 128 byte
{
   /* Kamus Lokal */
   char * hasil;
   word64 panjangPesan;
   unsigned int temp;
   int i, j;

   /* Algoritma */
   hasil = (char *) malloc
      (129 * sizeof(char));

   panjangPesan =
      getPanjangPesan(pesan);

   strcpy(hasil,
      getBagianPesan(pesan,
      (panjangPesan / 64) * 64 ,
      panjangPesan % 64));

   strcat(hasil,
      wordToString(0x01));

   if(getPanjangPesan(hasil) %
      64 > 56)

   {
      for(i =
       (getPanjangPesan(hasil));
       i < 120; i++)
      {
         hasil[i] = 0x00;
      }
      hasil[128] = '\0';
   }
   else
   {
      for(i =
       (getPanjangPesan(hasil));
       i < 56; i++)
      {
         hasil[i] = 0x00;
      }
      hasil[64] = '\0';
   }

   for(j = 0; j < 8; j++)
   {
      temp = ((panjangPesan * 8)
         >> (j * 8)) & 0xFF;
      hasil[j + i] = temp;
   }
   return hasil;
}
```

```
void round(word64 * a, word64 *
b, word64 * c, word64 x, int
mul)
//I.S. a,b,c,x terdefinisi
//F.S. nilai a,b,c,x merupakan
  hasil satu loop dari fungsi
  tiger
{
   (*c) ^= x;
   (*a) -=
     kotakS[getC_i((*c), 0)] ^
     kotakS[256 +
        getC_i((*c), 2)] ^
     kotakS[512 +
        getC_i((*c), 4)] ^
     kotakS[768 +
        getC_i((*c), 6)];

   (*b) +=
     kotakS[768 +
        getC_i((*c), 1)] ^
     kotakS[512 +
        getC_i((*c), 3)] ^
     kotakS[256 +
        getC_i((*c), 5)] ^
     kotakS[getC_i((*c), 7)];

   (*b) *= mul;
}

void pass(word64 * a, word64 *
b, word64 * c, int mul)
//I.S. a,b,c terdefinisi
//F.S. melakukan 8 buah loop,
  fungsi round
{
   round(a,b,c,x[0],mul);
   round(b,c,a,x[1],mul);
   round(c,a,b,x[2],mul);
   round(a,b,c,x[3],mul);
   round(b,c,a,x[4],mul);
   round(c,a,b,x[5],mul);
   round(a,b,c,x[6],mul);
   round(b,c,a,x[7],mul);
}

void key_schedule()
//I.S. x0..x7 telah diinisiasi
//F.S. mengubah nilai x0..x7
{
   x[0] -= (x[7] ^
        0xA5A5A5A5A5A5A5A5LL);

   x[1] ^= x[0];

   x[2] += x[1];
```

```
    x[3] -= (x[2] ^
         (~x[1] << 19));

    x[4] ^= x[3];

    x[5] += x[4];

    x[6] -= (x[5] ^
         (~x[4] >> 23));

    x[7] ^= x[6];

    x[0] += x[7];

    x[1] -= (x[0] ^
         (~x[7] << 19));

    x[2] ^= x[1];

    x[3] += x[2];

    x[4] -= (x[3] ^
         (~x[2] >> 23));

    x[5] ^= x[4];

    x[6] += x[5];

    x[7] -= (x[6] ^
         0x0123456789ABCDEFLL);
}

char * hash(char * pesan)
//Menerima masukan bagianPesan
  yang panjang karakternya 512
  bit dan mengolah masukan
  tersebut menjadi nilai hash
  sementara
{
    /* Kamus */
    word64 a, b, c;
    char * hasil;

    /* Algoritma */
    hasil = (char *) malloc
         (25 * sizeof(char));

    a = MD[0];
    b = MD[1];
    c = MD[2];

    inisiasiX(pesan);

    pass(&a, &b, &c, 5);

    key_schedule();

    pass(&c, &a, &b, 7);

    key_schedule();

    pass(&b, &c, &a, 9);

    a ^= MD[0];
    b -= MD[1];
    c += MD[2];

    strcpy(hasil,
         wordToString(a));

    strcat(hasil,
         wordToString(b));

    strcat(hasil,
         wordToString(c));

    return hasil;
}

char * tiger(char * pesan)
//Menerima masukan pesan berupa
  string dan mengolah masukan
  tersebut sehingga menjadi
  nilai hash tiger
{
    /* Kamus */
    char * hasil;
    char * pesanTemp;
    char * __MD;
    word64 panjangPesan;
    int i;

    /* Algoritma */
    __MD = (char *) malloc
       (25 * sizeof(char));

    strcpy(__MD,
       wordToString(
          0x0123456789ABCDEFLL));

    strcat(__MD,
       wordToString(
          0xFEDCBA9876543210LL));

    strcat(__MD,
       wordToString(
          0xF096A5B4C3B2E187LL));

    inisiasiMD(__MD);

    hasil = (char *) malloc
        (25 * sizeof(char));

    panjangPesan =
```

```
        getPanjangPesan(pesan);

    //proses tanpa padding, ini
      berjalan bila pesan > 64
      byte

    for(i = 0; i <
       (panjangPesan / 64); i++)
    {
       strcpy(hasil,
          hash(getBagianPesan(
          pesan, (i * 64), 64)));

       inisiasiMD(hasil);
    }

    //padding pesan
    pesanTemp = (char *) malloc
        (129 * sizeof(char));
    pesanTemp =
        paddingPesan(pesan);

    i = 0;
    if(getPanjangPesan(
        pesanTemp) > 64)

    //bila padding pesan
      menghasilkan > 64 byte
    {
       strcpy(hasil,
          hash(getBagianPesan(
          pesanTemp, i * 64,
          64)));
       i++;
       inisiasiMD(hasil);
    };

    //proses terakhir
    strcpy(hasil,
       hash(getBagianPesan(
       pesanTemp, i * 64, 64)));

    return hasil;
}

void tulisHasil(char * hasil)
{
    /* Kamus */

    /* Algoritma */
    printf("nilai hash = %08X",
       stringToWord64(
       getBagianPesan(hasil, 0,
       8)) >> 32);

    printf("%08X  ",
       stringToWord64(
       getBagianPesan(hasil, 0,
       8)));

    printf("%08X",
       stringToWord64(
       getBagianPesan(hasil, 8,
       8)) >> 32);

    printf("%08X  ",
       stringToWord64(
       getBagianPesan(hasil, 8,
       8)));

    printf("%08X",
       stringToWord64(
       getBagianPesan(hasil, 16,
       8)) >> 32);

