[HKUST Firebird CTF 2024]
Random Secure Algorithm (II)

Description:

According to chall-2.py, we can find that: (same as Random Secure Algorithm (I) LOLLL)

```
e = 0x10001 # given
rsa.n # is given
secret = [os.urandom(8) for _ in range(10)] # not given
iv = os.urandom(16) # is given
enc_flag = iv + AES.new(key =
hashlib.sha256(b''.join(secret)).digest(), mode = AES.MODE_CBC, iv =
iv).encrypt(pad(FLAG, 16))
# encrypted flag is given
enc_secret = []
for block in secret:
    enc_secret.append(rsa.encrypt(int.from_bytes(block, 'big')))
enc_s = ':'.join(hex(c)[2:].rjust(1024 // 8 * 2, '0') for c in
enc_secret)
# 10 encrypted secrets are given
```

AND:

```
def dot_mod(c1, c2, n):
    assert len(c1) == len(c2)
    return sum(i * j for i, j in zip(c1, c2)) % n

ct = input('🔒 ').strip().replace('-', '').split(':')
pt = [rsa.decrypt(int(c, 16)) for c in ct]

weights = enc_secret # the ONLY different line.. Oh..
random.shuffle(weights)

res = dot_mod(pt, weights, rsa.n)
print(f'📄 {hex(res)[2:]}')
```

Which means we can perform chosen ciphertext attack as we are allowed to decrypt our ciphertext and receive the plaintext.

However, we need to send exactly 10 ciphertexts and they would be decrypted then multiplied by random weights which is in enc_secret, and added together then mod n

Solution:

Our target is to find the 10 secret keys from cracking this challenge's RSA cryptosystem.

Let's take a look at `enc_secret` and the decryption first, we can get these formula:

$$\varepsilon(s) = [{s_1}^e, {s_2}^e, {s_3}^e, \dots, {s_{10}}^e]\ (mod\ n)$$

$$res = sum([{c_1}^d, {c_2}^d, {c_3}^d, \dots, {c_{10}}^d](mod\ n) * RandOrder([{s_1}^e, {s_2}^e, {s_3}^e, \dots, {s_{10}}^e))(mod\ n)$$

It is obvious that we can easily apply **Fermat's little theorem** with chosen ciphertext attack, let's see what happen if we send `enc_secret` directly:

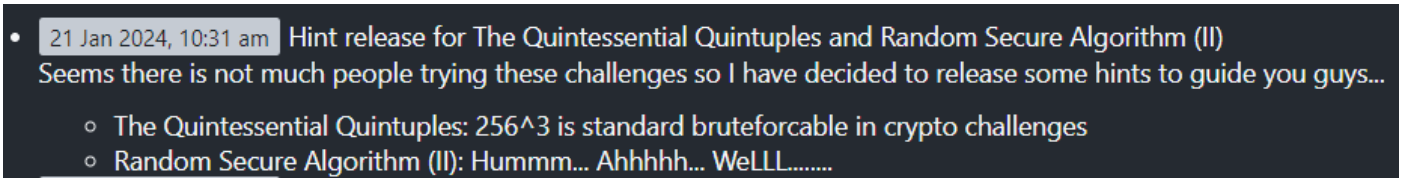
$$sent = [{s_1}^e, {s_2}^e, {s_3}^e, \dots, {s_{10}}^e](mod\ n)$$

$$res = sum([{s_1}^{ed}, {s_2}^{ed}, {s_3}^{ed}, \dots, {s_{10}}^{ed}](mod\ n) * weights)(mod\ n)$$
$$\rightarrow res = sum([s_1, s_2, s_3, \dots, s_{10}](mod\ n) * weights)(mod\ n)$$

However, the secret keys are mixed up since they are added together.

After looking more carefully, you should also find that value of weights are NOW larger than n. Because the weights are random ordered encrypted secret keys, so we can't perform the same attack from Random Secure Algorithm (I) !!! GG !!

What if there is a THING that can filter out res to get secret keys?
and then there was a hint:



LOLLL, what kind of hint is that.. WAIT, LLL? Lenstra–Lenstra–Lovász lattice basis reduction algorithm? I have heard of that!

Simply put, LLL algorithm can help you to find short vectors in a matrix (I can't explain it much further caz I suck at linear algebra :P)

For example, if you have ONE secret key s only, the equation (sending s^e) will be:

$$res = (s_1(mod\ n) * s_1^e)(mod\ n) \rightarrow\ (s_1 * s_1^e) + C * n - res = 0$$

your matrix should be originally: $\begin{matrix} s_1^e & 1 & 0 \\ -res & 0 & 1 \\ n & 0 & 0 \end{matrix} \rightarrow after\ LLL \rightarrow \begin{matrix} C & 0 & s_1 \\ ? & ? & ? \\ ? & ? & ? \end{matrix}$

LLL will turn matrix into reduced form which all the keys would be presented on the first row.

