RACTF 2020 Crypto - Really Secret Algorithm

deuterium — 2020-06-09

Canonical HTML: https://deut-erium.github.io/WriteUps/2020/ractf/crypto/Really˙Secret˙Algorithm/2020-06-09-RACTF-2020-Really-Secret-Algorithm.html

Abstract

CTF writeup: The walrus operator isn't a vault. RACTF 2020's Really Secret Algorithm hides RSA primes behind reversible running XOR and Base85.

Really Secret Algorithm

Challenge description reads as follows

```
We've received a weird message, but it's not in a format we've ever seen before.

  -*-*-*- BEGIN ARR ESS AYY MSG -*-*-*-
     00000000000000000000000000000000
     00000000000000000000000000000000
     00000000000000000000000000000000
     00000000000000000000000000000000
     00000000000000000000000000000000
     00000s&nYASMBl==Raa6f1mSybO1&`P
     n=MSlA^HVasQovKL?f9nB=?Wjz*-}bj
     4rNeU}9v(Tcn16Ji;Mjv?)4T@pD@76=
     9j%)LevT&=&p%BMcIckO@P450UqkjIR
     6DT^igJmh5<xI<alHa3p;VuZ%5HWp>1
                #T6e(?T*2I
                  00962
     jx@>fERjV6gRSH!+pdv<kOoEVD#<P05
     <nAMIT@fYQOcbQ{VfQh+sli_--_zE8)
     G@9Y^2j=XLkGz;kZTPS&eJtOKwM~!V6
     SmtDRCJ%568a_utlnc?ywyQ^??W!-Ro
     `%%d9c?q+nQ*s<4Sn4@*0vXe9sl<*c8
                  *WY0^
   -*-*-*- END ARR ESS AYY MSG -*-*-*-

We also recovered a snippet of the generator function, but we've not been able to get
anywhere with it.

def encrypt(message):
    p,q=rsa.prime_pair(bits=1024)
    ct=base64.b85encode(rsa.encrypt(rsa.solve_for(p=p,q=q,e=e),message.encode()))
    ct=b'\n'.join(ct[i:i+31].center(41)for i in range(0,len(ct),31))
    p,q=int.to_bytes(p,128,'big'),int.to_bytes(q,128,'big')
    s,key=0,bytearray()
    for(i,j)in zip(p,q):
        key.append(i^s)
        key.append((j^(s:=s^i),s:=s^j)[0])
    key=base64.b85encode(key)
    key=b'\n'.join(key[i:i+31].center(41)for i in range(0,len(key),31))
    e_str=base64.b85encode(int.to_bytes(e,4,'big')).center(41)
```

```
    return b'  -*-*-*- BEGIN ARR ESS AYY MSG -*-*-*-\n' + key + b'\n' + e_str + b'\n'
+ ct + b'\n' + b'   -*-*-*- END ARR ESS AYY MSG -*-*-*-\n'
```

Lets work out how the `encrypt` algorithm works line by line

- It generates two 1024 bit RSA primes `p` and `q`
- converts `p` and `q` to bytes (byte-arrays)
- initialises an integer s to zero, and a `bytearray`, lets further break down next few lines
    - `for (i,j) in zip(p,q)` reads `p` and `q` byte-by-byte
    - append `i^s` to key
    - This line is something new because of the new walrus operator `:=` in python3.8 which is a kind of assignment operator which assigns and returns the assigned value.
      so the following line would be equivalently:-

      ```
      s = s^i
      key.append(j^s)
      s = s^j
      ```
- base85 encode the key
- format the key into 3 sections of `key`, `e_str`, and `ct` with centering as a visual demarking.

Lets quickly split the provided key into parts

```
import base64
import gmpy2
key_part = """
     00000000000000000000000000000000
     00000000000000000000000000000000
     00000000000000000000000000000000
     00000000000000000000000000000000
     00000000000000000000000000000000
     00000s&nYASMBl==Raa6f1mSybO1&`P
     n=MSlA^HVasQovKL?f9nB=?Wjz*-}bj
     4rNeU}9v(Tcn16Ji;Mjv?)4T@pD@76=
     9j%)LevT&=&p%BMcIckO@P450UqkjIR
     6DT^igJmh5<xI<alHa3p;VuZ%5HWp>1
                #T6e(?T*2I
"""

e_str_part = """00962"""

ct_part = """
     jx@>fERjV6gRSH!+pdv<kOoEVD#<P05
     <nAMIT@fYQOcbQ{VfQh+sli_--_zE8)
     G@9Y^2j=XLkGz;kZTPS&eJtOKwM~!V6
     SmtDRCJ%568a_utlnc?ywyQ^??W!-Ro
     `%%d9c?q+nQ*s<4Sn4@*0vXe9sl<*c8
                  *WY0^
"""
```

```
key_b85 = "".join(i for i in key_part.strip().split()).encode()
key = base64.b85decode(key_b85)
e = int.from_bytes(base64.b85decode(e_str_part.encode()), "big")
ct_b85 = "".join(i for i in ct_part.strip().split()).encode()
ct = int.from_bytes(base64.b85decode(ct_b85), "big")
```

Now we will extract `p` and `q` from the `key` by reversing the walrus snippet in `encrypt`, which is done in following steps:-

- initialize `s` to zero
- initialize two byte arrays for `p` and `q`
- take out two bytes at a time from key namely `a` and `b`
- as `i` was coming from `p` and `j` from `q`, we extract `i` and `j`
- `i` is `a ^ s` of current iteration
- `j` is `a ^ b` of current iteration
- update `s` for next iteration

```
def get_pq(key):
    s = 0
    p = bytearray()
    q = bytearray()
    for i in range(0, len(key), 2):
        a, b = key[i:i + 2]
        p.append(a ^ s)
        q.append(a ^ b)
        s = b
    return int.from_bytes(p, 'big'), int.from_bytes(q, 'big')
```

Now once we have `p` and `q`, rest is cakewalk

```
p, q = get_pq(key)
phi = (p - 1) * (q - 1)
d = gmpy2.invert(e, phi)
m = pow(ct, d, p * q)
print(bytes.fromhex(hex(m)[2:]))
```

And we get our flag

ractf{DoY0uLik3MyW4lrus35"}

> Interactive controls are omitted from this PDF. See the canonical HTML version.