TEA

O algoritmo *Tiny Encryption Algorithm* foi desenvolvido na Universidade de Cambridge em 1994, pelos autores David Wheeler e Roger Needham. Foi apresentado no Fast Software Encryption Workshop. Não está sujeito a nenhum tipo de patente. A função de criptografia, programada na linguagem C aparece abaixo:

```
void encripta(unsigned long k[], unsigned long text[]) {
     unsigned long y= text[0], z=text[1];
     unsigned long delta =0x9e3779b9, sum=0;
     int n;
     for (n=0; n<32; n++) {
       sum = sum + delta;
       y = y + ((z << 4) + k[0]) ^ (z + sum) ^ ((z >> 5) + k[1]);
       z = z + ((y << 4) + k[2]) ^ (y + sum) ^ ((y >> 5) + k[3]);
       }
       text[0] = y; text[1] = z;
       }
```

[1]

O algoritmo TEA usa várias passagens de adição de inteiros, combinados com XOR e deslocamentos lógicos em nível de bit (<< e >>) para obter mistura e difusão dos padrões de bit no texto puro. O texto puro é um bloco de 64 bits representado como dois inteiros longos de 32 bits no vetor *Text[]*. A chave tem 128 bits de comprimento, representado como quatro inteiros de 32 bits.

Em cada uma das 32 passagens as duas metades do texto são repetidamente combinadas com as partes deslocadas da chave e uma com as outras. O uso da operação XOR e das partes deslocadas do texto proporciona a mistura e o deslocamento. A troca das duas partes do texto proporciona a difusão. A constante *delta* que não se repete, é combinada, em cada ciclo, com cada parte do texto para ocultar a chave. Isso para evitar o caso dela poder ser revelada por uma seção de texto que não varie. A função de decifração é a inversa da função de criptografia e tem o seu código abaixo:

```
void decript (unsigned long k[], unsigned long text[]) {
     unsigned long y = text[0], z=text[1];
     unsigned long delta = 0x9e3779b9, sum = delta << 5;
     int n;
     for (n=0; n<32; n++) {
       z = z - ((y << 4) + k[2]) ^ (y + sum) ^ ((y >> 5) + k[3]);
       y = y - ((z << 4) + k[0]) ^ (z + sum) ^ ((z >> 5) + k[1]);
       sum = sum - delta;
      }
       text[0] = y; text[1]=z;
    }
```

[2]

Esses programas curtos oferecem criptografia de chave secreta de forma segura e razoavelmente rápida. Ele é bem mais rápido do que o algoritmo DES e o caráter conciso do programa se presta à otimização e implementação em hardware. A chave de 128 bits é segura contra ataques de força bruta. Estudos realizados por seus autores e por outros autores revelaram apenas duas deficiências muito pequenas. [3] Na verdade, existem 3 outras chaves distintas, que geram a mesma mensagem cifrada. Nesse sentido, a longitude efetiva da chave é apenas de 126 bits (e não 128). Esta debilidade deu origem a um método que foi usado para quebrar a proteção da console Xbox da Microsoft, já que esta usava o algoritmo TEA como função *hash* para proteção. Em trabalhos posteriores, tais deficiências foram removidas no algoritmo XTEA.

A seguir, um programa C que utiliza o TEA para cifrar ou decifrar dois arquivos previamente abertos (usando *stdio.h*).

```
 while (!feof(infile)) {
   i = fread(Text,1,8,infile);
   if (i<=0) break;
   while (i<8){Text[i++]=' ';}
   switch (tipo) {
     case 1: encript(k,(unsigned long *) Text); break;
     case 2: decript(k,(unsigned long *) Text); break;   }
   j = fwrite(Text,1,8,outfile);
   }
```

[1] O símbolo << indica um *shift* à esquerda – note que não é uma rotação. O operando é deslocado à esquerda e entram 4 zeros à direita. Bits eventualmente perdidos, ficam perdidos. Exemplo: se x é 0 0 0 0 0 1 1 1, $x << 1$ é igual a 0 0 0 0 1 1 1 0. O símbolo $\otimes$, que no código C aparece como um acento circunflexo (é o seu sinal neste linguagem), significa o exclusive-or, operador binário que tem o seguinte resultado: $0 \otimes 0 = 0$, $1 \otimes 1 = 0$, $0 \otimes 1 = 1$ e $1 \otimes 0 = 1$. Finalmente, as somas acima indicadas são todas elas módulo 2^{32}.

