**TEA系列算法介绍**

TEA算法使用64位的明文分组和128位的密钥，它使用Feistel分组加密框架，需要进行 64 轮迭 代。该算法使用了一个神秘常数δ作为倍数，它来源于黄金比率，以保证每一轮加密都不相同。但δ的 精确值似乎并不重要，这里 TEA 把它定义为 δ=「(√5 - 1)231」（也就是程序中

的 0×9E3779B9）。之后 TEA 算法被发现存在缺陷，作为回应，设计者提出了一个 TEA 的升级版 本——XTEA（有时也被称为“tean”）。XTEA 跟 TEA 使用了相同的简单运算，但它采用了截然不同 的顺序，为了阻止密钥表攻击，四个子密钥（在加密过程中，原 128 位的密钥被拆分为 4 个 32 位的子密钥）采用了一种不太正规的方式进行混合，但速度更慢了。

**TEA算法流程图（一次加密）**

**TEA算法加密、解密代码**

// 加密代码： for(i=0;i<64;i++){

sum += delta;

v0 += ((v1<<4) + k0) ^ (v1 + sum) ^ ((v1>>5) + k1); v1 += ((v0<<4) + k2) ^ (v0 + sum) ^ ((v0>>5) + k3);

}

// 简化加密：

v0' = v0 + fun1(v1); v1' = v1 + fun2(v0');

// 简化解密代码：

v1 = v1' - fun2(v0');

v0 = v0' - fun1(v1'); // 解密代码：

sum = delta \* 64; for(i=0;i<64;i++){

v1 -= ((v0<<4) + k2) ^ (v0 + sum) ^ ((v0>>5) + k3); v0 -= ((v1<<4) + k0) ^ (v1 + sum) ^ ((v1>>5) + k1); sum -= delta;

}

**TEA基于c语言的解密脚本**

#include <iostream> using namespace std;

int main() {

int data[6] = { 3746099070,550153460,3774025685,1548802262,2652626477,2230518816 };

unsigned int tmp1, tmp2; unsigned int v3, v4;

int key[4] = { 2,2,3,4 };

for (int i = 0; i < 5; i += 2) { v3 = data[i];

v4 = data[i + 1];

int v5 = 1166789954 \* 64; for (int j = 0; j < 64; ++j) {

v4 -= (v3 + v5 + 20) ^ ((v3 << 6) + key[2]) ^ ((v3 >> 9) + key[3]) ^ 0x10;

v3 -= (v4 + v5 + 11) ^ ((v4 << 6) + key[0]) ^ ((v4 >> 9) + key[1]) ^ 0x20;

v5 -= 1166789954; }

data[i] = v3; data[i + 1] = v4;

}

for (unsigned int i = 0; i < 6; i++)

printf("%c%c%c", \*((char\*)&data[i] + 2), \*((char\*)&data[i] + 1), \*(char\*)&data[i]);

}

**XTEA算法相对TEA算法的变化**

1、由之前的((v1<<4) + k0) ^ (v1>>5)) + k1 变成了((v1<<4) ^ (v1>>5) + v1)，此时 v1内部数据不再受到密钥的影响。

2、原先的 v1 + sum 变成了 (sum + key[sum & 3]) 以及 sum + key[sum>>2 & 3]，密钥 变成了轮转使用，而不是固定只

针对某种数据进行加密（解密）。并且此时密钥的选取收到了sum的影响。 3、sum += delta的位置变到了v0，v1两个block加密的中间。

**XTEA算法加密、解密代码**

// 加密代码：

void encipher(unsigned int num\_rounds, uint32\_t v[2], uint32\_t const key[4]) {

unsigned int i;

uint32\_t v0=v[0], v1=v[1], sum=0, delta=0x9E3779B9; for (i=0; i < num\_rounds; i++) {

v0 += (((v1 << 4) ^ (v1 >> 5)) + v1) ^ (sum + key[sum & 3]); sum += delta;

v1 += (((v0 << 4) ^ (v0 >> 5)) + v0) ^ (sum + key[(sum>>11) & 3]); }

v[0]=v0; v[1]=v1; }

// 解密代码：

void decipher(unsigned int num\_rounds, uint32\_t v[2], uint32\_t const key[4]) {

unsigned int i;

uint32\_t v0=v[0], v1=v[1], delta=0x9E3779B9, sum=delta\*num\_rounds; for (i=0; i < num\_rounds; i++) {

v1 -= (((v0 << 4) ^ (v0 >> 5)) + v0) ^ (sum + key[(sum>>11) & 3]); sum -= delta;

v0 -= (((v1 << 4) ^ (v1 >> 5)) + v1) ^ (sum + key[sum & 3]); }

v[0]=v0; v[1]=v1; }

**XXTEA算法**

特点：在可变长度块上运行，这些块是32位大小的任意倍数（最小64位），使用128位密钥, 是目前 TEA系列中最安全的算法，但性能较上两种有所降低。

**XXTEA算法流程图**

1、可以利用python自带的xxtea模块进行解密 2、基于c语言的解密代码如下：

#include <stdio.h> #include <stdint.h> #define DELTA 0x9e3779b9

#define MX (((z>>5^y<<2) + (y>>3^z<<4)) ^ ((sum^y) + (key[(p&3)^e] ^ z)))

void btea(uint32\_t \*v, int n, uint32\_t const key[4]) {

uint32\_t y, z, sum; unsigned p, rounds, e;

if (n > 1) /\* Coding Part \*/ {

rounds = 6 + 52/n; sum = 0;

z = v[n-1]; do

{

sum += DELTA;

e = (sum >> 2) & 3; for (p=0; p<n-1; p++) {

y = v[p+1];

z = v[p] += MX; }

y = v[0];

z = v[n-1] += MX; }

while (--rounds); }

else if (n < -1) /\* Decoding Part \*/ {

n = -n;

rounds = 6 + 52/n; sum = rounds\*DELTA; y = v[0];

do {

e = (sum >> 2) & 3; for (p=n-1; p>0; p--) {

z = v[p-1];

y = v[p] -= MX; }

z = v[n-1];

y = v[0] -= MX; sum -= DELTA;

}

while (--rounds); }

}

int main() {

uint32\_t v[4]= {0x73647979,0x726b6f5f,0x646f675f,0x0}; uint32\_t const k[4]= {0X95C4C,0X871D,0X1A7B7,0X12C7C7}; int n= 2; //n的绝对值表示v的长度，取正表示加密，取负表示解密 // v为要加密的数据是两个32位无符号整数

// k为加密解密密钥，为4个32位无符号整数，即密钥长度为128位 printf("加密前原始数据：%s\n",(char\*)v);

btea(v, n, k);

printf("加密后的数据：%u %u %u\n",v[0],v[1],v[3]); btea(v, -n, k); printf("解密后的数据：%s\n",(char\*)v);

return 0; }