Features Business Explore Pricing

This repository Search

Sign in or Sign up

zionspike / ctf-writeup

Watch 0 Star 0 Fork 1

Code Issues 0 Pull requests 0 Projects 0 Pulse Graphs

Branch: master ctf-writeup / Crypto / [picoCTF 2017] - Weird RSA - 90 / kapi-note.md

Find file Copy path

zionspike add weir RSA 0b5fa52 14 days ago

1 contributor

122 lines (99 sloc) 3.68 KB

Raw Blame History

Weird RSA - 90

Problem:

```
ed some data. It was labeled as RSA, but what in the world are "dq" and "dp"? Can you decrypt the ciphertext for us?
```

This is the data file they gave us: RSA.txt

The challenge said about **dp** and **dq** in RSA so we googled them and we found dp and dq related to Chinese Remainder Theorem to RSA.

By implementing Chinese remainder algorithm

1. p and q are the primes
2. dp = d mod (p - 1)
3. dq = d mod (q - 1)
4. Qinv = 1/q mod p *This is not integer devision but multiplicative inverse
5. m1 = pow(c, dp, p)
6. m2 = pow(c, dq, q) 7-1) h = Qinv(m1 - m2) mod p ; if m1 < m2 7-2) h = Qinv * (m1 + q/p)
7. m = m2 + hq

We implement the algorithm and solve the problem.

```
import binascii
import struct

# return (g, x, y) a*x + b*y = gcd(x, y)
def egcd(a, b):
    if a == 0:
        return (b, 0, 1)
    else:
        g, x, y = egcd(b % a, a)
        return (g, y - (b // a) * x, x)

def decryptRSA(p,q,e,ct):
        # compute n
        n = p * q
        phi = (p - 1) * (q - 1)
        gcd, a, b = egcd(e, phi)
        d = a
        print "d: " + str(d)
        pt = pow(ct, d, n)
        return pt

def encryptRSA(p,q,e,pt):
        # compute n
        n = p * q
        phi = (p - 1) * (q - 1)
```

```
        gcd, a, b = egcd(e, phi)
        d = a
        print "d: " + str(d)
        ct = pow(pt, e, n)
        return ct


def convert(int_value):
   encoded = format(int_value, 'x')
   length = len(encoded)
   encoded = encoded.zfill(length+length%2)
   return encoded.decode('hex')

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

def main():
        # By implementing Chinese remainder algorithm
        # 1) p and q are the primes
        # 2) dp = d mod (p - 1)
        # 3) dq = d mod (q - 1)
        # 4) Qinv = 1/q mod p *This is not integer devision but multiplicative inverse
        # 5) m1 = pow(c, dp, p)
        # 6) m2 = pow(c, dq, q)
        # 7-1) h = Qinv(m1 - m2) mod p  ; if m1 < m2
        # 7-2) h = Qinv * (m1 + q/p)
        # 8) m = m2 + hq

        # m = 65
        # p = 61
        # q = 53
        # dp = 53
        # dq = 49
        # c = 2790

        p = 113874805849098549851253358482403842266539299427577563844893812422061571979865552439953351583287819703106
        q = 129722228752180865474258189614772579151055157059822837268518335080796004605424792679720502168386046497428
        dp = 819195772616111188086602822995016674222414765313689424808867824454881508674481065676552987628462282988444
        dq = 357069575758014809337024260850619146475642595470393023692458306581173054893227059556808837244180953591700
        c = 952727959864751895055189802511370035092926211401663838878548538637206924202041424484240748346571493268535

        Qinv = mulinv(q,p)
        print "Qinv: " + str(Qinv)

        m1 = pow(c, dp, p)
        print "m1: " + str(m1)

        m2 = pow(c, dq, q)
        print "m2: " + str(m2)

        h = (Qinv * (m1 - m2)) % p
        print "h: " + str(h)

        m = m2 + (h*q)
        print "m: " + str(int(m))

        hexadecimals = str(hex(m))[2:-1]
        print "solved: " + str(binascii.unhexlify(hexadecimals))
        # solved: Theres_more_than_one_way_to_RSA

if __name__ == "__main__":
        main()


# http://crypto.stackexchange.com/questions/19413/what-are-dp-and-dq-in-encryption-by-rsa-in-c
# https://en.wikipedia.org/wiki/RSA_(cryptosystem)#Using_the_Chinese_remainder_algorithm
# https://zzundel.blogspot.com/2011/02/rsa-implementation-using-python.html
```

The flag was:

- flag{Theres_more_than_one_way_to_RSA}