Millerkjc [tuctf-2018:crypt] RSAyyyy - wp Latest commit 4abf535 on Nov 28, 2018

..

README.md [tuctf-2018:crypt] RSAyyyy - wp 3 months ago

README.md

RSAyyyy [Reverse 358 points]

Chall's description

```
This challenge is designed to give an overview
of the RSA algorithm. If you have a team member
that is less familiar with RSA that wants to be,
give this challenge to them. This might be useful.

nc 3.16.57.250 12345
```

Link

RSA_link

Solution

This is a "educational" chall; we enjoyed to do it to `refresh` our RSA knowledge. There are multiple tasks that help you to understand how RSA works (with some math's help)

Level 1: Calculating

```
p = 2800941491
q = 4071798539
What is n?
```

Answer

```
11404869470878281649
```

Explanation

From the formula n = p*q

Level 2: Calculating m

```
message = "hover exculpatory broadminded Bromley constructible"
What is m?
```

Answer

```
269678299779785121551311846594485190570778179799456098570707555275403511188378404406315707027746721729831218

Congratulations! You beat Level 2!

Now, we are going to actually calculate ciphertext.
```

Explanation(Code)

```
message = "hover exculpatory broadminded Bromley constructible"

''.join([hex(ord(x))[2:] for x in message])            # <--- hex value
int(''.join([hex(ord(x))[2:] for x in message]),16)    # <--- int value
```

Level 3: Calculating c

```
p = 2871875029
q = 3482439599
What is n?
#1              <===

Ayyyyy

e = 65537
m = 7089056601674313572
What is c?
#2              <===

Ayyyyy

Congratulations! You beat Level 3!

In order for RSA to be asymmetrical,
the private exponent, d, needs to be calculated.
```

Answer and Explanation

```
#1
For the formula n = p*q
n = 2871875029 * 3482439599
  = 10001131324368873371

#2
m^e mod(n) = (7089056601674313572 ** 65537) % 10001131324368873371
           = 8733466864177587635
```

Level 4: Calculating d

```
p = 3361037497
q = 3507131801
What is tot(n)?
#1              <===

Whoop whoop!

e = 65537
What is d?
#2              <===


Nice job!

Congratulations! You beat Level 4!

The easiest way to break RSA is factoring n, if it is possible.
```

Answer and Explanation

```
#1
For the formula tot(n) = (p-1) * (q-1)
tot(n) = (3361037497-1)*(3507131801-1) = 11787601483213972800

#2
e = 65537
tot(n) = 11787601483213972800
mulinv(e, tot(n)) = 1841065181228591873
```

Code from Wikibooks

```
def xgcd(b, a):
    x0, x1, y0, y1 = 1, 0, 0, 1
    while a != 0:
```

```
        q, b, a = b // a, a, b % a
        x0, x1 = x1, x0 - q * x1
        y0, y1 = y1, y0 - q * y1
    return  b, x0, y0

# x = mulinv(b) mod n, (x * b) % n == 1
def mulinv(b, n):
    g, x, _ = xgcd(b, n)
    if g == 1:
        return x % n

print(mulinv(65537, 11787601483213972800))
```

Link

Extended_Euclidean_algorithm
Modular_multiplicative_inverse

Level 5: Factoring n

```
n = 9613631774438905337
What is p?
```

Answer

```
3536763563

Ayyyyy

Congratulations! You beat Level 5!

Now, let's put everything together and break RSA!
```

Explanation

```
[[[on Wolframalpha]]]
factor 9613631774438905337

Result:
2718200299 x 3536763563 (2 distinct prime factors)

-> 9613631774438905337 / 2718200299 = 3536763563

I think you could use the other number as the answer but I didn't test it.
```

Link

WolframAlpha

Level 6: Breaking simple RSA

```
c = 1940747889140053401
n = 11191414813257978223
e = 65537
What is p?
#1              <===

That was adequate.

What is q?
#2              <===

Way to go!

What is tot(n)?
#3              <===

Whoop whoop!

What is d?
#4              <===

Ayyyyy
```

```
Finally, what is m?
#5              <===

Nice job!

Congratulations! You beat Level 6!


Congratulations on finishing this introduction to RSA!
I hope this was fun and informative.

Here's your flag:
TUCTF{RSA_1$_R34LLY_C00L_4ND_1MP0RT4NT_CRYPT0}
```

Code and Tools

Answer and Explanation

```
#1
2749593919 (see below how we catch this result)

#2
From the formula n = p*q -> q = n/p
q = int(11191414813257978223/2749593919) = 4070206417

#3
From the formula tot(n) = (p-1) * (q-1)
tot(n) = (2749593919-1) * (4070206417-1) = 11191414806438177888

#4
e = 65537
tot(n) = 11191414806438177888
d = mulinv(e, tot(n)) = 9196881566601171041 (we reused the solution #5)

#5
[[[on Wolframalpha]]]
(194074788914005340 1^9196881566601171041) mod 11191414813257978223
Result: 8244229906027274615
```

RsaCtfTool (#1)

```
./RsaCtfTool.py -n 11191414813257978223 -e 65537 --verbose --private

RsaCtfTool output and redirect on a file:
echo '''-----BEGIN RSA PRIVATE KEY-----
MD4CAQACCQCbT+R6XklRbwIDAQABAgh/oeMOwpnQYQIFAPKaa9ECBQCj43k/AgUA
mB/asQIEa7KhSwIEQovFpw==
-----END RSA PRIVATE KEY-----''' > priv.key
```

Python code (#1)

```
from Crypto.PublicKey import RSA
public_key = RSA.importKey(open('priv.key', 'r').read())
print(public_key.p)
# Answer 2749593919
```

The Flag:

```
TF{RSA_1$_R34LLY_C00L_4ND_1MP0RT4NT_CRYPT0}
```

Refs Summary

RSA_link
Wikibooks
Extended_Euclidean_algorithm
Modular_multiplicative_inverse