    printf("%08X\n",
       stringToWord64(
       getBagianPesan(hasil, 16,
       8)));
}
```

IV. Kriptanalisis Pada Tiger

Serangan yang dirancang oleh John Kelsey dan Stefan Lucks untuk menemukan kolisi pada fungsi Tiger menggunakan pendekatan serangan diferensial yang dibagi menjadi 3 bagian utama. Secara keseluruhan, serangan tersebut menukar antara turunan XOR dengan turunan penjumlahan. Hal ini dapat dilakukan karena secara umum, menukar dua buah turunan akan menghasilkan kemungkinan probabilistik tidak 0. Contohnya penjumlahn dengan 1 dapat kita ubah menjadi XOR dengan 1, dengan probabilistik adalah ½.

Walaupun dengan mengubah suatu turunan ke turunan lainnya menghasilkan probabilistik tidak 0, tetapi ada beberapa nilai, yang probablistiknya adalah 1.

Contohnya, jika $X - Y = 2^i$, maka $P[X \oplus Y] = ½$. Tetapi, ada pengecualian untuk $i = 63$, dimana $P[X \oplus Y] = 1$, karena nilai x dan y yang menghasilkan 2^{63} pada operasi 64 bit, hanyalah 1 buah yaitu $X=2^{63}$ dan $Y = 0$ sehingga nilai dari $P[X \oplus Y] = 1$.

Untuk selanjutnya, nilai i yang akan kita gunakan pada serangan kolisi untuk fungsi Tiger adalah $i = 63$.

Adapun konvensi yang terdapat pada isi dari subbab ini adalah sebagai berikut :

1. I = 2^{63}, karena menukar hasil penjumlahan yang nilainya I dengan hasil XOR yang bernilai I nilai probabilistiknya adalah 1.
2. Kita mulai menghitung putara dari 0, dan kita menulis Xi untuk pesan masukan pada putaran ke-I, dan Ai, Bi, Ci, untuk hasil putaran ke-i.
3. $\Delta^{+}(W) = W\text{-}W^{*} \bmod 264$ untuk turunan jumlah
4. $\Delta^{\oplus}(W) = W \oplus W'$ untuk turunan lainnya.

Pada awal bab telah disebutkan bahwa serangan kolisi pada fungsi Tiger terbagi menjadi tiga bagian, yaitu :

1. Karakteristik differensial (I,I,I,I,0,0,0,0) → (I,I,0,0,0,0,0,0) pada proses key schedule.
2. Karakteristik differensial (I,I,0) → (0,0,0) pada putaran 6-9. Hal ini disebabkan oleh word pada pesan saat putaran ke 10-15 tidak berubah, hal inilah yang menyebabkan kolisi pada 16 putaran.
3. Pengubahan pesan yang menghasilkan turunan (I,I,0) setelah putaran ke-6.

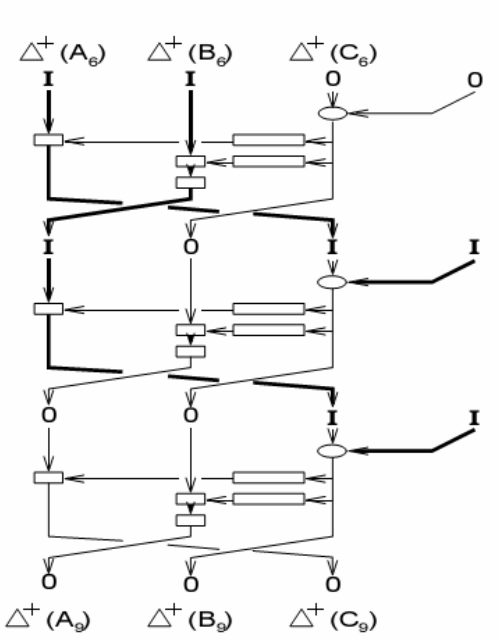
IV.1. Turunan Key Schedule

Misalkan pola turunan pada pesan adalah (I,I,I,I,0,0,0,0), operasi pertama pada key schedule mengubah pola ini menjadi (I,0,I,0,0,0,0,0), dan menjadi polah (I,I,0,0,0,0,0,0) pada operasi ke dua. Pola turunan inilah yang dipakai pada serangan kolisi fungsi Tiger, karena pola ini menghasilkan nilai probabilistik 1 dan mencakup words pada pesan yang telah diperpanjang untuk putaran 0-15.

Pesan-pesan yang bersesuaian hanya akan berbeda pada empat word pertama berdasar bit-bit urutan tinggi. Word pada pesan yang diganjal akan berbeda pada putaran 8-9, dan hanya pada bit-bit urutan tinggi. Pesan yang diganjal, pada word 10-15 tidak akan memiliki perbedaan. Ini berarti, jika hasil operasi perbandingan dua buah pesan setelah putaran ke-9 sama, maka hasil tersebut tetap sama sampai putaran ke-15, hal inilah yang menyebabkan adalah kolisi pada fungsi Tiger 16 putaran.

IV.2. Turunan fungsi putaran

Dari karakteristik turunan key schedule di atas, kita dapat menentukan karakteristik turunan untuk fungsi putaran dari putaran 6 sampai pada akhir putaran 9. Berasal dari pola (I,I,0) menjadi (0,0,0) dengan mengabikan turunan pada putaran 8-9. Words pada pesan yang telah diperpanjang dari putaran 10-15 tidak memiliki perbedaan, dengan demikian kolisi setelah putaran 9 menjadi kolisi untuk Tiger dengan 16 putaran. Gambar 3 berikut menunjukkan karakteristik tersebut :



Gambar 3. Probabilistik karakteristik putaran 6-9

IV.3. Pengubahan Pesan

Kesulitan utama pada serangan ini adalah pada saat pengubahan pesan. Ingat kembali bahwa target turunan kita pada saat putaran 6 adalah

$$\Delta^{+}(A_6) = I,\ \Delta^{+}(B_6) = I,\ \Delta^{+}(C_6) = 0.$$

Terlepas dari pada pemilihan words pesan, kita tahu nilai Δ^+ (C_5) dan $\Delta^+(C_4)$. Sejak $\Delta^+(X_6) = 0$, kita perlu $\Delta^+(C_5) = \Delta^+(B_6) = I$. Dengan cara yang sama, kita tahu bahwa hubungan $\Delta^+(C_4) = I + \Delta^+(odd(B_6))$.

IV.4. Modifikasi Pesan Lokal dengan Pendekatan Meeting in the Middle(MITM)

Asumsikan kita mengetahui masukan $(A_{i-1}, B_{i-1}, C_{i-1})$, dan $(A'_{i-1}, B'_{i-1}, C'_{i-1})$ dan beberapa nilai turunan words pesan X_i dan X_{i+1}. Kita ingin memaksa supaya turunan penjumlahan $\delta^* = \Delta^+(C_{i+1}) = C_{i+1} - C_{i+1}$. Seperti diperlihatkan pada gambar 4, turunan $\Delta^+(C_{i+1})$ bergantung dari $\Delta^+(B_{i-1})$, hasil dari penjumlahan dengan turunan dari fungsi odd pada putaran ke-i, dan hasil penjumlahan dengan turunan dari fungsi even pada putaran i+i.

IV.5. Modifikasi Pesan Masukan

Pertama-tama, lihatlah fungsi even pada Tiger. Setelah komputasi $B_{i+1} = C_i \oplus X_{i+1}$, dijalankan fungsi

$$even(B_{i+1}) = T_1(B_{i+1}[0]) \oplus T_2(B_{i+1}[2]) \oplus T_3(B_{i+1}[4]) \oplus T_4(B_{i+1}[6]).$$

Untuk setiap turunan XOR bukan 0 antara words B_{i+1} dan B'_{i+1}, kita mengharapkan 2^{32} hasil turunan penjumlahan yang berbeda dari bentuk $\delta_{even} = even(B_{i+1}) - even(B'_{i+1})$. Dengan cara yang sama, ketika kita melihat fungsi odd pada Tiger

$$odd(B_i) = T_1(B_{i+1}[7]) \oplus T_2(B_{i+1}[5]) \oplus T_3(B_{i+1}[3]) \oplus T_4(B_{i+1}[1]).$$

Kita juga mengharapkan 2^{32} hasil turunan penjumlahan yang berbeda dari bentuk $\delta_{odd} = odd(B_i) - odd(B'_i)$.

Jadi, jika turunan pada Bi+1[even] dan Bi[odd] keduanya bukan 0, kita dapat menerapkan pendekatan meet in the middle untuk memaksa supaya

$$\delta even = (\Delta^+(B_{i-1}) + \delta_{odd}) * (const) - \delta^*$$

Atau dapat juga ditulis

$$\delta_{odd} = (\delta even + \delta^*) / (const) - \Delta^+(B_{i-1}) \quad ..^{(1)}$$

Teknik ini melakukan evaluasi sebanyak 2^{32} untuk setiap fungsi even dan fungsi odd, yang equivalen dengan evaluasi sebanyak 2^{32} pada fungsi perbandingan, dan tentu saja, ruangan penyimpanan yang dibutuhkan sebanyak 2^{32} unit.

Kita memperkirakan untuk setiap $\Delta^+(B_{i-1})$ dan δ^* yang diberikan, pendekatan meet in the middle memberikan tingkat keberhasilan mendekati ½. Pada skenario serangan, jika diperlukan kita akan mengulangi pendekatan dengan $\Delta^+(B_{i-1})$ atau δ^* yang lain.

Asumsikan X_i[even] telah ditentukan dan pendekatan meet in the middle menghasilkan δ_{even} dan δ_{odd} yang memuaskan terhadap persamaan (1); δ_{even} mendefinisikan B_{i+i}[even] dan δodd mendefinisikan B_i[odd].

Akhirnya kita dapat melakukan komputasi 64 bit pesan lokal :

$$X_i[odd] = C_{i-1}[odd] \oplus B_i[odd]$$
$$X_{i+1}[even] = C_i[even] \oplus B_{i+1}[even]$$

IV.6. Pengubahan Pesan untuk Mendapatkan Turunan XOR

Pada langkah 3 dari serangan yang akan dijelaskan di bawah, kita membutuhkan turunan XOR yang spesifik pada C_3. Walaupun demikian, teknik meet in the middle di atas hanya bekerja untuk turunan penjumlahan. Solusi untuk masalah ini adalah dengan melakukan komputasi brute force, yaitu :

Untuk sebuah turunan XOR yang diinginkan dari bobok Hamming, k, secara sederhana kita melakukan pencarian meet in the middle untuk setiap turunan penjumlahan yang dihasilkan dari turunan XOR, sampai selesai atau ditemukannya sebuah turunan penjumlahan yang sama dengan turunan XOR yang diinginkan.

Sebuah turunan penjumlahan yang dapat memberikan turunan XOR k-bit memiliki probabilitas 2-k. Ini berarti, kita memerlukan percobaan sebanyak 2k turunan penjumlahan yang konsisten dengan turunan k-bit sebelum menemukan turunan yang sesuai. Waktu yang dibutuhkan meet in the middle untuk menemukan turunan yang cocok sekitar ½ waktu keseluruhan, kita membutuhkan total 2^{k-1} langkah MIM. Walaupun demikian, kita dapat mengoptimalkan ini

dengan hanya melakukan satu sisi dari pencarian MIM untuk setiap target turunan penjumlahan. Banyaknya kalkulasi maksimal adalah 2^{28+k}.

IV.7. Pengubahan Pesan dengan Batasan

Dua dari langkah-langkah middle in the match pada serangan di bawah, berada pada batasan-batasan dari pemilihan bit-bit pesan. Batasan tersebut berasal dari perpindahan antara turunan XOR pada C_3 dan turunan penjumlahan pada B_4. Semenjak turunan XOR memiliki k-bit yang aktif, dan turunan jumlah hanya konsisten dengan satu set nilai dari bit-bit tersebut, sehingga k-bit dari word pesan X_4 dibatasi.

Pengubahan pesan yang dibatasi, relatif sederhana, daripada melakukan pencarian 232 yang mungkin untuk turunan jumlah, kita mencari dalam jumlah kemungkinan yang lebih sedikit, dengan batasan bit pada pesan. Untuk mendapatkan kebutuhan nilainya. Untuk penyederhanaan, kita mengasumsikan bahwa k/2 bit dibatasi pada byte genap dan k/2 pada byte ganjil. Walaupun demikian, tingkat keberhasilan menurun. Dengan hanya memiliki 228 pilihan dari satu sisi, dan 232 dari sisi lainnya, kita mengharapkan kemungkinan 2-4 untuk mendapatkan turunan yang sama. Jadi, kita hanya membutuhkan pengulangan pencarian MIM dengan 4 bit batasan sekitar 16 kali.

IV.8. Skenario Global Pengubahan Pesan (Serangan Kolisi)

0 Lakukan satu kali komputasi untuk menemukan sebuah turunan penjumlahan L dengan sebuah bobot Hamming yang kecil berkoresponden dengan turunan XOR yang dapat kita buang dengan pilihan kita pada byte genap dari X_6. Perhitungan ini membutuhkan 2^{27} equivalen fungsi hash Tiger 16 putaran, dan kita mengharapkan hasil tersebut memenuhi sebuah turunan penjumlahan yang konsisiten dengan 8-bit turunan XOR.

1 Pilih X_0 dan X_1[even] untuk menjamin bahwa C_0 dan C_1 memiliki turunan yang berguna untuk proses selanjutnya. Catat bahwa pada akhir langkah ini, ke akan mengetahui nilai $\Delta^{\oplus}(C_1)$ dan $\Delta^{\oplus}(C_2)$. Kita menggunakan hasil ini untuk langkah selanjutnya.

2 Pilih X_1[odd] dan X_2[even] untuk menjamin bahwa C_2 memiliki turunan yang berguna. Catat bahwan pada akhir langkah ini, kita mengetahui nilai $\Delta^{\oplus}(C_2)$. Kita menggunakan turunan XOR ini untuk langkah selanjutnya.

3 Lakukan langkah pengubahan pesan untuk mendapatkan turunan XOR di $\Delta^{\oplus}$ C_3 yang konsisten dengan turunan penjumlahan L. Ini telah dijelaskan di atas. Pekerjaan yang dilakukan equivalen dengan 2^{38} fungsi hash Tiger-16, dan kita menentukan nilai X_2^{odd}, X_3^{even}.

4 Lakukan langkah pendekatan meet in the middle yang dibatasi, pilih X_3[odd], X_4[even] untuk undapatkan ΔC_4 = I. Kita mengharapkan bahwa disana adalah 4 bit yang dibatasi, yang berarti kita membutuhkan paling banyak 16 kali percobaan untuk mendapatkan hasil yang diinginkan. Setiap kegagalan menyebabkan harus kembali ke langkah 2. Yang berarti waktu yang dibutuhkan adalah $2^4 2^{36}=2^{40}$ yang ekivalen dengan fungsi hash Tiger 16 putaran.

5 Lakukan langkah pendekatan meet in the middle yang dibatasi, pilih X_4[odd],X_5[even] untuk memaksa ΔC_5 = I. Seperti sebelumnya., kita mengharapkan ini dibatasi oleh 4 bit, sehingga kita butuh pengunalan 16 kali. Setiap kegagalan mengharuskan kembali ke langkah 2. Langkah ini membutuhkan $2^4 2^{40} = 2^{44}$ komputasi.

6 Berikan nilai untuk C_5, kita menggunakan hasil dari langkah 0 untuk menentukan nilai dari byte genap dari X_6. Langkah ini tidak pernah gagal.

Hasil dari langkah terakhir adalah I,0,I, yang dengan probabilitas 1, dapat dibuang dengan menggunkaan karakteristik key schedule, yang menyebabkan kolisi 16 putaran.

V. Laporan Teknik Kriptanalisis pada Tiger

Bagian ini hanya berisi pengujian terhadap implementasi fungsi .

V.1. Pengujian Terhadap Implementasi Fungsi Tiger

Pada bagian pengujian implementasi ini, digunakan script program sebagai berikut:

