TEA, a Tiny Encryption Algorithm

David J. Wheeler
Roger M. Needham
Computer Laboratory
Cambridge University
England

Abstract. We give a short routine which is based on a Feistel iteration and uses a large number of rounds to get security with simplicity.

Introduction

We design a short program which will run on most machines and encypher safely. It uses a large number of iterations rather than a complicated program. It is hoped that it can easily be translated into most languages in a compatible way. The first program is given below. It uses little set up time and does a weak non linear iteration enough rounds to make it secure. There are no preset tables or long set up times. It assumes 32 bit words.

Encode Routine

Routine, written in the C language, for encoding with key k[0] - k[3]. Data in v[0] and v[1].

```
void code(long* v, long* k)  {
unsigned long y=v[0],z=v[1], sum=0,   /* set up */
 delta=0x9e3779b9,   /* a key schedule constant */
 n=32 ;
while (n-->0) {            /* basic cycle start */
  sum += delta ;
    y += ((z<<4)+k[0]) ^ (z+sum) ^ ((z>>5)+k[1]) ;
    z += ((y<<4)+k[2]) ^ (y+sum) ^ ((y>>5)+k[3]) ;
              }    /* end cycle */
v[0]=y ; v[1]=z ; }
```

Basics of the routine

It is a Feistel type routine although addition and subtraction are used as the reversible operators rather than XOR. The routine relies on the alternate use of XOR and ADD to provide nonlinearity. A dual shift causes all bits of the key and data to be mixed repeatedly.

The number of rounds before a single bit change of the data or key has spread very close to 32 is at most six, so that sixteen cycles may suffice and we suggest 32.

The key is set at 128 bits as this is enough to prevent simple search techniques being effective.

The top 5 and bottom four bits are probably slightly weaker than the middle bits. These bits are generated from only two versions of z (or y) instead of three, plus the other y or z. Thus the convergence rate to even diffusion is slower. However the shifting evens this out with perhaps a delay of one or two extra cycles.

The key scheduling uses addition, and is applied to the unshifted z rather than the other uses of the key. In some tests k[0] etc. were changed by addition, but this version is simpler and seems as effective. The number delta, derived from the golden number is used where

$$\text{delta} = (\sqrt{5} - 1)2^{31}$$

A different multiple of delta is used in each round so that no bit of the multiple will not change frequently. We suspect the algorithm is not very sensitive to the value of delta and we merely need to avoid a bad value. It will be noted that delta turns out to be odd with truncation or nearest rounding, so no extra precautions are needed to ensure that all the digits of sum change.

The use of multiplication is an effective mixer, but needs shifts anyway. It was about twice as slow per cycle on our implementation and more complicated.

The use of a table look up in the cycle was investigated. There is the possibility of a delay ere one entry of the table is used. For example if k[z&3] is used instead of k[0], there is a chance one element may not be used of $(3/4)^{32}$, and a much higher chance that the use is delayed appreciably. The table also needed preparation from the key. Large tables were thought to be undesirable due to the set up time and complication.

The algorithm will easily translate into assembly code as long as the exclusive or is an operation. The hardware implementation is not difficult, and is of the same order of complexity as DES [1], taking into account the double length key.

Usage

This type of algorithm can replace DES in software, and is short enough to write into almost any program on any computer. Although speed is not a strong objective with 32 cycles (64 rounds), on one implementation it is three times as fast as a good software implementation of DES which has 16 rounds.

The modes of use of DES are all applicable. The cycle count can readily be varied, or even made part of the key. It is expected that security can be enhanced by increasing the number of iterations.

Selection of Algorithm

A considerable number of small algorithms were tried and the selected one is neither the fastest, nor the shortest but is thought to be the best compromise for safety, ease of implementation, lack of specialised tables, and reasonable performance. On languages which lack shifts and XOR it will be difficult to code. Standard C does makes an arithmetic right shift and overflows implementation dependent so that the right shift is logical and y and z are unsigned.

Analysis

A few tests were run to detect when a single change had propagated to 32 changes within a small margin. Also some loop tests were run including a differential loop test to determine loop closures. These tests failed to show any unexpected behaviour.

The shifts and XOR cause changes to be propagated left and right, and a single change will have propagated the full word in about 4 iterations. Measurements showed the diffusion was complete at about six iterations.

There was also a cycle test using up to 34 of the bits to find the lengths of the cycles. A more powerful version found the cycle length of the differential function.

$$\mathrm{d(x)=f(x\ XOR\ }2^p\mathrm{)\ XOR\ f(x)}$$

which may test the resistance to some forms of differential crypto-analysis [2].

Conclusions

We present a simple algorithm which can be translated into a number of different languages and assembly languages very easily. It is short enough to be programmed from memory or a copy. It is hoped it is safe because of the number of cycles in the encoding and length of key. It uses a sequence of word operations rather than wasting the power of a computer by doing byte or 4 bit operations.

Acknowledgements

Thanks are due to Mike Roe and other colleagues who helped in discussion and tests and to the helpful improvements suggested by the editor.

References

1 National Institute of Standards, Data Encryption Standard, Federal Information Processing Standards Publication 46. January 1977
2 E. Biham and A. Shamir, Differential Analysis of the Data Encryption Standard, Springer-Verlag, 1993
3 B. Schneier, Applied Cryptology, John Wiley & sons, New York 1994.

Appendix

Decode Routine

```
void decode(long* v,long* k)  {
 unsigned long n=32, sum, y=v[0], z=v[1],
 delta=0x9e3779b9 ;
sum=delta<<5 ;
                       /* start cycle */
while (n-->0) {
    z-= ((y<<4)+k[2]) ^ (y+sum) ^ ((y>>5)+k[3]) ;
    y-= ((z<<4)+k[0]) ^ (z+sum) ^ ((z>>5)+k[1]) ;
   sum-=delta ;  }
                       /* end cycle */
v[0]=y ; v[1]=z ;  }
```

Implementation Notes

It can be shortened, or made faster, but we hope this version is the simplest to implement or remember.

A simple improvement is to copy k[0-3] into a,b,c,d before the iteration so that the indexing is taken out of the loop. In one implementation it reduced the time by about 1/6th.

It can be implemented as a couple of macros, which would remove the calling overheads.

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