Generation of the S boxes of Tiger

Ross Anderson[1] and Eli Biham[2]

[1] Cambridge University, England; email `rja14@cl.cam.ac.uk`
[2] Technion, Haifa, Israel; email `biham@cs.technion.ac.il`

The generation algorithm of the S boxes of Tiger uses the compression function of Tiger in order to achieve random-like data for generating the random-like S boxes. The algorithm (given in Figure 1) takes a 512-bit (64-byte) randomizing parameter (which was chosen to be easy to remember) and the number of passes. It initializes the S boxes to the unity columns, and the state to the initial value of the state of Tiger. Then it randomizes each byte-column in each S box by swapping pairs of entries, each time the newly computed compression function depends on the result of all the previously swapped entries in all the S boxes and all the columns. Thus, every swap of bytes in the columns depends on the previous swaps in all the columns — not only in the same column — through the compression function.

We tried several variants of this algorithm, especially with respect to various numbers of passes, and various randomizing parameters. The resultant S boxes were compared according to the following criteria, and the one that seemed best was chosen:

1. All the entries of all the S boxes should be distinct. Moreover, no two entries should have more than three equal bytes.
2. Each byte-column is a permutation of all the 256 possible byte values.
3. The columns of all the S boxes should be as different as possible, and have some long cycles.
4. No two differences of S box entries ($S_i(t_1) \oplus S_i(t_2)$ and $S_j(t_3) \oplus S_j(t_4)$) should have more than four equal bytes.
5. The speed of the generation should not be too slow, in order to enable applications to generate the S boxes on the fly.
6. This algorithm, and the structure of the S boxes of Tiger, were chosen in a way with reduces linear and differential properties, and similarities of these properties in the four S boxes (between other things, by reducing the number of S boxes, and making them independent, unlike some original idea we had, which was intended to reduce the total memory size of the S boxes, and by choosing the large S boxes, which reduce linear and differential properties).
7. The randomizing parameter is easy to remember.

The chosen parameters to the function `gen` in Figure 1 are

1. The randomizing parameter is the title of the paper describing Tiger: "Tiger - A Fast New Hash Function, by Ross Anderson and Eli Biham".
2. The number of passes is five.

This article was processed using the LaTeX macro package with LLNCS style

```
/* This function generates the S boxes of Tiger, by calling                   */
/*  gen("Tiger - A Fast New Hash Function, by Ross Anderson and Eli Biham",5); */

/* This code is written for little-endian computers.                          */
/* small changes (indicated in the code) are required for                     */
/* big-endian computers.                                                      */

/* word64 and word32 are unsigned int/long/long long                          */
/* of the given sizes in bits.                                                */
/* The type definition might vary on various machines.                        */
typedef unsigned long long int word64;
typedef unsigned int word32;
typedef unsigned char byte;

typedef unsigned char octet[8];

extern word32 table[4*256][2];
static octet *table_ch = (octet*) table;

gen(word32 str[16], int passes)
{
  word64 state[3];
  octet *state_ch = (octet*) state;
  byte tempstr[64];

  int i, j;
  int cnt;
  int sb, col;
  int abc;

  state[0]=0x0123456789ABCDEFLL;
  state[1]=0xFEDCBA9876543210LL;
  state[2]=0xF096A5B4C3B2E187LL;

  for(j=0; j<64; j++)
    tempstr[j] = ((byte*)str)[j];
/* On big-endian computers the above line should be:                          */
/*  tempstr[j^7] = ((byte*)str)[j];                                           */

  for(i=0; i<1024; i++)
    for(col=0; col<8; col++)
      table_ch[i][col] = i&255;

  abc=2;
  for(cnt=0; cnt<passes; cnt++)
    for(i=0; i<256; i++)
      for(sb=0; sb<1024; sb+=256) {
          abc++;
          if(abc == 3) {
              abc = 0;
              tiger_compress(tempstr, state);
            }
          for(col=0; col<8; col++) {
              byte tmp = table_ch[sb+i][col];
              table_ch[sb+i][col] = table_ch[sb+state_ch[abc][col]][col];
              table_ch[sb+state_ch[abc][col]][col] = tmp;
            }
        }
}
```

Fig. 1. The S box generation algorithm