```
/* File : mTiger.c */

int main()
{
   final = (char *) malloc
      (25 * sizeof(char));

   //uji 1
   strcpy(final, tiger(""));
   tulisHasil(final);

   //uji 2
   strcpy(final, tiger("abc"));
   tulisHasil(final);

   //uji 3
   strcpy(final,
     tiger("ABCDEFGHIJKLMNOPQRS
       TUVWXYZabcdefghijklmnopqr
       stuvwxyz0123456789+-"));

   tulisHasil(final);
}
```

Hasil fungsi tiger :

a. uji 1:

```
24F0130C63AC9332
16166E76B1BB925F
F373DE2D49584E7A
```

seharusnya :

```
24F0130C63AC9332
16166E76B1BB925F
F373DE2D49584E7A
```

b. uji 2 :

```
B76950EA912118F1
E2FA6B8B2E0C7446
606BED6F943FBA6E
```

seharusnya :

```
F258C1E88414AB2A
527AB541FFC5B8BF
935F7B951C132951
```

c. uji 3 :

```
56DD41A325EFAFC2
D713B2744B38B597
429A11B5FDBA0AC6
```

seharusnya :

```
87FB2A9083851CF7
470D2CF810E6DF9E
B586445034A5A386
```

Dari ketiga hasil diatas, terdapat perbedaan antara hasil 2 dan 3. Hal ini dikarenakan, pada fungsi Tiger terdapat perbedaan mekanisme pada penambahan pesan dengan MD4, baik bit-bit pengganjal ataupun penambahan panjang pesan semula. Selain itu, representasi bit pada fungsi Tiger adalah representasi little-endian.

Hasil uji 1 menunjukkan nilai yang sama dikarenakan masukan stringnya merupakan string kosong. Sehingga tidak terpengaruh dengan perbedaan penambahan panjang pesan semula.

Adapun perbedaan mekanisme fungsi Tiger dengan MD4 adalah sebagai berikut :

Pembeda	Tiger	MD4
Bit pengganjal pertama	0x01	0x80
Penambahan panjang pesan semula	Little-endian	Big-endian
Arsitektur	64 bit	32 bit

VI. Kesimpulan

1. Dari hasil pengujian implementasi Tiger, dapat disimpulkan bahwa implementasi fungsi Tiger pada makalah ini masih butuh perbaikan pada implementasi penambahan pesan.
2. Waktu total yang dibutuhkan untuk melakukan serangan kolisi pada fungsi hash Tiger 16 putaran adalah 2^{40}.

Daftar Pustaka

1. Munir, Rinaldi. (2004). Bahan Kuliah IF5054 Kriptografi. Departemen Teknik Informatika. Institut Teknologi Bandung.
2. http://www.cs.technion.ac.il/%7Ebiham/Reports/Tiger/
3. http://th.informatik.uni-mannheim.de/people/lucks/

Lampiran

a. Kotak S Pada Fungsi Tiger

```
// File : kotakS.c
// Kotak S
typedef unsigned long long int word64;
word64 kotakS[1024] = \
{
    0x02AAB17CF7E90C5ELL   /*    0 */,    0xAC424B03E243A8ECLL   /*    1 */,
    0x72CD5BE30DD5FCD3LL   /*    2 */,    0x6D019B93F6F97F3ALL   /*    3 */,
    0xCD9978FFD21F9193LL   /*    4 */,    0x7573A1C9708029E2LL   /*    5 */,
    0xB164326B922A83C3LL   /*    6 */,    0x46883EEE04915870LL   /*    7 */,
    0xEAACE3057103ECE6LL   /*    8 */,    0xC54169B808A3535CLL   /*    9 */,
    0x4CE754918DDEC47CLL   /*   10 */,    0x0AA2F4DFDC0DF40CLL   /*   11 */,
    0x10B76F18A74DBEFALL   /*   12 */,    0xC6CCB6235AD1AB6ALL   /*   13 */,
    0x13726121572FE2FFLL   /*   14 */,    0x1A488C6F199D921ELL   /*   15 */,
    0x4BC9F9F4DA0007CALL   /*   16 */,    0x26F5E6F6E85241C7LL   /*   17 */,
    0x859079DBEA5947B6LL   /*   18 */,    0x4F1885C5C99E8C92LL   /*   19 */,
    0xD78E761EA96F864BLL   /*   20 */,    0x8E36428C52B5C17DLL   /*   21 */,
    0x69CF6827373063C1LL   /*   22 */,    0xB607C93D9BB4C56ELL   /*   23 */,
    0x7D820E760E76B5EALL   /*   24 */,    0x645C9CC6F07FDC42LL   /*   25 */,
    0xBF38A078243342E0LL   /*   26 */,    0x5F6B343C9D2E7D04LL   /*   27 */,
    0xF2C28AEB600B0EC6LL   /*   28 */,    0x6C0ED85F7254BCACLL   /*   29 */,
    0x71592281A4DB4FE5LL   /*   30 */,    0x1967FA69CE0FED9FLL   /*   31 */,
    0xFD5293F8B96545DBLL   /*   32 */,    0xC879E9D7F2A7600BLL   /*   33 */,
    0x860248920193194ELL   /*   34 */,    0xA4F9533B2D9CC0B3LL   /*   35 */,
    0x9053836C15957613LL   /*   36 */,    0xDB6DCF8AFC357BF1LL   /*   37 */,
    0x18BEEA7A7A370F57LL   /*   38 */,    0x037117CA50B99066LL   /*   39 */,
    0x6AB30A9774424A35LL   /*   40 */,    0xF4E92F02E325249BLL   /*   41 */,
    0x7739DB07061CCAE1LL   /*   42 */,    0xD8F3B49CECA42A05LL   /*   43 */,
    0xBD56BE3F51382F73LL   /*   44 */,    0x45FAED5843B0BB28LL   /*   45 */,
    0x1C813D5C11BF1F83LL   /*   46 */,    0x8AF0E4B6D75FA169LL   /*   47 */,
    0x33EE18A487AD9999LL   /*   48 */,    0x3C26E8EAB1C94410LL   /*   49 */,
    0xB510102BC0A822F9LL   /*   50 */,    0x141EEF310CE6123BLL   /*   51 */,
    0xFC65B90059DDB154LL   /*   52 */,    0xE0158640C5E0E607LL   /*   53 */,
    0x884E079826C3A3CFLL   /*   54 */,    0x930D0D9523C535FDLL   /*   55 */,
    0x35638D754E9A2B00LL   /*   56 */,    0x4085FCCF40469DD5LL   /*   57 */,
    0xC4B17AD28BE23A4CLL   /*   58 */,    0xCAB2F0FC6A3E6A2ELL   /*   59 */,
    0x2860971A6B943FCDLL   /*   60 */,    0x3DDE6EE212E30446LL   /*   61 */,
    0x6222F32AE01765AELL   /*   62 */,    0x5D550BB5478308FELL   /*   63 */,
    0xA9EFA98DA0EDA22ALL   /*   64 */,    0xC351A71686C40DA7LL   /*   65 */,
    0x1105586D9C867C84LL   /*   66 */,    0xDCFFEE85FDA22853LL   /*   67 */,
    0xCCFBD0262C5EEF76LL   /*   68 */,    0xBAF294CB8990D201LL   /*   69 */,
    0xE69464F52AFAD975LL   /*   70 */,    0x94B013AFDF133E14LL   /*   71 */,
    0x06A7D1A32823C958LL   /*   72 */,    0x6F95FE5130F61119LL   /*   73 */,
    0xD92AB34E462C06C0LL   /*   74 */,    0xED7BDE33887C71D2LL   /*   75 */,
    0x79746D6E6518393ELL   /*   76 */,    0x5BA419385D713329LL   /*   77 */,
    0x7C1BA6B948A97564LL   /*   78 */,    0x31987C197BFDAC67LL   /*   79 */,
    0xDE6C23C44B053D02LL   /*   80 */,    0x581C49FED002D64DLL   /*   81 */,
    0xDD474D6338261571LL   /*   82 */,    0xAA4546C3E473D062LL   /*   83 */,
    0x928FCE349455F860LL   /*   84 */,    0x48161BBACAAB94D9LL   /*   85 */,
    0x63912430770E6F68LL   /*   86 */,    0x6EC8A5E602C6641CLL   /*   87 */,
    0x87282515337DDD2BLL   /*   88 */,    0x2CDA6B42034B701BLL   /*   89 */,
    0xB03D37C181CB096DLL   /*   90 */,    0xE108438266C71C6FLL   /*   91 */,
    0x2B3180C7EB51B255LL   /*   92 */,    0xDF92B82F96C08BBCLL   /*   93 */,
    0x5C68C8C0A632F3BALL   /*   94 */,    0x5504CC861C3D0556LL   /*   95 */,
    0xABBFA4E55FB26B8FLL   /*   96 */,    0x41848B0AB3BACEB4LL   /*   97 */,
    0xB334A273AA445D32LL   /*   98 */,    0xBCA696F0A85AD881LL   /*   99 */,
    0x24F6EC65B528D56CLL   /*  100 */,    0x0CE1512E90F4524ALL   /*  101 */,
    0x4E9DD79D5506D35ALL   /*  102 */,    0x258905FAC6CE9779LL   /*  103 */,
    0x2019295B3E109B33LL   /*  104 */,    0xF8A9478B73A054CCLL   /*  105 */,
    0x2924F2F934417EB0LL   /*  106 */,    0x3993357D536D1BC4LL   /*  107 */,
    0x38A81AC21DB6FF8BLL   /*  108 */,    0x47C4FBF17D6016BFLL   /*  109 */,
    0x1E0FAADD7667E3F5LL   /*  110 */,    0x7ABCFF62938BEB96LL   /*  111 */,
    0xA78DAD948FC179C9LL   /*  112 */,    0x8F1F98B72911E50DLL   /*  113 */,
    0x61E48EAE27121A91LL   /*  114 */,    0x4D62F7AD31859808LL   /*  115 */,
    0xECEBA345EF5CEAEBLL   /*  116 */,    0xF5CEB25EBC9684CELL   /*  117 */,
    0xF633E20CB7F76221LL   /*  118 */,    0xA32CDF06AB8293E4LL   /*  119 */,
    0x985A202CA5EE2CA4LL   /*  120 */,    0xCF0B8447CC8A8FB1LL   /*  121 */,
    0x9F765244979859A3LL   /*  122 */,    0xA8D516B1A1240017LL   /*  123 */,
```

