Boston Key Party: Mind Your Ps and Qs

10 Mar 2014 in Security
Tags: CTF, Boston Key Party, Security

About a week old, but I thought I'd put together a writeup for mind your Ps and Qs because I thought it was an interesting challenge.

You are provided 24 RSA public keys and 24 messages, and the messages are encrypted using RSA-OAEP using the private components to the keys. The flag is spread around the 24 messages.

So, we begin with an analysis of the problem. If they're using RSA-OAEP, then we're not going to attack the ciphertext directly. While RSA-OAEP might be vulnerable to timing attacks, we're not on a network service, and there are no known ciphertext-only attacks on RSA-OAEP. So how are the keys themselves? Looking at them, we have a ~1024 bit modulus:

```
1 >>> key = RSA.importKey(open('challenge/0.key').read())
2 >>> key.size()
3 1023
```

So, unless you happen to work for a TLA, you're not going to be breaking these keys by brute force or GNFS factorization. However, we all know that weak keys exist. How do these weak keys come to be? Well, in 2012, some researchers discovered that a number of badly generated keys could be factored. Heninger, et al discovered that many poorly generated keys share common factors, allowing them to be trivially factored! Find the greatest common divisor and you have one factor (p or q).

Then you can simply divide the public moduli by this common divisor and get the other, and you can trivially get the private modulus.

So far we don't know that this will work for our keys, so we need to verify this is the attack that will get us what we want, so we do a quick trial of this.

```
>>> import gmpy
>>> from Crypto.PublicKey import RSA
>>> key_1 = RSA.importKey(open('challenge/1.key').read())
>>> key_2 = RSA.importKey(open('challenge/2.key').read())
>>> gmpy.gcd(key_1.n, key_2.n)
mpz(127327280058646515192535368624440927590711679622088805147102534078
```

Great! Looks like we can factor at least this pair of keys. Let's scale up and automate getting the keys and then getting the plaintext. We'll try to go over all possible keypairs, in case they don't have one single common factor.

```
#!python
import gmpy
from Crypto.Cipher import PKCS1_OAEP
from Crypto.PublicKey import RSA

E=65537

def get_key_n(filename):
  pubkey = RSA.importKey(open(filename).read())
  assert pubkey.e == E
  return gmpy.mpz(pubkey.n)

def load_keys():
  keys = []
  for i in xrange(24):
    keys.append(get_key_n('challenge/%d.key' % i))
  return keys

def factor_keys(keys):
  factors = [None]*len(keys)
  for i, k1 in enumerate(keys):
    for k, k2 in enumerate(keys):
      if factors[i] and factors[k]:
        # Both factored
        continue
      common = gmpy.gcd(k1, k2)
      if common > 1:
        factors[i] = (common, k1/common)
        factors[k] = (common, k2/common)

  for f in factors:
    if not f:
      raise ValueError('At least 1 key was not factored!')

  return factors

def form_priv_keys(pubkeys, factors):
  privkeys = []
  for n, (p, q) in zip(pubkeys, factors):
    assert p*q == n
```

```
    phi = (p-1) * (q-1)
    d = gmpy.invert(E, phi)
    key = RSA.construct((long(n), long(E), long(d), long(p), long(q)))
    privkeys.append(key)

  return privkeys

def decrypt_file(filename, key):
  cipher = PKCS1_OAEP.new(key)
  return cipher.decrypt(open(filename).read())

def decrypt_files(keys):
  text = []
  for i, k in enumerate(keys):
    text.append(decrypt_file('challenge/%d.enc' % i, k))
  return ''.join(text)

if __name__ == '__main__':
  pubkeys = load_keys()
  factors = factor_keys(pubkeys)
  privkeys = form_priv_keys(pubkeys, factors)
  print decrypt_files(privkeys)
```

Let's run it and see if we can succeed in getting the flag.

```
$ python factorkeys.py
FLAG{ITS_NADIA_BUSINESS}
```

Win! Nadia, of course, is a reference to Nadia Heninger, 1st author on the Factorable Key paper.

Related Posts

Belden Garrettcom 6K/10K Switches: Auth Bypasses, Memory Corruption 19 May 2017

Applied Physical Attacks and Hardware Pentesting 13 May 2017

Security Issues in Alerton Webtalk (Auth Bypass, RCE) 27 Apr 2017