1. xxtea算法

（1）Python库xxtea的使用

>>>importos>>>importxxtea>>>importbinascii>>>>>>key=os.urandom(16)# Key must be a 16-byte string.>>>s=b"xxtea is good">>>>>>enc=xxtea.encrypt(s,key)>>>dec=xxtea.decrypt(enc,key)>>>s==decTrue>>>>>>hexenc=xxtea.encrypt\_hex(s,key)>>>hexencb'7ad85672d770fb5cf636c49d57e732ae'>>>s==xxtea.decrypt\_hex(hexenc,key)True>>>>>>binascii.hexlify(enc)==hexencTrue

>>>hexenc=binascii.hexlify(xxtea.encrypt(s,key))>>>s==xxtea.decrypt(binascii.unhexlify(hexenc),key)True



网址

<https://www.cnpython.com/pypi/xxtea>

print(xxtea.decrypt(bytes(m),key,padding=False))

1. tea，xtea，xxtea算法分析

Tea

#include <stdio.h>

#include <stdint.h>

//加密函数

void encrypt (uint32\_t\* v, uint32\_t\* k) {

uint32\_t v0=v[0], v1=v[1], sum=0, i; /\* set up \*/

uint32\_t delta=0x9e3779b9; /\* a key schedule constant \*/

uint32\_t k0=k[0], k1=k[1], k2=k[2], k3=k[3]; /\* cache key \*/

for (i=0; i < 32; i++) { /\* basic cycle start \*/

sum += delta;

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

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

} /\* end cycle \*/

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

}

//解密函数

void decrypt (uint32\_t\* v, uint32\_t\* k) {

uint32\_t v0=v[0], v1=v[1], sum=0xC6EF3720, i; /\* set up \*/

uint32\_t delta=0x9e3779b9; /\* a key schedule constant \*/

uint32\_t k0=k[0], k1=k[1], k2=k[2], k3=k[3]; /\* cache key \*/

for (i=0; i<32; i++) { /\* basic cycle start \*/

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

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

sum -= delta;

} /\* end cycle \*/

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

}

int main()

{

uint32\_t v[2]={1,2},k[4]={2,2,3,4};

// 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：%u %u\n",v[0],v[1]);

encrypt(v, k);

printf("加密后的数据：%u %u\n",v[0],v[1]);

decrypt(v, k);

printf("解密后的数据：%u %u\n",v[0],v[1]);

return 0;

}

Xtea

#include <stdio.h>

#include <stdint.h>

/\* take 64 bits of data in v[0] and v[1] and 128 bits of key[0] - key[3] \*/

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;

}

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;

}

int main()

{

uint32\_t v[2]={1,2};

uint32\_t const k[4]={2,2,3,4};

unsigned int r=32;//num\_rounds建议取值为32

// 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：%u %u\n",v[0],v[1]);

encipher(r, v, k);

printf("加密后的数据：%u %u\n",v[0],v[1]);

decipher(r, v, k);

printf("解密后的数据：%u %u\n",v[0],v[1]);

return 0;

}

Xxtea

#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[2]= {1,2};

uint32\_t const k[4]= {2,2,3,4};

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：%u %u\n",v[0],v[1]);

btea(v, n, k);

printf("加密后的数据：%u %u\n",v[0],v[1]);

btea(v, -n, k);

printf("解密后的数据：%u %u\n",v[0],v[1]);

return 0;

}

网址 <https://blog.csdn.net/gsls200808/article/details/48243019>

Xxtea python实现

import struct

\_DELTA = 0x9E3779B9

def \_long2str(v, w):

n = (len(v) - 1) << 2

if w:

m = v[-1]

if (m < n - 3) or (m > n): return ''

n = m

s = struct.pack('<%iL' % len(v), \*v)

return s[0:n] if w else s

def \_str2long(s, w):

n = len(s)

m = (4 - (n & 3) & 3) + n

s = s.ljust(m, "\0")

v = list(struct.unpack('<%iL' % (m >> 2), s))

if w: v.append(n)

return v

def encrypt(str, key):

if str == '': return str

v = \_str2long(str, True)

k = \_str2long(key.ljust(16, "\0"), False)

n = len(v) - 1

z = v[n]

y = v[0]

sum = 0

q = 6 + 52 // (n + 1)

while q > 0:

sum = (sum + \_DELTA) & 0xffffffff

e = sum >> 2 & 3

for p in xrange(n):

y = v[p + 1]

v[p] = (v[p] + ((z >> 5 ^ y << 2) + (y >> 3 ^ z << 4) ^ (sum ^ y) + (k[p & 3 ^ e] ^ z))) & 0xffffffff

z = v[p]

y = v[0]

v[n] = (v[n] + ((z >> 5 ^ y << 2) + (y >> 3 ^ z << 4) ^ (sum ^ y) + (k[n & 3 ^ e] ^ z))) & 0xffffffff

z = v[n]

q -= 1

return \_long2str(v, False)

def decrypt(str, key):

if str == '': return str

v = \_str2long(str, False)

k = \_str2long(key.ljust(16, "\0"), False)

n = len(v) - 1

z = v[n]

y = v[0]

q = 6 + 52 // (n + 1)

sum = (q \* \_DELTA) & 0xffffffff

while (sum != 0):

e = sum >> 2 & 3

for p in xrange(n, 0, -1):

z = v[p - 1]

v[p] = (v[p] - ((z >> 5 ^ y << 2) + (y >> 3 ^ z << 4) ^ (sum ^ y) + (k[p & 3 ^ e] ^ z))) & 0xffffffff

y = v[p]

z = v[n]