```
    0x0BD7BA3EBB5DC726LL   /*  124 */,    0xE54BCA55B86ADB39LL   /*  125 */,
    0x1D7A3AFD6C478063LL   /*  126 */,    0x519EC608E7669EDDLL   /*  127 */,
    0x0E5715A2D149AA23LL   /*  128 */,    0x177D4571848FF194LL   /*  129 */,
    0xEEB55F3241014C22LL   /*  130 */,    0x0F5E5CA13A6E2EC2LL   /*  131 */,
    0x8029927B75F5C361LL   /*  132 */,    0xAD139FABC3D6E436LL   /*  133 */,
    0x0D5DF1A94CCF402FLL   /*  134 */,    0x3E8BD948BEA5DFC8LL   /*  135 */,
    0xA5A0D357BD3FF77ELL   /*  136 */,    0xA2D12E251F74F645LL   /*  137 */,
    0x66FD9E525E81A082LL   /*  138 */,    0x2E0C90CE7F687A49LL   /*  139 */,
    0xC2E8BCBEBA973BC5LL   /*  140 */,    0x000001BCE509745FLL   /*  141 */,
    0x423777BBE6DAB3D6LL   /*  142 */,    0xD1661C7EAEF06EB5LL   /*  143 */,
    0xA1781F354DAACFD8LL   /*  144 */,    0x2D11284A2B16AFFCLL   /*  145 */,
    0xF1FC4F67FA891D1FLL   /*  146 */,    0x73ECC25DCB920ADALL   /*  147 */,
    0xAE610C22C2A12651LL   /*  148 */,    0x96E0A810D356B78ALL   /*  149 */,
    0x5A9A381F2FE7870FLL   /*  150 */,    0xD5AD62EDE94E5530LL   /*  151 */,
    0xD225E5E8368D1427LL   /*  152 */,    0x65977B70C7AF4631LL   /*  153 */,
    0x99F889B2DE39D74FLL   /*  154 */,    0x233F30BF54E1D143LL   /*  155 */,
    0x9A9675D3D9A63C97LL   /*  156 */,    0x5470554FF334F9A8LL   /*  157 */,
    0x166ACB744A4F5688LL   /*  158 */,    0x70C74CAAB2E4AEADLL   /*  159 */,
    0xF0D091646F294D12LL   /*  160 */,    0x57B82A89684031D1LL   /*  161 */,
    0xEFD95A5A61BE0B6BLL   /*  162 */,    0x2FBD12E969F2F29ALL   /*  163 */,
    0x9BD37013FEFF9FE8LL   /*  164 */,    0x3F9B0404D6085A06LL   /*  165 */,
    0x4940C1F3166CFE15LL   /*  166 */,    0x09542C4DCDF3DEFBLL   /*  167 */,
    0xB4C5218385CD5CE3LL   /*  168 */,    0xC935B7DC4462A641LL   /*  169 */,
    0x3417F8A68ED3B63FLL   /*  170 */,    0xB80959295B215B40LL   /*  171 */,
    0xF99CDAEF3B8C8572LL   /*  172 */,    0x018C0614F8FCB95DLL   /*  173 */,
    0x1B14ACCD1A3ACDF3LL   /*  174 */,    0x84D471F200BB732DLL   /*  175 */,
    0xC1A3110E95E8DA16LL   /*  176 */,    0x430A7220BF1A82B8LL   /*  177 */,
    0xB77E090D39DF210ELL   /*  178 */,    0x5EF4BD9F3CD05E9DLL   /*  179 */,
    0x9D4FF6DA7E57A444LL   /*  180 */,    0xDA1D60E183D4A5F8LL   /*  181 */,
    0xB287C38417998E47LL   /*  182 */,    0xFE3EDC121BB31886LL   /*  183 */,
    0xC7FE3CCC980CCBEFLL   /*  184 */,    0xE46FB590189BFD03LL   /*  185 */,
    0x3732FD469A4C57DCLL   /*  186 */,    0x7EF700A07CF1AD65LL   /*  187 */,
    0x59C64468A31D8859LL   /*  188 */,    0x762FB0B4D45B61F6LL   /*  189 */,
    0x155BAED099047718LL   /*  190 */,    0x68755E4C3D50BAA6LL   /*  191 */,
    0xE9214E7F22D8B4DFLL   /*  192 */,    0x2ADDBF532EAC95F4LL   /*  193 */,
    0x32AE3909B4BD0109LL   /*  194 */,    0x834DF537B08E3450LL   /*  195 */,
    0xFA209DA84220728DLL   /*  196 */,    0x9E691D9B9EFE23F7LL   /*  197 */,
    0x0446D288C4AE8D7FLL   /*  198 */,    0x7B4CC524E169785BLL   /*  199 */,
    0x21D87F0135CA1385LL   /*  200 */,    0xCEBB400F137B8AA5LL   /*  201 */,
    0x272E2B66580796BELL   /*  202 */,    0x3612264125C2B0DELL   /*  203 */,
    0x057702BDAD1EFBB2LL   /*  204 */,    0xD4BABB8EACF84BE9LL   /*  205 */,
    0x91583139641BC67BLL   /*  206 */,    0x8BDC2DE08036E024LL   /*  207 */,
    0x603C8156F49F68EDLL   /*  208 */,    0xF7D236F7DBEF5111LL   /*  209 */,
    0x9727C4598AD21E80LL   /*  210 */,    0xA08A0896670A5FD7LL   /*  211 */,
    0xCB4A8F4309EBA9CBLL   /*  212 */,    0x81AF564B0F7036A1LL   /*  213 */,
    0xC0B99AA778199ABDLL   /*  214 */,    0x959F1EC83FC8E952LL   /*  215 */,
    0x8C505077794A81B9LL   /*  216 */,    0x3ACAAF8F056338F0LL   /*  217 */,
    0x07B43F50627A6778LL   /*  218 */,    0x4A44AB49F5ECCC77LL   /*  219 */,
    0x3BC3D6E4B679EE98LL   /*  220 */,    0x9CC0D4D1CF14108CLL   /*  221 */,
    0x4406C00B206BC8A0LL   /*  222 */,    0x82A18854C8D72D89LL   /*  223 */,
    0x67E366B35C3C432CLL   /*  224 */,    0xB923DD61102B37F2LL   /*  225 */,
    0x56AB2779D884271DLL   /*  226 */,    0xBE83E1B0FF1525AFLL   /*  227 */,
    0xFB7C65D4217E49A9LL   /*  228 */,    0x6BDBE0E76D48E7D4LL   /*  229 */,
    0x08DF828745D9179ELL   /*  230 */,    0x22EA6A9ADD53BD34LL   /*  231 */,
    0xE36E141C5622200ALL   /*  232 */,    0x7F805D1B8CB750EELL   /*  233 */,
    0xAFE5C7A59F58E837LL   /*  234 */,    0xE27F996A4FB1C23CLL   /*  235 */,
    0xD3867DFB0775F0D0LL   /*  236 */,    0xD0E673DE6E88891ALL   /*  237 */,
    0x123AEB9EAFB86C25LL   /*  238 */,    0x30F1D5D5C145B895LL   /*  239 */,
    0xBB434A2DEE7269E7LL   /*  240 */,    0x78CB67ECF931FA38LL   /*  241 */,
    0xF33B0372323BBF9CLL   /*  242 */,    0x52D66336FB279C74LL   /*  243 */,
    0x505F33AC0AFB4EAALL   /*  244 */,    0xE8A5CD99A2CCE187LL   /*  245 */,
    0x534974801E2D30BBLL   /*  246 */,    0x8D2D5711D5876D90LL   /*  247 */,
    0x1F1A412891BC038ELL   /*  248 */,    0xD6E2E71D82E56648LL   /*  249 */,
    0x74036C3A497732B7LL   /*  250 */,    0x89B67ED96361F5ABLL   /*  251 */,
    0xFFED95D8F1EA02A2LL   /*  252 */,    0xE72B3BD61464D43DLL   /*  253 */,
    0xA6300F170BDC4820LL   /*  254 */,    0xEBC18760ED78A77ALL   /*  255 */,
    0xE6A6BE5A05A12138LL   /*  256 */,    0xB5A122A5B4F87C98LL   /*  257 */,
    0x563C6089140B6990LL   /*  258 */,    0x4C46CB2E391F5DD5LL   /*  259 */,
    0xD932ADDBC9B79434LL   /*  260 */,    0x08EA70E42015AFF5LL   /*  261 */,
    0xD765A6673E478CF1LL   /*  262 */,    0xC4FB757EAB278D99LL   /*  263 */,
    0xDF11C6862D6E0692LL   /*  264 */,    0xDDEB84F10D7F3B16LL   /*  265 */,
```

