CMSC 443

Dylan Whichard

Dr. Zieglar

Hard RSA Cipher

May 12, 2015
Solved: May 12, 2015 4:45pm

1 Ciphertext

117634495262446775794344497070295862269406810003712811610619068472710752193033737263339593458183920009303572743187465466287817710212787415400140318547596385517374085065265482403835111363414795056815750699750809235773789773865255625080974763900355786533324843697806457720534917640720060358454934626959306496344527823913161179507888073861028667549621250036779610776275155980270299552934576806515996593795457494644955651587553158335967106247236524646089585864103228581458784400493784083758996812094836682259975641947685249422353894604754863548311188784113523423165536580623721856629110489902501346818080974032057446625196844294918326837245130393235378674794094609699793231044384966861396794839024948844572597861146730233182486830682683248981470177111414852150160662459622289202768908468153299783703057135864587252095165716970269986567891927698575161527190277896464443427157769762069473526568489019675613658783550423622797913102806138001707880385808102315826827188677031502119682483259391192797150902795797414137213139833006543069923243039251384902256248012241647160128658053326688292411137522762749761620068618106082910409687327177981720090837764124219180669967166918227186366853823897128283124540199699408398079411258455513645299557555162767294007053444758781448857049958853536252601245401996994083980794112584555136452995575551627672940070534447587814488570499588535362526017291137375252603520756091771634848319814978424775199292804608306856665309910123355209987133897305280581009343552437922021571207831185188701339965146629657506686133382701398857992835721715887858195052014590626369599395267172006586750915334246635231094529881092046697954218352196270617363611569962586016215303254566508781045612994140581663290933358550102569732157030541482349396675969417348915409456173508833935146778630095237994644664514978344923364562962303912126960576464075328264015374347753306036168628200745954883499446309084003155555861456514083156814163251533893128731829906165827225700258279095100592528528396560776215347184319112040501596043122997016818981631705777321157284734865800458845739707280859176843221324009084113069516560696305468793223649117269570718948196163286264197215344399151451439117312165955367507881205265908429401746814515043706740632449783834698024942147655482133262496214450124409986600014283571889666088180119736743940983304156482961509893842490996915612090703609532832835984644

2 Key

$a = 568420336116841869739492738904940040118849007625351752070854518944902754053086552660666619 5$

$b = 104357338478516236294480586766834030479967354464073276408025382550186579504739293300403361 55$

$p = 257703077330001045366553772498895839087739373 7$

$q = 890493366576925414100297530244263354415141275 7$

Where a is the private exponent, b is the public exponent, and p and q are the prime factors of n.

3 Plaintext

```
    If I Can Stop One Heart From Breaking - Emily Dickinson A wonderful life
creed.  What you do does matter to someone, somewhere.  If I can stop one heart
from breaking, I shall not live in vain; If I can ease one life the aching,
Or cool one pain, Or help one fainting robin Unto his nest again, I shall not
live in vain.
```

4 Methodology

My approach to breaking this cipher was to use a program called `GGNFS`[1][2] along with `msieve`[**?**] via a wrapper script[4] (with slight modifications required make it run), to factor the public key. It was then trivial to write my own script[5] which used Python's built-in support for big-number arithmetic and fast modular exponentiation to perform the decryption action with these derived values. In my solution, I also used an existing piece of code[6] to perform the extended Euclidean algorithm for modular inversion.

5 Time Spent

I spent approximately 8 hours working on the solution, and approximately 30 minutes on this writeup.

References

[1] http://sourceforge.net/projects/ggnfs/

[2] http://gilchrist.ca/jeff/factoring/nfs_beginners_guide.html

[3] http://sourceforge.net/projects/msieve/

[4] https://github.com/GDSSecurity/cloud-and-control/blob/master/scripts/gengnfsjob-testharness/factmsieve.74.py

[5] See below.

[6] http://rosettacode.org/mw/index.php?title=Modular_inverse&oldid=196584#Python

RSA.PY

```
#!/usr/bin/python
import sys

def parse_line(text):
    res = []

    if type(text) is int:
        text = str(text)
    for char in text:
        if not res:
            res.append(char)
            continue
        cur = res[-1]
        if int(cur+char) < 127:
            res[-1] += char
        else:
            if int(cur) < 32:
                print("Invalid: {}".format(cur))
            res.append(char)

    return ''.join((chr(int(n)) for n in res))

# From rosettacode.org/wiki/Modular_inverse#Python
def extended_gcd(aa, bb):
    last_rem, rem = abs(aa), abs(bb)
    x, last_x, y, last_y = 0, 1, 1, 0
    while rem:
        last_rem, (quotient, rem) = rem, divmod(last_rem, rem)
        x, last_x = last_x - quotient * x, x
        y, last_y = last_y - quotient * y, y
    return last_rem, last_x * (-1 if aa < 0 else 1), last_y * (-1 if bb <
         0 else 1)

def modinv(a, m):
    g, x, y = extended_gcd(a, m)
    if g != 1:
        raise ValueError
    return x % m

class RSA:
    def __init__(self, p, q, public_exponent, private_exponent=None):
        self.p = p
        self.q = q
        self.n = p*q
        self.e_pub = public_exponent

        if not private_exponent:
            private_exponent = modinv(public_exponent, (p-1)*(q-1))
```

```
        self.e_priv = private_exponent

    def encrypt(self, number):
        return pow(number, self.e_pub, self.n)

    def decrypt(self, number):
        return pow(number, self.e_priv, self.n)

def load(fn):
    # returns p, q, n, b, ciphertext
    p=q=n=b=None
    ciphertext = []

    with open(fn) as f:
        for line in f:
            if '=' in line:
                name, val = [s.strip() for s in line.split('=')]
                if name is 'p':
                    p = int(val)
                elif name is 'q':
                    q = int(val)
                elif name is 'n':
                    n = int(val)
                elif name is 'b':
                    b = int(val)
            elif line.strip():
                ciphertext.append(line.strip())

    if None in (p, q, b):
        raise ValueError
    if n is None:
        n = p * q

    return RSA(p, q, b), ciphertext

if __name__ == "__main__":
    if len(sys.argv) < 2:
        print("""␣Usage:␣{}␣<cipher-file>

␣␣␣␣␣␣␣␣cipher-file␣should␣consist␣of␣lines␣of␣the␣format␣<name>=<val>,
␣␣␣␣␣␣␣␣where␣name␣is␣'p',␣'q',␣'n',␣or␣'b',␣corresponding␣to␣the␣factors
    ␣of␣the
␣␣␣␣␣␣␣␣RSA␣modular␣value,␣the␣RSA␣modular␣value,␣and␣the␣public␣exponent
    .
␣␣␣␣␣␣␣␣Providing␣n␣is␣optional.␣Following␣these␣lines␣should␣be␣the␣
    ciphertext,
␣␣␣␣␣␣␣␣as␣numbers,␣one␣per␣line.""")
        exit(0)

    crypto, text = load(sys.argv[1])
```

```
print(''.join([parse_line(crypto.decrypt(int(line))) for line in text
    ]))
```