Therefore, in 10 secret keys version:

$$res = s_1 * {s_{b_1}}^e + s_2 * {s_{b_2}}^e + s_3 * {s_{b_3}}^e + \cdots + s_{10} * {s_{b_{10}}}^e (mod\ n)$$

```
[[n, 0, 0, 0, 0, 0 ,0 ,0 ,0 ,0 , 0, 0],
 [-res, 1, 0, 0, 0, 0 ,0 ,0 ,0 ,0 ,0, 0],
 [enc_s[0], 0, 1, 0, 0, 0 ,0 ,0 ,0 ,0 ,0,0],
 [enc_s[1], 0, 0, 1, 0, 0 ,0 ,0 ,0 ,0 ,0,0],
 [enc_s[2], 0, 0, 0, 1, 0 ,0 ,0 ,0 ,0 ,0,0],
 [enc_s[3], 0, 0, 0, 0, 1 ,0 ,0 ,0 ,0 ,0,0],
 [enc_s[4], 0, 0, 0, 0, 0 ,1 ,0 ,0 ,0 ,0,0],
 [enc_s[5], 0, 0, 0, 0, 0 ,0 ,1 ,0 ,0 ,0,0],
 [enc_s[6], 0, 0, 0, 0, 0 ,0 ,0 ,1 ,0 ,0,0],
 [enc_s[7], 0, 0, 0, 0, 0 ,0 ,0 ,0 ,1 ,0,0],
 [enc_s[8], 0, 0, 0, 0, 0 ,0 ,0 ,0 ,0 ,1,0],
 [enc_s[9], 0, 0, 0, 0, 0 ,0 ,0 ,0 ,0 ,0,1]]
```

But… do we need to make the LLL algorithm? Will it be wasting TONS of time?

DON'T WORRY! Sagemath can do LLL algorithm for you :D

https://doc.sagemath.org/html/en/reference/matrices/sage/matrix/matrix_integer_dense.html

Now we have the 10 secret keys on hand, so what to do next is to match each secret key to each item in enc_secret, and then do aes.decrypt the flag with cracked key and given iv and given enc_flag (I did it manually as I m lazy to split the key by writing more codes :P)

BTW if you are interested of LLL or how I get this idea, here is an blog post on lattice-based cryptography for an in-depth mathematical context: https://www.klwu.co/maths-in-crypto/lattice-based-cryptography-basics/