```
    0x6F2EF604A665EA04LL   /*  266 */,    0x4A8E0F0FF0E0DFB3LL   /*  267 */,
    0xA5EDEEF83DBCBA51LL   /*  268 */,    0xFC4F0A2A0EA4371ELL   /*  269 */,
    0xE83E1DA85CB38429LL   /*  270 */,    0xDC8FF882BA1B1CE2LL   /*  271 */,
    0xCD45505E8353E80DLL   /*  272 */,    0x18D19A00D4DB0717LL   /*  273 */,
    0x34A0CFEDA5F38101LL   /*  274 */,    0x0BE77E518887CAF2LL   /*  275 */,
    0x1E341438B3C45136LL   /*  276 */,    0xE05797F49089CCF9LL   /*  277 */,
    0xFFD23F9DF2591D14LL   /*  278 */,    0x543DDA228595C5CDLL   /*  279 */,
    0x661F81FD99052A33LL   /*  280 */,    0x8736E641DB0F7B76LL   /*  281 */,
    0x15227725418E5307LL   /*  282 */,    0xE25F7F46162EB2FALL   /*  283 */,
    0x48A8B2126C13D9FELL   /*  284 */,    0xAFDC541792E76EEALL   /*  285 */,
    0x03D912BFC6D1898FLL   /*  286 */,    0x31B1AAFA1B83F51BLL   /*  287 */,
    0xF1AC2796E42AB7D9LL   /*  288 */,    0x40A3A7D7FCD2EBACLL   /*  289 */,
    0x1056136D0AFBBCC5LL   /*  290 */,    0x7889E1DD9A6D0C85LL   /*  291 */,
    0xD33525782A7974AALL   /*  292 */,    0xA7E25D09078AC09BLL   /*  293 */,
    0xBD4138B3EAC6EDD0LL   /*  294 */,    0x920ABFBE71EB9E70LL   /*  295 */,
    0xA2A5D0F54FC2625CLL   /*  296 */,    0xC054E36B0B1290A3LL   /*  297 */,
    0xF6DD59FF62FE932BLL   /*  298 */,    0x3537354511A8AC7DLL   /*  299 */,
    0xCA845E9172FADCD4LL   /*  300 */,    0x84F82B60329D20DCLL   /*  301 */,
    0x79C62CE1CD672F18LL   /*  302 */,    0x8B09A2ADD124642CLL   /*  303 */,
    0xD0C1E96A19D9E726LL   /*  304 */,    0x5A786A9B4BA9500CLL   /*  305 */,
    0x0E020336634C43F3LL   /*  306 */,    0xC17B474AEB66D822LL   /*  307 */,
    0x6A731AE3EC9BAAC2LL   /*  308 */,    0x8226667AE0840258LL   /*  309 */,
    0x67D4567691CAECA5LL   /*  310 */,    0x1D94155C4875ADB5LL   /*  311 */,
    0x6D00FD985B813FDFLL   /*  312 */,    0x51286EFCB774CD06LL   /*  313 */,
    0x5E8834471FA744AFLL   /*  314 */,    0xF72CA0AEE761AE2ELL   /*  315 */,
    0xBE40E4CDAEE8E09ALL   /*  316 */,    0xE9970BBB5118F665LL   /*  317 */,
    0x726E4BEB33DF1964LL   /*  318 */,    0x703B000729199762LL   /*  319 */,
    0x4631D816F5EF30A7LL   /*  320 */,    0xB880B5B51504A6BELL   /*  321 */,
    0x641793C37ED84B6CLL   /*  322 */,    0x7B21ED77F6E97D96LL   /*  323 */,
    0x776306312EF96B73LL   /*  324 */,    0xAE528948E86FF3F4LL   /*  325 */,
    0x53DBD7F286A3F8F8LL   /*  326 */,    0x16CADCE74CFC1063LL   /*  327 */,
    0x005C19BDFA52C6DDLL   /*  328 */,    0x68868F5D64D46AD3LL   /*  329 */,
    0x3A9D512CCF1E186ALL   /*  330 */,    0x367E62C2385660AELL   /*  331 */,
    0xE359E7EA77DCB1D7LL   /*  332 */,    0x526C0773749ABE6ELL   /*  333 */,
    0x735AE5F9D09F734BLL   /*  334 */,    0x493FC7CC8A558BA8LL   /*  335 */,
    0xB0B9C1533041AB45LL   /*  336 */,    0x321958BA470A59BDLL   /*  337 */,
    0x852DB00B5F46C393LL   /*  338 */,    0x91209B2BD336B0E5LL   /*  339 */,
    0x6E604F7D659EF19FLL   /*  340 */,    0xB99A8AE2782CCB24LL   /*  341 */,
    0xCCF52AB6C814C4C7LL   /*  342 */,    0x4727D9AFBE11727BLL   /*  343 */,
    0x7E950D0C0121B34DLL   /*  344 */,    0x756F435670AD471FLL   /*  345 */,
    0xF5ADD442615A6849LL   /*  346 */,    0x4E87E09980B9957ALL   /*  347 */,
    0x2ACFA1DF50AEE355LL   /*  348 */,    0xD898263AFD2FD556LL   /*  349 */,
    0xC8F4924DD80C8FD6LL   /*  350 */,    0xCF99CA3D754A173ALL   /*  351 */,
    0xFE477BACAF91BF3CLL   /*  352 */,    0xED5371F6D690C12DLL   /*  353 */,
    0x831A5C285E687094LL   /*  354 */,    0xC5D3C90A3708A0A4LL   /*  355 */,
    0x0F7F903717D06580LL   /*  356 */,    0x19F9BB13B8FDF27FLL   /*  357 */,
    0xB1BD6F1B4D502843LL   /*  358 */,    0x1C761BA38FFF4012LL   /*  359 */,
    0x0D1530C4E2E21F3BLL   /*  360 */,    0x8943CE69A7372C8ALL   /*  361 */,
    0xE5184E11FEB5CE66LL   /*  362 */,    0x618BDB80BD736621LL   /*  363 */,
    0x7D29BAD68B574D0BLL   /*  364 */,    0x81BB613E25E6FE5BLL   /*  365 */,
    0x071C9C10BC07913FLL   /*  366 */,    0xC7BEEB7909AC2D97LL   /*  367 */,
    0xC3E58D353BC5D757LL   /*  368 */,    0xEB017892F38F61E8LL   /*  369 */,
    0xD4EFFB9C9B1CC21ALL   /*  370 */,    0x99727D26F494F7ABLL   /*  371 */,
    0xA3E063A2956B3E03LL   /*  372 */,    0x9D4A8B9A4AA09C30LL   /*  373 */,
    0x3F6AB7D500090FB4LL   /*  374 */,    0x9CC0F2A057268AC0LL   /*  375 */,
    0x3DEE9D2DEDBF42D1LL   /*  376 */,    0x330F49C87960A972LL   /*  377 */,
    0xC6B2720287421B41LL   /*  378 */,    0x0AC59EC07C00369CLL   /*  379 */,
    0xEF4EAC49CB353425LL   /*  380 */,    0xF450244EEF0129D8LL   /*  381 */,
    0x8ACC46E5CAF4DEB6LL   /*  382 */,    0x2FFEAB63989263F7LL   /*  383 */,
    0x8F7CB9FE5D7A4578LL   /*  384 */,    0x5BD8F7644E634635LL   /*  385 */,
    0x427A7315BF2DC900LL   /*  386 */,    0x17D0C4AA2125261CLL   /*  387 */,
    0x3992486C93518E50LL   /*  388 */,    0xB4CBFEE0A2D7D4C3LL   /*  389 */,
    0x7C75D6202C5DDD8DLL   /*  390 */,    0xDBC295D8E35B6C61LL   /*  391 */,
    0x60B369D302032B19LL   /*  392 */,    0xCE42685FDCE44132LL   /*  393 */,
    0x06F3DDB9DDF65610LL   /*  394 */,    0x8EA4D21DB5E148F0LL   /*  395 */,
    0x20B0FCE62FCD496FLL   /*  396 */,    0x2C1B912358B0EE31LL   /*  397 */,
    0xB28317B818F5A308LL   /*  398 */,    0xA89C1E189CA6D2CFLL   /*  399 */,
    0x0C6B18576AAADBC8LL   /*  400 */,    0xB65DEAA91299FAE3LL   /*  401 */,
    0xFB2B794B7F1027E7LL   /*  402 */,    0x04E4317F443B5BEBLL   /*  403 */,
    0x4B852D325939D0A6LL   /*  404 */,    0xD5AE6BEEFB207FFCLL   /*  405 */,
    0x309682B281C7D374LL   /*  406 */,    0xBAE309A194C3B475LL   /*  407 */,
```

```
    0x8CC3F97B13B49F05LL   /*  408 */,    0x98A9422FF8293967LL   /*  409 */,
    0x244B16B01076FF7CLL   /*  410 */,    0xF8BF571C663D67EELL   /*  411 */,
    0x1F0D6758EEE30DA1LL   /*  412 */,    0xC9B611D97ADEB9B7LL   /*  413 */,
    0xB7AFD5887B6C57A2LL   /*  414 */,    0x6290AE846B984FE1LL   /*  415 */,
    0x94DF4CDEACC1A5FDLL   /*  416 */,    0x058A5BD1C5483AFFLL   /*  417 */,
    0x63166CC142BA3C37LL   /*  418 */,    0x8DB8526EB2F76F40LL   /*  419 */,
    0xE10880036F0D6D4ELL   /*  420 */,    0x9E0523C9971D311DLL   /*  421 */,
    0x45EC2824CC7CD691LL   /*  422 */,    0x575B8359E62382C9LL   /*  423 */,
    0xFA9E400DC4889995LL   /*  424 */,    0xD1823ECB45721568LL   /*  425 */,
    0xDAFD983B8206082FLL   /*  426 */,    0xAA7D29082386A8CBLL   /*  427 */,
    0x269FCD4403B87588LL   /*  428 */,    0x1B91F5F728BDD1E0LL   /*  429 */,
    0xE4669F39040201F6LL   /*  430 */,    0x7A1D7C218CF04ADELL   /*  431 */,
    0x65623C29D79CE5CELL   /*  432 */,    0x2368449096C00BB1LL   /*  433 */,
    0xAB9BF1879DA503BALL   /*  434 */,    0xBC23ECB1A458058ELL   /*  435 */,
    0x9A58DF01BB401ECCLL   /*  436 */,    0xA070E868A85F143DLL   /*  437 */,
    0x4FF188307DF2239ELL   /*  438 */,    0x14D565B41A641183LL   /*  439 */,
    0xEE13337452701602LL   /*  440 */,    0x950E3DCF3F285E09LL   /*  441 */,
    0x59930254B9C80953LL   /*  442 */,    0x3BF299408930DA6DLL   /*  443 */,
    0xA955943F53691387LL   /*  444 */,    0xA15EDECAA9CB8784LL   /*  445 */,
    0x29142127352BE9A0LL   /*  446 */,    0x76F0371FFF4E7AFBLL   /*  447 */,
    0x0239F450274F2228LL   /*  448 */,    0xBB073AF01D5E868BLL   /*  449 */,
    0xBFC80571C10E96C1LL   /*  450 */,    0xD267088568222E23LL   /*  451 */,
    0x9671A3D48E80B5B0LL   /*  452 */,    0x55B5D38AE193BB81LL   /*  453 */,
    0x693AE2D0A18B04B8LL   /*  454 */,    0x5C48B4ECADD5335FLL   /*  455 */,
    0xFD743B194916A1CALL   /*  456 */,    0x2577018134BE98C4LL   /*  457 */,
    0xE77987E83C54A4ADLL   /*  458 */,    0x28E11014DA33E1B9LL   /*  459 */,
    0x270CC59E226AA213LL   /*  460 */,    0x71495F756D1A5F60LL   /*  461 */,
    0x9BE853FB60AFEF77LL   /*  462 */,    0xADC786A7F7443DBFLL   /*  463 */,
    0x0904456173B29A82LL   /*  464 */,    0x58BC7A66C232BD5ELL   /*  465 */,
    0xF306558C673AC8B2LL   /*  466 */,    0x41F639C6B6C9772ALL   /*  467 */,
    0x216DEFE99FDA35DALL   /*  468 */,    0x11640CC71C7BE615LL   /*  469 */,
    0x93C43694565C5527LL   /*  470 */,    0xEA038E6246777839LL   /*  471 */,
    0xF9ABF3CE5A3E2469LL   /*  472 */,    0x741E768D0FD312D2LL   /*  473 */,
    0x0144B883CED652C6LL   /*  474 */,    0xC20B5A5BA33F8552LL   /*  475 */,
    0x1AE69633C3435A9DLL   /*  476 */,    0x97A28CA4088CFDECLL   /*  477 */,
    0x8824A43C1E96F420LL   /*  478 */,    0x37612FA66EEEA746LL   /*  479 */,
    0x6B4CB165F9CF0E5ALL   /*  480 */,    0x43AA1C06A0ABFB4ALL   /*  481 */,
    0x7F4DC26FF162796BLL   /*  482 */,    0x6CBACC8E54ED9B0FLL   /*  483 */,
    0xA6B7FFEFD2BB253ELL   /*  484 */,    0x2E25BC95B0A29D4FLL   /*  485 */,
    0x86D6A58BDEF1388CLL   /*  486 */,    0xDED74AC576B6F054LL   /*  487 */,
    0x8030BDBC2B45805DLL   /*  488 */,    0x3C81AF70E94D9289LL   /*  489 */,
    0x3EFF6DDA9E3100DBLL   /*  490 */,    0xB38DC39FDFCC8847LL   /*  491 */,
    0x123885528D17B87ELL   /*  492 */,    0xF2DA0ED240B1B642LL   /*  493 */,
    0x44CEFADCD54BF9A9LL   /*  494 */,    0x1312200E433C7EE6LL   /*  495 */,
    0x9FFCC84F3A78C748LL   /*  496 */,    0xF0CD1F72248576BBLL   /*  497 */,
    0xEC6974053638CFE4LL   /*  498 */,    0x2BA7B67C0CEC4E4CLL   /*  499 */,
    0xAC2F4DF3E5CE32EDLL   /*  500 */,    0xCB33D14326EA4C11LL   /*  501 */,
    0xA4E9044CC77E58BCLL   /*  502 */,    0x5F513293D934FCEFLL   /*  503 */,
    0x5DC9645506E55444LL   /*  504 */,    0x50DE418F317DE40ALL   /*  505 */,
    0x388CB31A69DDE259LL   /*  506 */,    0x2DB4A83455820A86LL   /*  507 */,
    0x9010A91E84711AE9LL   /*  508 */,    0x4DF7F0B7B1498371LL   /*  509 */,
    0xD62A2EABC0977179LL   /*  510 */,    0x22FAC097AA8D5C0ELL   /*  511 */,
    0xF49FCC2FF1DAF39BLL   /*  512 */,    0x487FD5C66FF29281LL   /*  513 */,
    0xE8A30667FCDCA83FLL   /*  514 */,    0x2C9B4BE3D2FCCE63LL   /*  515 */,
    0xDA3FF74B93FBBBC2LL   /*  516 */,    0x2FA165D2FE70BA66LL   /*  517 */,
    0xA103E279970E93D4LL   /*  518 */,    0xBECDEC77B0E45E71LL   /*  519 */,
    0xCFB41E723985E497LL   /*  520 */,    0xB70AAA025EF75017LL   /*  521 */,
    0xD42309F03840B8E0LL   /*  522 */,    0x8EFC1AD035898579LL   /*  523 */,
    0x96C6920BE2B2ABC5LL   /*  524 */,    0x66AF4163375A9172LL   /*  525 */,
    0x2174ABDCCA7127FBLL   /*  526 */,    0xB33CCEA64A72FF41LL   /*  527 */,
    0xF04A4933083066A5LL   /*  528 */,    0x8D970ACDD7289AF5LL   /*  529 */,
    0x8F96E8E031C8C25ELL   /*  530 */,    0xF3FEC02276875D47LL   /*  531 */,
    0xEC7BF310056190DDLL   /*  532 */,    0xF5ADB0AEBB0F1491LL   /*  533 */,
    0x9B50F8850FD58892LL   /*  534 */,    0x4975488358B74DE8LL   /*  535 */,
    0xA3354FF691531C61LL   /*  536 */,    0x0702BBE481D2C6EELL   /*  537 */,
    0x89FB24057DEDED98LL   /*  538 */,    0xAC3075138596E902LL   /*  539 */,
    0x1D2D3580172772EDLL   /*  540 */,    0xEB738FC28E6BC30DLL   /*  541 */,
    0x5854EF8F63044326LL   /*  542 */,    0x9E5C52325ADD3BBELL   /*  543 */,
    0x90AA53CF325C4623LL   /*  544 */,    0xC1D24D51349DD067LL   /*  545 */,
    0x2051CFEEA69EA624LL   /*  546 */,    0x13220F0A862E7E4FLL   /*  547 */,
    0xCE39399404E04864LL   /*  548 */,    0xD9C42CA47086FCB7LL   /*  549 */,
```