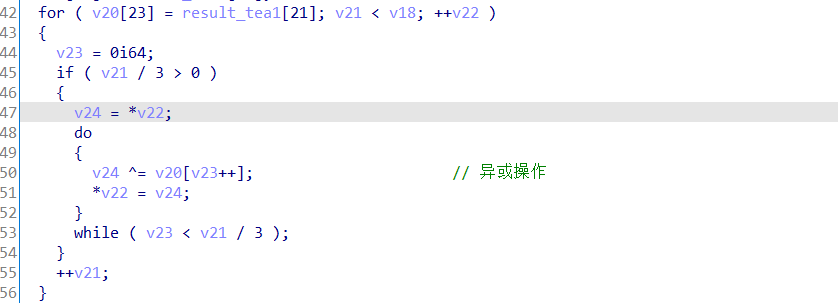
v[0] = (v[0] - ((z >> 5 ^ y << 2) + (y >> 3 ^ z << 4) ^ (sum ^ y) + (k[0 & 3 ^ e] ^ z))) & 0xffffffff

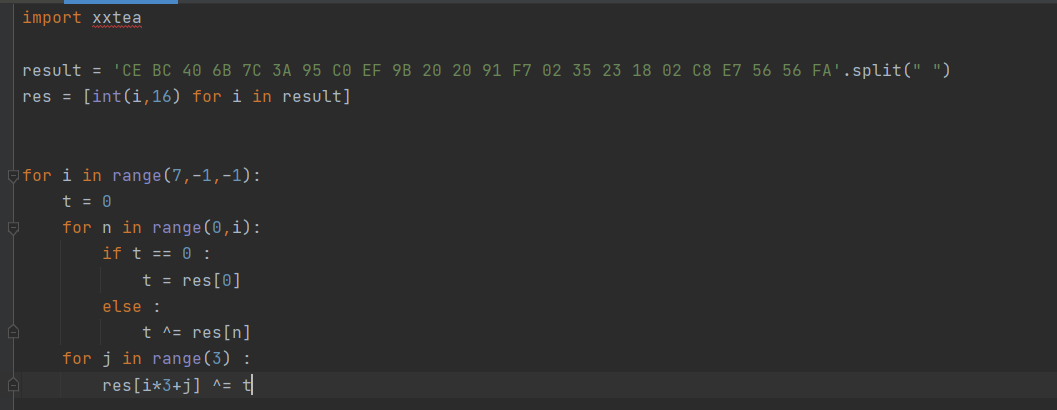
y = v[0]

sum = (sum - \_DELTA) & 0xffffffff

return \_long2str(v, True)

1. 复杂算法分析之异或

逆向

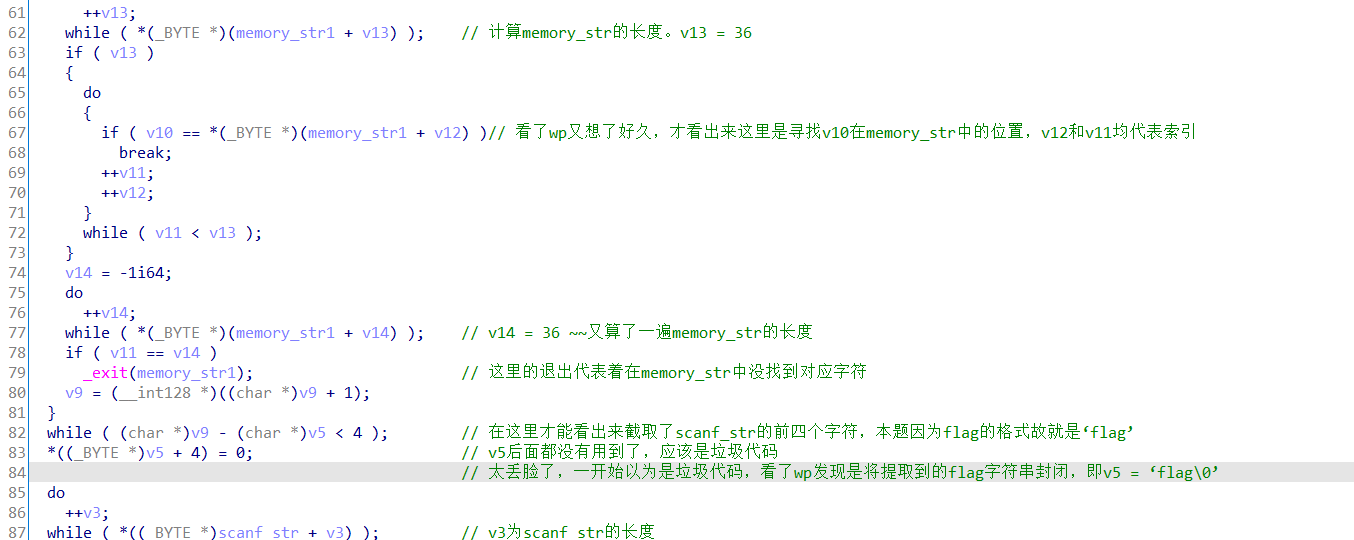


1. 复杂算法分析之扩充



1. 复杂算法分析之找索引

虽然这块的算法很复杂值得学习一下，但是注意本题中是垃圾代码



# new operator