Solve Scripts:

```
ffrom sage.all import *
import itertools
#from z3 import * <- you can guess what I was trying before lol
from pwn import *
e = 0x10001
io = remote('ash-chal.firebird.sh', int(36009))
io.recvuntil('🏁 '.encode())
enc_flag = int(io.recvline().decode().strip(),16)
io.recvuntil('📣 '.encode())
n = int(io.recvline().decode().strip(),16)
io.recvuntil('🔍 '.encode())
enc_s = [int(i,16) for i in io.recvline().decode().strip().split(':')]
heheboi = []
for block in enc_s:
    heheboi.append(block)

tosend = ':'.join(hex(c)[2:].rjust(1024 // 8 * 2, '0') for c in heheboi)
io.sendlineafter('🔒 '.encode(),tosend.encode())
io.recvuntil('📜 '.encode())
what7usaid = io.recvline().decode().strip()
what7usaid = int(what7usaid,16)
io.close()
A = matrix(ZZ,[[n, 0, 0, 0, 0, 0 ,0 ,0 ,0 , 0, 0],
               [-what7usaid, 1, 0, 0, 0, 0 ,0 ,0 ,0 ,0, 0],
               [enc_s[0], 0, 1, 0, 0, 0 ,0 ,0 ,0 ,0 ,0,0],
               [enc_s[1], 0, 0, 1, 0, 0 ,0 ,0 ,0 ,0 ,0,0],
               [enc_s[2], 0, 0, 0, 1, 0 ,0 ,0 ,0 ,0 ,0,0],
               [enc_s[3], 0, 0, 0, 0, 1 ,0 ,0 ,0 ,0 ,0,0],
               [enc_s[4], 0, 0, 0, 0, 0 ,1 ,0 ,0 ,0 ,0,0],
               [enc_s[5], 0, 0, 0, 0, 0 ,0 ,1 ,0 ,0 ,0,0],
               [enc_s[6], 0, 0, 0, 0, 0 ,0 ,0 ,1 ,0 ,0,0],
               [enc_s[7], 0, 0, 0, 0, 0 ,0 ,0 ,0 ,1 ,0,0],
               [enc_s[8], 0, 0, 0, 0, 0 ,0 ,0 ,0 ,0 ,1,0],
               [enc_s[9], 0, 0, 0, 0, 0 ,0 ,0 ,0 ,0 ,0,1]]).LLL()
for i in A:
    for j in i:
        if j < 0:
            print('bruh')
        else:
            if (len(hex(j)[2:])) <= 17:
```

```
                print(j,len(hex(j)[2:]))
print(hex(n)[2:])
print("flag: ",enc_flag)
print("enc_s: ",enc_s)
```

```
from Crypto.Util.number import getPrime
from Crypto.Cipher import AES
from Crypto.Util.Padding import pad
import signal
import hashlib
import os
x1 = 7979550014458264565
x2 = 11691911367047039438
x3 = 5889260536131842307
x4 = 17289539282490345797
x5 = 8917379675744562127
x6 = 17975411919254932675
x7 = 15908186223886587487
x8 = 17859823020892429648
x9 = 12407431723241681591
x10 =11190209786029676191

x_list = [x10,x9,x8,x7,x6,x5,x4,x3,x2,x1]
e = 0x10001
n =
0x90ef4c38f3aa15bb6f5fa2750514c363cb20c97e4bcc33c9e023e659205e7e518dc52c8b848a4814a17498
4c89f043b7460766cae35c2a92c7291f6784716d0aaec4614764d8dba7a1a085b2b27b4c9943c4ef20bde544
a5a42e956e1d32af9a80e6c424e25ddf68f8da38a94e7a6654808a114f18504be309cecb4053a4cd2d
flag=  70319560373009947819110556268403662797113639939967622664093306843255193190574000 7
97518911018633725799767362106746160120818413566919850669791049887320103562848668102986 08
946446793342969335361449060427153314502775009180318995970804 07

iv =
int.to_bytes(int('7539922ac7323dbbdfbc4907ae48badfc107aba0017ef8f2373b0b3d12c5c3dffe6ca4
34362e23567527a0c7a72edd03abdff6447202712d27fd1cd6fe336055d8c636f19fbd60a2c471156b94a629
5c84bb099b077e6473aff6c6c441beaa1c'[:32],16),16,'big')
enc_s =
[569159265509414370271400734725277313782657771291552812247412173744993545687972476796052
873592486793946319807224271569377026999710992112699282683742614404052493988299156642240 7
598292419751314787818936424254177597876437192313478237146780402234150390092400470419982 8
384746062244949143083559479997250798265816395,
541859061334612728147559027541399711621333039452224930166456761137531565921811492936919 0
```

```
701381533133828985430644732506981332440543183636255165785221286611077437367292841740899429917124720417319387395681276410856723101492061611448464289028759392623824902763752799768245760542469200893727429223907990972881234 9,
832862017798622702437006549941090192818172744374138086896711831403738279312539983893001413563618440152375804880462991426569255653218076683354450017937636377120600792391148141677709291267275190336213623568758114574252122151399483901836973014378265018980410845329497851251732598723406119243832928635010926558 52,
683716157540723591598647299050544588375220075134242555537562570152232945276782529401582767978222768843923463184850978906536441695180133998262140960570614521022730684282002529371358357297458249391449252304514768209614819969804230878859132258421660581202225699358192879443387152650374675007277055829817109107 14,
732292793764489972220230953557082791339536203502834793023005615991027622465146728049556351463596371752979785279355528651304881367915293576251480845167305030495901028497044856093216835386246274743519877643188220967316356371882516691568573663796500139966014954909909061622884794024656543488393286211108776609 09,
261770248432269062233604165590425211018767886436145621051711634014988014446082744064835076980869802283757699866172075977696107765832906309214204704371115207134329986421116459185307170343606896257183461073484124808420688823862822433979260892146778051474356687158798679068447292816802573652027234494296787848 18,
858988391320228521179680307540716208695638124902543311534331673335558841381897593135552116591400502672275179179233283962712664688231222315371588227731958198848305114575362570960118072831153031415425796733214534423822327489311671243708685223102757786263092323571643712842224296026044658825815654334094323056 02,
271535883821138570039119913826147357583547106270957441184975348983562027785667512555254592546591827182777295412285523613612804271062391816083181529157896702165580112864577733287757885498277754687490099254588309779556713795228077414527882585034058010068474665638985027678634030139475024206267886777570662097 02,
616696337389352743125007812163489010934548209088873029364488150477700025841534520030518385664763106301462168518065695048374098618541533965247483469334612519124429041339196363800522923743566953973364200089601751954807455527308460598275112291102503088599061022906718682878480585533881640031022560537762180660 52,
965086165768048129946991742192817651497256526865023538588904125301934966436004416800446663383052668634581496275085783472637142908534605766382326718523264295948233145835341898622926725876990180389741265453991502528818783397665938953639619712432132921757419050421257715468821986073342096221221064067562648709 05]
#weights = list(range(1,11))[::-1]
#print(weights)
secret = [0,0,0,0,0,0,0,0,0,0]

def check(item):
    if pow((item ),e,n) in enc_s:
        print("ok!")
```

```
        secret[enc_s.index(pow((item),e,n))] = int.to_bytes(item,8,'big')
        return
    print("bruh")
    return
for x in x_list:
    check(x)
the_flag = AES.new(key = hashlib.sha256(b''.join(secret)).digest(), mode = AES.MODE_CBC,
iv =
iv).decrypt(int.to_bytes(int('7539922ac7323dbbdfbc4907ae48badfc107aba0017ef8f2373b0b3d12
c5c3dffe6ca434362e23567527a0c7a72edd03abdff6447202712d27fd1cd6fe336055d8c636f19fbd60a2c4
71156b94a6295c84bb099b077e6473aff6c6c441beaa1c'[32:],16),80,'big'))
print(the_flag)
```

Thank you for watching!