```
    0x685AD2238A03E7CCLL   /*  550 */,    0x066484B2AB2FF1DBLL   /*  551 */,
    0xFE9D5D70EFBF79ECLL   /*  552 */,    0x5B13B9DD9C481854LL   /*  553 */,
    0x15F0D475ED1509ADLL   /*  554 */,    0x0BEBCD060EC79851LL   /*  555 */,
    0xD58C6791183AB7F8LL   /*  556 */,    0xD1187C5052F3EEE4LL   /*  557 */,
    0xC95D1192E54E82FFLL   /*  558 */,    0x86EEA14CB9AC6CA2LL   /*  559 */,
    0x3485BEB153677D5DLL   /*  560 */,    0xDD191D781F8C492ALL   /*  561 */,
    0xF60866BAA784EBF9LL   /*  562 */,    0x518F643BA2D08C74LL   /*  563 */,
    0x8852E956E1087C22LL   /*  564 */,    0xA768CB8DC410AE8DLL   /*  565 */,
    0x38047726BFEC8E1ALL   /*  566 */,    0xA67738B4CD3B45AALL   /*  567 */,
    0xAD16691CEC0DDE19LL   /*  568 */,    0xC6D4319380462E07LL   /*  569 */,
    0xC5A5876D0BA61938LL   /*  570 */,    0x16B9FA1FA58FD840LL   /*  571 */,
    0x188AB1173CA74F18LL   /*  572 */,    0xABDA2F98C99C021FLL   /*  573 */,
    0x3E0580AB134AE816LL   /*  574 */,    0x5F3B05B773645ABBLL   /*  575 */,
    0x2501A2BE5575F2F6LL   /*  576 */,    0x1B2F74004E7E8BA9LL   /*  577 */,
    0x1CD7580371E8D953LL   /*  578 */,    0x7F6ED89562764E30LL   /*  579 */,
    0xB15926FF596F003DLL   /*  580 */,    0x9F65293DA8C5D6B9LL   /*  581 */,
    0x6ECEF04DD690F84CLL   /*  582 */,    0x4782275FFF33AF88LL   /*  583 */,
    0xE41433083F820801LL   /*  584 */,    0xFD0DFE409A1AF9B5LL   /*  585 */,
    0x4325A3342CDB396BLL   /*  586 */,    0x8AE77E62B301B252LL   /*  587 */,
    0xC36F9E9F6655615ALL   /*  588 */,    0x85455A2D92D32C09LL   /*  589 */,
    0xF2C7DEA949477485LL   /*  590 */,    0x63CFB4C133A39EBALL   /*  591 */,
    0x83B040CC6EBC5462LL   /*  592 */,    0x3B9454C8FDB326B0LL   /*  593 */,
    0x56F56A9E87FFD78CLL   /*  594 */,    0x2DC2940D99F42BC6LL   /*  595 */,
    0x98F7DF096B096E2DLL   /*  596 */,    0x19A6E01E3AD852BFLL   /*  597 */,
    0x42A99CCBDBD4B40BLL   /*  598 */,    0xA59998AF45E9C559LL   /*  599 */,
    0x366295E807D93186LL   /*  600 */,    0x6B48181BFAA1F773LL   /*  601 */,
    0x1FEC57E2157A0A1DLL   /*  602 */,    0x4667446AF6201AD5LL   /*  603 */,
    0xE615EBCACFB0F075LL   /*  604 */,    0xB8F31F4F68290778LL   /*  605 */,
    0x22713ED6CE22D11ELL   /*  606 */,    0x3057C1A72EC3C93BLL   /*  607 */,
    0xCB46ACC37C3F1F2FLL   /*  608 */,    0xDBB893FD02AAF50ELL   /*  609 */,
    0x331FD92E600B9FCFLL   /*  610 */,    0xA498F96148EA3AD6LL   /*  611 */,
    0xA8D8426E8B6A83EALL   /*  612 */,    0xA089B274B7735CDCLL   /*  613 */,
    0x87F6B3731E524A11LL   /*  614 */,    0x118808E5CBC96749LL   /*  615 */,
    0x9906E4C7B19BD394LL   /*  616 */,    0xAFED7F7E9B24A20CLL   /*  617 */,
    0x6509EADEEB3644A7LL   /*  618 */,    0x6C1EF1D3E8EF0EDELL   /*  619 */,
    0xB9C97D43E9798FB4LL   /*  620 */,    0xA2F2D784740C28A3LL   /*  621 */,
    0x7B8496476197566FLL   /*  622 */,    0x7A5BE3E6B65F069DLL   /*  623 */,
    0xF96330ED78BE6F10LL   /*  624 */,    0xEEE60DE77A076A15LL   /*  625 */,
    0x2B4BEE4AA08B9BD0LL   /*  626 */,    0x6A56A63EC7B8894ELL   /*  627 */,
    0x02121359BA34FEF4LL   /*  628 */,    0x4CBF99F8283703FCLL   /*  629 */,
    0x398071350CAF30C8LL   /*  630 */,    0xD0A77A89F017687ALL   /*  631 */,
    0xF1C1A9EB9E423569LL   /*  632 */,    0x8C7976282DEE8199LL   /*  633 */,
    0x5D1737A5DD1F7ABDLL   /*  634 */,    0x4F53433C09A9FA80LL   /*  635 */,
    0xFA8B0C53DF7CA1D9LL   /*  636 */,    0x3FD9DCBC886CCB77LL   /*  637 */,
    0xC040917CA91B4720LL   /*  638 */,    0x7DD00142F9D1DCDFLL   /*  639 */,
    0x8476FC1D4F387B58LL   /*  640 */,    0x23F8E7C5F3316503LL   /*  641 */,
    0x032A2244E7E37339LL   /*  642 */,    0x5C87A5D750F5A74BLL   /*  643 */,
    0x082B4CC43698992ELL   /*  644 */,    0xDF917BECB858F63CLL   /*  645 */,
    0x3270B8FC5BF86DDALL   /*  646 */,    0x10AE72BB29B5DD76LL   /*  647 */,
    0x576AC94E7700362BLL   /*  648 */,    0x1AD112DAC61EFB8FLL   /*  649 */,
    0x691BC30EC5FAA427LL   /*  650 */,    0xFF246311CC327143LL   /*  651 */,
    0x3142368E30E53206LL   /*  652 */,    0x71380E31E02CA396LL   /*  653 */,
    0x958D5C960AAD76F1LL   /*  654 */,    0xF8D6F430C16DA536LL   /*  655 */,
    0xC8FFD13F1BE7E1D2LL   /*  656 */,    0x7578AE66004DDBE1LL   /*  657 */,
    0x05833F01067BE646LL   /*  658 */,    0xBB34B5AD3BFE586DLL   /*  659 */,
    0x095F34C9A12B97F0LL   /*  660 */,    0x247AB64525D60CA8LL   /*  661 */,
    0xDCDBC6F3017477D1LL   /*  662 */,    0x4A2E14D4DECAD24DLL   /*  663 */,
    0xBDB5E6D9BE0A1EEBLL   /*  664 */,    0x2A7E70F7794301ABLL   /*  665 */,
    0xDEF42D8A270540FDLL   /*  666 */,    0x01078EC0A34C22C1LL   /*  667 */,
    0xE5DE511AF4C16387LL   /*  668 */,    0x7EBB3A52BD9A330ALL   /*  669 */,
    0x77697857AA7D6435LL   /*  670 */,    0x004E831603AE4C32LL   /*  671 */,
    0xE7A21020AD78E312LL   /*  672 */,    0x9D41A70C6AB420F2LL   /*  673 */,
    0x28E06C18EA1141E6LL   /*  674 */,    0xD2B28CBD984F6B28LL   /*  675 */,
    0x26B75F6C446E9D83LL   /*  676 */,    0xBA47568C4D418D7FLL   /*  677 */,
    0xD80BADBFE6183D8ELL   /*  678 */,    0x0E206D7F5F166044LL   /*  679 */,
    0xE258A43911CBCA3ELL   /*  680 */,    0x723A1746B21DC0BCLL   /*  681 */,
    0xC7CAA854F5D7CDD3LL   /*  682 */,    0x7CAC32883D261D9CLL   /*  683 */,
    0x7690C26423BA942CLL   /*  684 */,    0x17E55524478042B8LL   /*  685 */,
    0xE0BE477656A2389FLL   /*  686 */,    0x4D289B5E67AB2DA0LL   /*  687 */,
    0x44862B9C8FBBFD31LL   /*  688 */,    0xB47CC8049D141365LL   /*  689 */,
    0x822C1B362B91C793LL   /*  690 */,    0x4EB14655FB13DFD8LL   /*  691 */,
```