[2] neste caso, as subtrações também são feitas em módulo 2^{32}.

[3] Wheeler, D. J. e Needham, R. M. (1997) *Tea Extensions*, Outubro de 1994, págs. 1-3.

Exemplo

Vamos criptografar a frase 'IVO VIU A UVA'. Convertendo-a em hexadecimal, fica x'49 56 4F 20 56 49 55 20 41 20 55 56 41'. Não se deve esquecer que o texto pleno deve ter tamanho múltiplo de 8, daí x'49 56 4F 20 56 49 55 20 41 20 55 56 41 20 20 20'.

Usando a chave de criptografia x'01 02 03 04 05 06 07 08 09 AA AB AC AD AE AF 00', criptografando fica x 36 25 27 D6 82 17 48 30 51 5E B4 E3 CB 4C 7B CE.

Obviamente, submetendo este string à decriptografia com a mesma chave, recupera-se a frase original, IVO VIU A UVA.

Apenas para constar, agora vai-se decriptografar o valor acima, com uma chave ligeiramente alterada (um '0' na 32^a posição da chave, virou 1, e a nova chave agora é x'01 02 03 04 05 06 07 08 09 AA AB AC AD AE AF 01').

O resultado da decriptografia ficou x'0E 2F C6 5C 0A 92 B1 10 2F CC 87 B5 D0 9F 59 F5', que obviamente nada tem a ver com a frase original (IVO VIU A UVA).

Mais dois exemplos

Bom para quem implementar o código acima.

```
senha=x'45464852545056494 74A47435854524C' (EFHRTPVIGJGCXTRL)
frase=PL I NUNCA FOI USADA
deu=x'FE954195FCAA47758FE544229D9EE4D5FC590E757F6B026E'
```

e também

```
senha=x'4D4A5855494642494E57535654594258' (MJXUIFBINWSVTYBX)
frase=SQL E PARA ACESSAR
deu=x'B41E5545CBDC3BC94A55F59CFEBA2D2F87E327F345BA26B2'
```

☞ Para você fazer

Utilize a senha (dados em hexadecimal):

504752494F4D59515343514C514E5445

Decriptografe a frase (também em hexadecimal):

0E9CEDD3FF8CEDAD11DF9242CCC48559595460D3CEF492CF

Resposta em ASCII:

Atente para o fato de que no retorno, obrigatoriamente deverão aparecer letras e não outros caracteres. Eis o subconjunto da tabela ASCII de que se fala

```
  A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
- - - - - - - - - - - - - - - - - - - - - - - - - - -
2 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 5 5 5 5 5 5 5 5 5 5
0 1 2 3 4 5 6 7 8 9 A B C D E F 0 1 2 3 4 5 6 7 8 9 A
```

Uma observação final: Note que tanto no algoritmo de criptografia quanto no de descriptografia existe uma constante numérica que obrigatoriamente tem que ter o mesmo valor em ambos os algoritmos. No original o valor é de delta =0x9e3779b9. Se você quiser, pode alterar tal valor para aquele que você quiser (desde que seja o mesmo em ambos os algoritmos). Se você mantiver tal valor escondido, ele passa a ser um nivel adicional de criptografia. Obviamente, se a criptografia for um com valor e a descriptografia com outro, o resultado será ininteligível.

Uma curiosidade Quanto tempo demoraria uma tentativa de quebra do algoritmo por "força bruta" ? Suponhamos um supercomputador capaz de examinar 1.000.000.000 de chaves por segundo. Quanto tempo este computador gastaria na tarefa ?

As chaves são $2^{126} = 8.5070 \times 10^{37}$. Se o computador examina 1 bilhão de combinações por segundo, gastará 8.5070×10^{28} segundos. Em dias, serão 9.8461×10^{23} dias. Em anos serão 2697570767123287000000 anos.