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

Tiger's S boxes are generated using its compression function to produce pseudorandom numbers that are then used to shuffle columns of bytes.

The algorithm (given in Figure 1 overleaf) takes a 512-bit randomising parameter (which was chosen to be easy to remember) and the number of passes. It initializes the S boxes so that each column is the identity permutation on bytes, and initialises the state to the randomising parameter.

Then it randomises each byte-column in each S box by swapping pairs of entries. Each time, the newly computed compression function depends on the previously swapped entries in all the S boxes. 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 with various numbers of passes and different randomizing parameters. The resulting S boxes were compared according to the following criteria, and the one that seemed best was chosen. Our goal was to control the linear and differential properties, and similarities of these properties in the four S boxes; so as well as having a relatively small number of relatively large S boxes, we wanted to make them independent.

1. Each byte-column is a permutation of all the 256 possible byte values.
2. The columns of all the S boxes should be as different as possible, and have some long cycles.
3. All the entries of all the S boxes should be distinct, and no two entries should have more than three equal bytes.
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. The randomizing parameter is easy to remember.

The parameters chosen in 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]=0xFEDCBA98765432100LL;
  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