Technical Report

UCAM-CL-TR-355
ISSN 1476-2986

Number 355

Two cryptographic notes

David Wheeler, Roger Needham

November 1994

ISSN 1476-2986

A Large Block DES-like Algorithm

David Wheeler
Roger Needham
Computer Laboratory
Cambridge University
England

November 1994

Introduction

DES was designed to be slow in software. We give here a DES type of code which applies directly to single blocks comprising two or more words of 32 bits. It is thought to be at least as secure as performing DES separately on two word blocks, and has the added advantage of not requiring chaining etc. It is about 8m/(12+2m) times as fast as DES for an m word block and has a greater gain for Feistel codes where the number of rounds is greater. We use the name GDES for the codes we discuss.

The principle can be used on any Feistel code.

Ideas behind the Derivation.

The action of DES on step n is

$$\begin{gathered}\text{x=x XOR f(n,y)}\\ \text{swop x and y}\\ \text{increase n}\end{gathered}$$

where f includes the effects of the permutation and key schedule.

By using a vector rather than two registers we can express the action as

$$v_{n+1}=v_{n-1} \text{ XOR } f(n,v_n)$$

where the two words are placed in v_0 and v_1 and the encoded values taken from v_{16} and v_{17}. To obtain identical results to DES we should permute the digits of the first two words and reverse permute the last two words. Let us omit considerations of that till later.

Assumptions

If we cyclically change the key schedule to start at m and go to m+15 mod 16 then GDES remains as strong.

Extra rounds do not decrease the strength of GDES.

If extra independent elements are included in earlier stages by an addition or XOR the final encypherment is not made weaker.

If we combine two words by XOR or addition before a stage and one has undergone m rounds and the other n rounds then the combined input may be considered to have gone at least max(m,n) rounds.

The latter assumption may be partially justified by the type of argument below.

> Let $1/2 - t_n$ and $1/2 - t_m$ be the probability of a set of initial changes affecting one result digit after n or m stages. The resultant probability is
>
> $$1/2 - t_r = (1/2 + t_n)(1/2 - t_m) + (1/2 - t_n)(1/2 + t_m)$$
>
> $$\text{so } t_r = 2t_n t_m$$
> $$\text{but } t_n, t_m < 1/2$$
> $$\text{so } t_r < t_n \text{ and } t_r < t_m$$
>
> The above assumes the probabilities t_n and t_m are independent, which cannot be completely true.

We can, by these assumptions, use the modified formula

$$v_{n+1} \mathrel{+}= v_{n-1} \text{ XOR } f(n,v_n)$$

where n, n+1 and n-1 are taken modulo m.

If m=2 then n-1=n+1 and we omit n-1 from the formula.

The formula is applied for n=1 until n=2m+12 inclusive.

The block is loaded with plain text and after 2m+12 steps, the block contains the encyphered text.

We note that from the last two words of the given text to the first two words of the output 16 steps are done. In fact from any two consecutive words of the plain text to any two consecutive words of the output there are at least 16 steps of GDES, with additional non linear mixing. We may expect therefore that the large block has most of the properties of a smaller DES block and expect its safety to be similar or even better. We have been unable to prove this assertion.

It is generally thought that the initial permutation of DES gives little extra safety. That permutation can be applied to all double words if it is thought that this gives extra safety. If fact we can use it to make the two word version compatible with DES itself.

Each step is precisely reversible, so as in DES the decoding is done by doing all the steps in reverse order. The start back position of the key schedule will be different from the start position unless the block size is 2 mod 8.

Implementation

The method was tried using TEA as the Feistel code. The extra complications caused the cycle time to be doubled, so the case of m=2 was done using the original code for TEA. The final column gives times for WAKE [Wheeler 1994] using 100 word blocks for comparison purposes.

Code/Decode times per word in 1/10th microsecond units on DEC computers Nene (5000 model 133) and Ely (5400).

m	2	4	6	12	60	480	WAKE@100	Computer
	222/228	271/412	177/295	100/156	39/59	25/41	30/33	Nene
	350/344	326/518	218/351	120/193	48/75	34/52	41/48	Ely

m gives block length in words

The extra complications in the cycle caused the cycle speed to be more than halved compared with TEA. The assymptotic gain was thereby reduced from 16 to about 8.

The DES cycle which was not implemented, would scarcely be slowed by the extra work in the cycle so almost the full gain is attained. We would expect at m=4 the speed to be multiplied by 1.6, and at m=6 the speed to multiplied by 2.

Conclusions

The above method applies to all Feistel codes, and is thought not to reduce safety. The asymptotic benefit is greater for Feistel codes with more simpler rounds and can give an order of magnitude improvement in both decoding and coding.

It may be considered an extra method of chaining, with the advantage of taking less time rather than more and without the weaknesses of some of the chaining methods. In many cases it is easier for a user as he can omit considerations of chaining. When a complete block is coded, a one single bit change propagates throughout the cypher block and makes most differential attacks at least as hard as the underlying code.

It is better than DES for tamper detection but suffers equally from the fact that the same block will always be coded to the same cypher block when using the same key. However the larger block used mitigates the problem. Low entropy blocks when the number of possible messages is not large will still need extra precautions, such as extending the message by the message number.

References

E. Biham and A. Shamir, Differential Analysis of the Data Encryption Standard, Springer-Verlag, 1993

National Institute of Standards, Data Encryption Standard, Federal Information Processing Standards Publication 46. January 1977

B. Schneier, Applied Cryptology, John Wiley & sons, New York 1994.

David Wheeler, Roger Needham, TEA, This Computer Laboratory Technical Report , Cambridge UK., November 1996.

David Wheeler, WAKE, A Bulk Data Encryption Algorithm,Lecture Notes in Computer Science 809,pp 127-134 1994, Springer-Verlag

TEA, a Tiny Encryption Algorithm.

David Wheeler
Roger Needham
Computer Laboratory
Cambridge University
England

November 1994

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, n=32 ;             /* a key schedule constant */
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, taking into account the double length key.

Tests

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

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.

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.

Analysis

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.

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

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

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.

References

E. Biham and A. Shamir, Differential Analysis of the Data Encryption Standard, Springer-Verlag, 1993
National Institute of Standards, Data Encryption Standard, Federal Information Processing Standards Publication 46. January 1977
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 ;  }
```

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.