h4x0rpsch0rr (/)

Blog (/blog/) About (/about/) Contact (/contact/)

Sat, 08 March 2014 ~ yyyyyyy

RuCTF Quals 2014: Crypto 500 "decrypt message" writeup

This challenge required solvers to perform a related message attack on RSA. The task description is as follows:

> Two agents, Alex and Jane, have simultaneously known very secret message and transmitted it to Center. You know following:
>
> 1. They used RSA with this *(see below)* public key
> 2. They sent exactly the same messages except the signatures (name appended, eg. "[message]Alex")
> 3. They did encryption this way:
>
> ```
> c, = pubKey.encrypt(str_to_num(message), 1) # using RSA from Crypto.PublicKey
> c = num_to_str(c).encode('hex')
> ```
>
> 4. And here are cryptograms you have intercepted:
>
> "61be5676e0f8311dce5d991e841d180c95b9fc15576f2ada0bc619cfb991cddfc51c4dcc5ecd150d7176c835449b5ad085abec38898be02d2749485b68378a8742544ebb8d6dc45b58fb9bac4950426e3383fa31a933718447decc5545a7105dcdd381e82db6acb72f4e335e244242a8e0fbbb940edde3b9e1c329880803931c"
>
> "9d3c9fad495938176c7c4546e9ec0d4277344ac118dc21ba4205a3451e1a7e36ad3f8c2a566b940275cb630c66d95b1f97614c3b55af8609495fc7b2d732fb58a0efdf0756dc917d5eeefc7ca5b4806158ab87f4f447139d1daf4845e18c8c7120392817314fec0f0c1f248eb31af153107bd9823797153e35cb7044b99f26b0"
>
> Now tell me that secret message! (The answer for this task starts from 'ructf_')

The public key was given as

```
-----BEGIN PUBLIC KEY-----
MIGeMA0GCSqGSIb3DQEBAQUAA4GMADCBiAKBgQCjX+QVVbBrI812miqtd8rTo9qm
p23nWRyLjyga+lElKX+xBUE4f4uZjS/Rp2Eg3RRygaxSCOpS0+ytHj58q1wNskfd
+HzYrcOtE7+1ceJtLhf/okKagLfp299AVIRf0iQq4HH+GhldKJAO2kBdo+k3yinf
8oTgUow9tRDeqcczvwICMAE=
-----END PUBLIC KEY-----
```

which decodes to

> n = 0xa35fe41555b06b23cd769a2aad77cad3a3daa6a76de7591c8b8f281afa5125297fb10541387f8b998d2fd1a76120dd147281ac5208ea52d3ecad1e3e7cab5c0db247ddf87cd8adc3ad13bfb571e26d2e17ffa2429a80b7e9dbdf4054845fd2242ae071fe1a195d28900eda405da3e937ca29dff284e0528c3db510dea9c733bf
>
> e = 0x3001

Let's try to collect potentially useful information. We know…

- …the public key (n, e) used to encrypt the plaintexts p_0, p_1.
- …the two plaintexts' difference (that is, $\delta := (p_1 - p_0) \bmod n$).
- …the ciphertexts $c_0 = m_0^e \bmod n$, $c_1 = m_1^e \bmod n$.

Some quick Google-Fu yielded the paper "Low-Exponent RSA with Related Messages (http://www.cs.unc.edu/~reiter/papers/1996/Eurocrypt.pdf)" (D. Coppersmith et al.) that describes an attack on settings like these. It says, among other things (note that the following talks about $e = 5$, $\delta = 1$):

> Let z denote the unknown message m. Then z satisfies the following two polynomial relations:
>
> $$z^5 - c_1 = 0 \quad \text{mod } N$$
> $$(z+1)^5 - c_2 = 0 \quad \text{mod } N$$
>
> where the c_i are treated as known constants. Apply the Euclidean algorithm to find the greatest common divisor of these two univariate polynomials over the ring Z/N:
>
> $$\gcd\,(z^5 - c_1, (z+1)^5 - c_2) \in \mathrm{Z}/N[z].$$
>
> This should yield the linear polynomial $z - m$ (except possibly in rare cases).

This means: Assuming the given ciphertexts are *not* an instance of "*rare cases*", the greatest common divisor $d \in (\mathrm{Z}/n\mathrm{Z})[X]$ of $X^e - c_0$ and $(X + \delta)^e - c_1$ equals $X - m$ multiplied by some constant in $\mathrm{Z}/n\mathrm{Z}$ (as d is only unique up to multiplication by units). Dividing d by its lead coefficient will result in $X - m$.

Using my Python algebra library (that, by the way, only came into existence while solving this challenge since I was unable to find packages that could properly handle polynomials over arbitrary rings), the required computation is easily implemented:

```
from algebra.ring.algorithm import euclid
from algebra.ring.mod import mod
from algebra.ring.poly import poly
from Crypto.PublicKey import RSA
import binascii

with open('key.pub', 'rb') as f:
    key = RSA.importKey(f.read())

c0 = int('61be5676e0f8311dce5d991e841d180c95b9fc15576f2ada0bc619cfb991cddfc51c4dcc5ecd150d7176c835449b5ad085abec
38898be02d2749485b68378a8742544ebb8d6dc45b58fb9bac4950426e3383fa31a933718447decc5545a7105dcdd381e82db6acb72f4e33
5e244242a8e0fbbb940edde3b9e1c329880803931c', 16)
c1 = int('9d3c9fad495938176c7c4546e9ec0d4277344ac118dc21ba4205a3451e1a7e36ad3f8c2a566b940275cb630c66d95b1f97614c
3b55af8609495fc7b2d732fb58a0efdf0756dc917d5eeefc7ca5b4806158ab87f4f447139d1daf4845e18c8c7120392817314fec0f0c1f24
8eb31af153107bd9823797153e35cb7044b99f26b0', 16)

delta  = int(binascii.hexlify(b'Jane'), 16) - int(binascii.hexlify(b'Alex'), 16)

Zn = mod(key.n)
ZnX = poly(Zn)

delta = Zn(delta)

#f = ZnX.X ** key.e - ZnX(R(c0)) #exponentiation is slow...
f = ZnX([Zn.zero] * key.e + [Zn.one], 'little') - ZnX(Zn(c0), 'little')

#g = (ZnX.X + ZnX(delta)) ** key.e - ZnX(Zn(c1)) #exponentiation is slow...
gs = [delta ** key.e]
for k in range(key.e):
    gs.append(Zn(key.e - k) / (Zn(k + 1) * delta) * gs[-1])
g = ZnX(gs, 'little') - ZnX(Zn(c1))

d = euclid(f, g)[0]
d /= ZnX(d.lc())

print(bytes(-d.get(0)))
```

Some tests with smaller constants suggested the program would run pretty long, but as there was plenty of time left, I gave it a shot. About an hour later, the solver printed the flag:

```
$ ./solve.py
b'The key is RUCTF_StandBackImGonnaDoMath. Alex'
```