```
    0x1ECBBA0714E2A97BLL   /*  692 */,    0x6143459D5CDE5F14LL   /*  693 */,
    0x53A8FBF1D5F0AC89LL   /*  694 */,    0x97EA04D81C5E5B00LL   /*  695 */,
    0x622181A8D4FDB3F3LL   /*  696 */,    0xE9BCD341572A1208LL   /*  697 */,
    0x1411258643CCE58ALL   /*  698 */,    0x9144C5FEA4C6E0A4LL   /*  699 */,
    0x0D33D06565CF620FLL   /*  700 */,    0x54A48D489F219CA1LL   /*  701 */,
    0xC43E5EAC6D63C821LL   /*  702 */,    0xA9728B3A72770DAFLL   /*  703 */,
    0xD7934E7B20DF87EFLL   /*  704 */,    0xE35503B61A3E86E5LL   /*  705 */,
    0xCAE321FBC819D504LL   /*  706 */,    0x129A50B3AC60BFA6LL   /*  707 */,
    0xCD5E68EA7E9FB6C3LL   /*  708 */,    0xB01C90199483B1C7LL   /*  709 */,
    0x3DE93CD5C295376CLL   /*  710 */,    0xAED52EDF2AB9AD13LL   /*  711 */,
    0x2E60F512C0A07884LL   /*  712 */,    0xBC3D86A3E36210C9LL   /*  713 */,
    0x35269D9B163951CELL   /*  714 */,    0x0C7D6E2AD0CDB5FALL   /*  715 */,
    0x59E86297D87F5733LL   /*  716 */,    0x298EF221898DB0E7LL   /*  717 */,
    0x55000029D1A5AA7ELL   /*  718 */,    0x8BC08AE1B5061B45LL   /*  719 */,
    0xC2C31C2B6C92703ALL   /*  720 */,    0x94CC596BAF25EF42LL   /*  721 */,
    0x0A1D73DB22540456LL   /*  722 */,    0x04B6A0F9D9C4179ALL   /*  723 */,
    0xEFFDAFA2AE3D3C60LL   /*  724 */,    0xF7C8075BB49496C4LL   /*  725 */,
    0x9CC5C7141D1CD4E3LL   /*  726 */,    0x78BD1638218E5534LL   /*  727 */,
    0xB2F11568F850246ALL   /*  728 */,    0xEDFABCFA9502BC29LL   /*  729 */,
    0x796CE5F2DA23051BLL   /*  730 */,    0xAAE128B0DC93537CLL   /*  731 */,
    0x3A493DA0EE4B29AELL   /*  732 */,    0xB5DF6B2C416895D7LL   /*  733 */,
    0xFCABBD25122D7F37LL   /*  734 */,    0x70810B58105DC4B1LL   /*  735 */,
    0xE10FDD37F7882A90LL   /*  736 */,    0x524DCAB5518A3F5CLL   /*  737 */,
    0x3C9E85878451255BLL   /*  738 */,    0x4029828119BD34E2LL   /*  739 */,
    0x74A05B6F5D3CECCBLL   /*  740 */,    0xB610021542E13ECALL   /*  741 */,
    0x0FF979D12F59E2ACLL   /*  742 */,    0x6037DA27E4F9CC50LL   /*  743 */,
    0x5E92975A0DF1847DLL   /*  744 */,    0xD66DE190D3E623FELL   /*  745 */,
    0x5032D6B87B568048LL   /*  746 */,    0x9A36B7CE8235216ELL   /*  747 */,
    0x80272A7A24F64B4ALL   /*  748 */,    0x93EFED8B8C6916F7LL   /*  749 */,
    0x37DDBFF44CCE1555LL   /*  750 */,    0x4B95DB5D4B99BD25LL   /*  751 */,
    0x92D3FDA169812FC0LL   /*  752 */,    0xFB1A4A9A90660BB6LL   /*  753 */,
    0x730C196946A4B9B2LL   /*  754 */,    0x81E289AA7F49DA68LL   /*  755 */,
    0x64669A0F83B1A05FLL   /*  756 */,    0x27B3FF7D9644F48BLL   /*  757 */,
    0xCC6B615C8DB675B3LL   /*  758 */,    0x674F20B9BCEBBE95LL   /*  759 */,
    0x6F31238275655982LL   /*  760 */,    0x5AE488713E45CF05LL   /*  761 */,
    0xBF619F9954C21157LL   /*  762 */,    0xEABAC46040A8EAE9LL   /*  763 */,
    0x454C6FE9F2C0C1CDLL   /*  764 */,    0x419CF6496412691CLL   /*  765 */,
    0xD3DC3BEF265B0F70LL   /*  766 */,    0x6D0E60F5C3578A9ELL   /*  767 */,
    0x5B0E608526323C55LL   /*  768 */,    0x1A46C1A9FA1B59F5LL   /*  769 */,
    0xA9E245A17C4C8FFALL   /*  770 */,    0x65CA5159DB2955D7LL   /*  771 */,
    0x05DB0A76CE35AFC2LL   /*  772 */,    0x81EAC77EA9113D45LL   /*  773 */,
    0x528EF88AB6AC0A0DLL   /*  774 */,    0xA09EA253597BE3FFLL   /*  775 */,
    0x430DDFB3AC48CD56LL   /*  776 */,    0xC4B3A67AF45CE46FLL   /*  777 */,
    0x4ECECFD8FBE2D05ELL   /*  778 */,    0x3EF56F10B39935F0LL   /*  779 */,
    0x0B22D6829CD619C6LL   /*  780 */,    0x17FD460A74DF2069LL   /*  781 */,
    0x6CF8CC8E8510ED40LL   /*  782 */,    0xD6C824BF3A6ECAA7LL   /*  783 */,
    0x61243D581A817049LL   /*  784 */,    0x048BACB6BBC163A2LL   /*  785 */,
    0xD9A38AC27D44CC32LL   /*  786 */,    0x7FDDFF5BAAF410ABLL   /*  787 */,
    0xAD6D495AA804824BLL   /*  788 */,    0xE1A6A74F2D8C9F94LL   /*  789 */,
    0xD4F7851235DEE8E3LL   /*  790 */,    0xFD4B7F886540D893LL   /*  791 */,
    0x247C20042AA4BFDALL   /*  792 */,    0x096EA1C517D1327CLL   /*  793 */,
    0xD56966B4361A6685LL   /*  794 */,    0x277DA5C31221057DLL   /*  795 */,
    0x94D59893A43ACFF7LL   /*  796 */,    0x64F0C51CCDC02281LL   /*  797 */,
    0x3D33BCC4FF6189DBLL   /*  798 */,    0xE005CB184CE66AF1LL   /*  799 */,
    0xFF5CCD1D1DB99BEALL   /*  800 */,    0xB0B854A7FE42980FLL   /*  801 */,
    0x7BD46A6A718D4B9FLL   /*  802 */,    0xD10FA8CC22A5FD8CLL   /*  803 */,
    0xD31484952BE4BD31LL   /*  804 */,    0xC7FA975FCB243847LL   /*  805 */,
    0x4886ED1E5846C407LL   /*  806 */,    0x28CDDB791EB70B04LL   /*  807 */,
    0xC2B00BE2F573417FLL   /*  808 */,    0x5C9590452180F877LL   /*  809 */,
    0x7A6BDDFFF370EB00LL   /*  810 */,    0xCE509E38D6D9D6A4LL   /*  811 */,
    0xEBEB0F00647FA702LL   /*  812 */,    0x1DCC06CF76606F06LL   /*  813 */,
    0xE4D9F28BA286FF0ALL   /*  814 */,    0xD85A305DC918C262LL   /*  815 */,
    0x475B1D8732225F54LL   /*  816 */,    0x2D4FB51668CCB5FELL   /*  817 */,
    0xA679B9D9D72BBA20LL   /*  818 */,    0x53841C0D912D43A5LL   /*  819 */,
    0x3B7EAA48BF12A4E8LL   /*  820 */,    0x781E0E47F22F1DDFLL   /*  821 */,
    0xEFF20CE60AB50973LL   /*  822 */,    0x20D261D19DFFB742LL   /*  823 */,
    0x16A12B03062A2E39LL   /*  824 */,    0x1960EB2239650495LL   /*  825 */,
    0x251C16FED50EB8B8LL   /*  826 */,    0x9AC0C330F826016ELL   /*  827 */,
    0xED152665953E7671LL   /*  828 */,    0x02D63194A6369570LL   /*  829 */,
    0x5074F08394B1C987LL   /*  830 */,    0x70BA598C90B25CE1LL   /*  831 */,
    0x794A15810B9742F6LL   /*  832 */,    0x0D5925E9FCAF8C6CLL   /*  833 */,
```

```
    0x3067716CD868744ELL   /*  834 */,    0x910AB077E8D7731BLL   /*  835 */,
    0x6A61BBDB5AC42F61LL   /*  836 */,    0x93513EFBF0851567LL   /*  837 */,
    0xF494724B9E83E9D5LL   /*  838 */,    0xE887E1985C09648DLL   /*  839 */,
    0x34B1D3C675370CFDLL   /*  840 */,    0xDC35E433BC0D255DLL   /*  841 */,
    0xD0AAB84234131BE0LL   /*  842 */,    0x08042A50B48B7EAFLL   /*  843 */,
    0x9997C4EE44A3AB35LL   /*  844 */,    0x829A7B49201799D0LL   /*  845 */,
    0x263B8307B7C54441LL   /*  846 */,    0x752F95F4FD6A6CA6LL   /*  847 */,
    0x927217402C08C6E5LL   /*  848 */,    0x2A8AB754A795D9EELL   /*  849 */,
    0xA442F7552F72943DLL   /*  850 */,    0x2C31334E19781208LL   /*  851 */,
    0x4FA98D7CEAEE6291LL   /*  852 */,    0x55C3862F665DB309LL   /*  853 */,
    0xBD0610175D53B1F3LL   /*  854 */,    0x46FE6CB840413F27LL   /*  855 */,
    0x3FE03792DF0CFA59LL   /*  856 */,    0xCFE700372EB85E8FLL   /*  857 */,
    0xA7BE29E7ADBCE118LL   /*  858 */,    0xE544EE5CDE8431DDLL   /*  859 */,
    0x8A781B1B41F1873ELL   /*  860 */,    0xA5C94C78A0D2F0E7LL   /*  861 */,
    0x39412E2877B60728LL   /*  862 */,    0xA1265EF3AFC9A62CLL   /*  863 */,
    0xBCC2770C6A2506C5LL   /*  864 */,    0x3AB66DD5DCE1CE12LL   /*  865 */,
    0xE65499D04A675B37LL   /*  866 */,    0x7D8F523481BFD216LL   /*  867 */,
    0x0F6F64FCEC15F389LL   /*  868 */,    0x74EFBE618B5B13C8LL   /*  869 */,
    0xACDC82B714273E1DLL   /*  870 */,    0xDD40BFE003199D17LL   /*  871 */,
    0x37E99257E7E061F8LL   /*  872 */,    0xFA52626904775AAALL   /*  873 */,
    0x8BBBF63A463D56F9LL   /*  874 */,    0xF0013F1543A26E64LL   /*  875 */,
    0xA8307E9F879EC898LL   /*  876 */,    0xCC4C27A4150177CCLL   /*  877 */,
    0x1B432F2CCA1D3348LL   /*  878 */,    0xDE1D1F8F9F6FA013LL   /*  879 */,
    0x606602A047A7DDD6LL   /*  880 */,    0xD237AB64CC1CB2C7LL   /*  881 */,
    0x9B938E7225FCD1D3LL   /*  882 */,    0xEC4E03708E0FF476LL   /*  883 */,
    0xFEB2FBDA3D03C12DLL   /*  884 */,    0xAE0BCED2EE43889ALL   /*  885 */,
    0x22CB8923EBFB4F43LL   /*  886 */,    0x69360D013CF7396DLL   /*  887 */,
    0x855E3602D2D4E022LL   /*  888 */,    0x073805BAD01F784CLL   /*  889 */,
    0x33E17A133852F546LL   /*  890 */,    0xDF4874058AC7B638LL   /*  891 */,
    0xBA92B29C678AA14ALL   /*  892 */,    0x0CE89FC76CFAADCDLL   /*  893 */,
    0x5F9D4E0908339E34LL   /*  894 */,    0xF1AFE9291F5923B9LL   /*  895 */,
    0x6E3480F60F4A265FLL   /*  896 */,    0xEEBF3A2AB29B841CLL   /*  897 */,
    0xE21938A88F91B4ADLL   /*  898 */,    0x57DFEFF845C6D3C3LL   /*  899 */,
    0x2F006B0BF62CAAF2LL   /*  900 */,    0x62F479EF6F75EE78LL   /*  901 */,
    0x11A55AD41C8916A9LL   /*  902 */,    0xF229D29084FED453LL   /*  903 */,
    0x42F1C27B16B000E6LL   /*  904 */,    0x2B1F76749823C074LL   /*  905 */,
    0x4B76ECA3C2745360LL   /*  906 */,    0x8C98F463B91691BDLL   /*  907 */,
    0x14BCC93CF1ADE66ALL   /*  908 */,    0x8885213E6D458397LL   /*  909 */,
    0x8E177DF0274D4711LL   /*  910 */,    0xB49B73B5503F2951LL   /*  911 */,
    0x10168168C3F96B6BLL   /*  912 */,    0x0E3D963B63CAB0AELL   /*  913 */,
    0x8DFC4B5655A1DB14LL   /*  914 */,    0xF789F1356E14DE5CLL   /*  915 */,
    0x683E68AF4E51DAC1LL   /*  916 */,    0xC9A84F9D8D4B0FD9LL   /*  917 */,
    0x3691E03F52A0F9D1LL   /*  918 */,    0x5ED86E46E1878E80LL   /*  919 */,
    0x3C711A0E99D07150LL   /*  920 */,    0x5A0865B20C4E9310LL   /*  921 */,
    0x56FBFC1FE4F0682ELL   /*  922 */,    0xEA8D5DE3105EDF9BLL   /*  923 */,
    0x71ABFDB12379187ALL   /*  924 */,    0x2EB99DE1BEE77B9CLL   /*  925 */,
    0x21ECC0EA33CF4523LL   /*  926 */,    0x59A4D7521805C7A1LL   /*  927 */,
    0x3896F5EB56AE7C72LL   /*  928 */,    0xAA638F3DB18F75DCLL   /*  929 */,
    0x9F39358DABE9808ELL   /*  930 */,    0xB7DEFA91C00B72ACLL   /*  931 */,
    0x6B5541FD62492D92LL   /*  932 */,    0x6DC6DEE8F92E4D5BLL   /*  933 */,
    0x353F57ABC4BEEA7ELL   /*  934 */,    0x735769D6DA5690CELL   /*  935 */,
    0x0A234AA642391484LL   /*  936 */,    0xF6F9508028F80D9DLL   /*  937 */,
    0xB8E319A27AB3F215LL   /*  938 */,    0x31AD9C1151341A4DLL   /*  939 */,
    0x773C22A57BEF5805LL   /*  940 */,    0x45C7561A07968633LL   /*  941 */,
    0xF913DA9E249DBE36LL   /*  942 */,    0xDA652D9B78A64C68LL   /*  943 */,
    0x4C27A97F3BC334EFLL   /*  944 */,    0x76621220E66B17F4LL   /*  945 */,
    0x967743899ACD7D0BLL   /*  946 */,    0xF3EE5BCAE0ED6782LL   /*  947 */,
    0x409F753600C879FCLL   /*  948 */,    0x06D09A39B5926DB6LL   /*  949 */,
    0x6F83AEB0317AC588LL   /*  950 */,    0x01E6CA4A86381F21LL   /*  951 */,
    0x66FF3462D19F3025LL   /*  952 */,    0x72207C24DDFD3BFBLL   /*  953 */,
    0x4AF6B6D3E2ECE2EBLL   /*  954 */,    0x9C994DBEC7EA08DELL   /*  955 */,
    0x49ACE597B09A8BC4LL   /*  956 */,    0xB38C4766CF0797BALL   /*  957 */,
    0x131B9373C57C2A75LL   /*  958 */,    0xB1822CCE61931E58LL   /*  959 */,
    0x9D7555B909BA1C0CLL   /*  960 */,    0x127FAFDD937D11D2LL   /*  961 */,
    0x29DA3BADC66D92E4LL   /*  962 */,    0xA2C1D57154C2ECBCLL   /*  963 */,
    0x58C5134D82F6FE24LL   /*  964 */,    0x1C3AE3515B62274FLL   /*  965 */,
    0xE907C82E01CB8126LL   /*  966 */,    0xF8ED091913E37FCBLL   /*  967 */,
    0x3249D8F9C80046C9LL   /*  968 */,    0x80CF9BEDE388FB63LL   /*  969 */,
    0x1881539A116CF19ELL   /*  970 */,    0x5103F3F76BD52457LL   /*  971 */,
    0x15B7E6F5AE47F7A8LL   /*  972 */,    0xDBD7C6DED47E9CCFLL   /*  973 */,
    0x44E55C410228BB1ALL   /*  974 */,    0xB647D4255EDB4E99LL   /*  975 */,
```

```
    0x5D11882BB8AAFC30LL   /*  976 */,    0xF5098BBB29D3212ALL   /*  977 */,
    0x8FB5EA14E90296B3LL   /*  978 */,    0x677B942157DD025ALL   /*  979 */,
    0xFB58E7C0A390ACB5LL   /*  980 */,    0x89D3674C83BD4A01LL   /*  981 */,
    0x9E2DA4DF4BF3B93BLL   /*  982 */,    0xFCC41E328CAB4829LL   /*  983 */,
    0x03F38C96BA582C52LL   /*  984 */,    0xCAD1BDBD7FD85DB2LL   /*  985 */,
    0xBBB442C16082AE83LL   /*  986 */,    0xB95FE86BA5DA9AB0LL   /*  987 */,
    0xB22E04673771A93FLL   /*  988 */,    0x845358C9493152D8LL   /*  989 */,
    0xBE2A488697B4541ELL   /*  990 */,    0x95A2DC2DD38E6966LL   /*  991 */,
    0xC02C11AC923C852BLL   /*  992 */,    0x2388B1990DF2A87BLL   /*  993 */,
    0x7C8008FA1B4F37BELL   /*  994 */,    0x1F70D0C84D54E503LL   /*  995 */,
    0x5490ADEC7ECE57D4LL   /*  996 */,    0x002B3C27D9063A3ALL   /*  997 */,
    0x7EAEA3848030A2BFLL   /*  998 */,    0xC602326DED2003C0LL   /*  999 */,
    0x83A7287D69A94086LL   /* 1000 */,    0xC57A5FCB30F57A8ALL   /* 1001 */,
    0xB56844E479EBE779LL   /* 1002 */,    0xA373B40F05DCBCE9LL   /* 1003 */,
    0xD71A786E88570EE2LL   /* 1004 */,    0x879CBACDBDE8F6A0LL   /* 1005 */,
    0x976AD1BCC164A32FLL   /* 1006 */,    0xAB21E25E9666D78BLL   /* 1007 */,
    0x901063AAE5E5C33CLL   /* 1008 */,    0x9818B34448698D90LL   /* 1009 */,
    0xE36487AE3E1E8ABBLL   /* 1010 */,    0xAFBDF931893BDCB4LL   /* 1011 */,
    0x6345A0DC5FBBD519LL   /* 1012 */,    0x8628FE269B9465CALL   /* 1013 */,
    0x1E5D01603F9C51ECLL   /* 1014 */,    0x4DE44006A15049B7LL   /* 1015 */,
    0xBF6C70E5F776CBB1LL   /* 1016 */,    0x411218F2EF552BEDLL   /* 1017 */,
    0xCB0C0708705A36A3LL   /* 1018 */,    0xE74D14754F986044LL   /* 1019 */,
    0xCD56D9430EA8280ELL   /* 1020 */,    0xC12591D7535F5065LL   /* 1021 */,
    0xC83223F1720AEF96LL   /* 1022 */,    0xC3A0396F7363A51FLL   /* 1023 */
};
```