RSA CRT Decryption Fault ENG

February 13, 2020

1 RSA CRT Decryption Fault Attack

Since RSA is an energy- and computation-intensive algorithm some devices use a Chinese Remainder Theorem Variant of RSA for decryption. First the values $dP = e^{-1} mod(p-1)$, $dQ = e^{-1} mod(q-1)$ and $qInv = q^{-1} modp$ are precomputed. Then, decryption works as follows:

$$M_p = C^{dP} modp$$

$$M_q = C^{dQ} modq$$

$$h = qInv \cdot (M_p - M_q) modp$$

$$M = M_q + h \cdot q$$

$M_p = Mmodp$ and $M_q = Mmodp$

The algorithm works great, significantly decreasing needed computation, however it has one flaw. If a fault is inject during the computation of one of M's remainders, then we are left with M', for which:

$$M'_p = M_p$$

$$M'_q \neq M_q$$

Since $M'_p = M_p$, $M - M' = kp$, for $k \in \mathbb{Z}$, so $GCD(M - M', N) = p$, allowing us to factor N and compute d.

Get the M and M' from the server, computed d and send it back to get the flag. Good luck!

[5]:
```
import socket
import re
from Crypto.Util.number import bytes_to_long,long_to_bytes,inverse,GCD
class VulnServerClient:
    def __init__(self,show=True):
        """Initialization, connecting to server"""
        self.s=socket.socket(socket.AF_INET,socket.SOCK_STREAM)
        self.s.connect(('cryptotraining.zone',1340))
        if show:
            print (self.recv_until().decode())
    def recv_until(self,symb=b'\n>'):
        """Receive messages from server, by default till new prompt"""
        data=b''
        while True:
            data+=self.s.recv(1)
            if data[-len(symb):]==symb:
                break
        return data
    def get_public_key(self,show=True):
        """Receive public key from the server"""
        self.s.sendall('public\n'.encode())
        response=self.recv_until().decode()
        if show:
            print (response)
        e=int(re.search('(?<=e: )\d+',response).group(0))
        N=int(re.search('(?<=N: )\d+',response).group(0))
        self.num_len=len(long_to_bytes(N))
        return (e,N)

    def decryptBytes(self,m,show=True):
        """Get a decryption for chosen byte message from the server"""
        try:
            num_len=self.num_len
        except KeyError:
            print ('You need to get the public key from the server first')
            return
        if len(m)>num_len:
            print ("The message is too long")
            return
        if len(m)<num_len:
            m=bytes((num_len-len(m))*[0x0])+m
        hex_m=m.hex().encode()
        self.s.sendall(b'decrypt '+hex_m+b'\n')
        response=self.recv_until().decode()
        if show:
            print (response)
        if response.find('flag')!=-1:
            print('You tried to submit \'flag\'')
            return None
        signature_hex=re.search('(?<=Signature: )[0-9a-f]+',response).group(0)
        signature_bytes=bytes.fromhex(signature_hex)
        return bytes_to_long(signature_bytes)


    def decryptNumber(self,m,show=True):
        """Get a decryption for chosen number from the server"""
        try:
            num_len=self.num_len
        except KeyError:
            print ('You need to get the public key from the server first')
            return
        return self.decryptBytes(long_to_bytes(m,num_len),show)

    def faultyDecryptBytes(self,m,show=True):
        """Get a faulty decryption for chosen byte message from the server"""
        try:
            num_len=self.num_len
        except KeyError:
            print ('You need to get the public key from the server first')
            return
        if len(m)>num_len:
            print ("The message is too long")
            return
        if len(m)<num_len:
            m=bytes((num_len-len(m))*[0x0])+m
        hex_m=m.hex().encode()
        self.s.sendall(b'faulty_decrypt '+hex_m+b'\n')
        response=self.recv_until().decode()
        if show:
            print (response)
        if response.find('flag')!=-1:
            print('You tried to submit \'flag\'')
            return None
        signature_hex=re.search('(?<=Signature: )[0-9a-f]+',response).group(0)
        signature_bytes=bytes.fromhex(signature_hex)
        return bytes_to_long(signature_bytes)


    def faultyDecryptNumber(self,m,show=True):
        """Get a faulty decryption for chosen number from the server"""
        try:
            num_len=self.num_len
        except KeyError:
            print ('You need to get the public key from the server first')
            return
        return self.faultyDecryptBytes(long_to_bytes(m,num_len),show)

    def checkDNumber(self,c,show=True):
        """Check if this number is d"""
        try:
            num_len=self.num_len
        except KeyError:
            print ('You need to get the public key from the server first')
            return
        signature_bytes=long_to_bytes(c,num_len)
        self.checkDBytes(signature_bytes,show)
```

```python
    def checkDBytes(self,c,show=True):
        """Check if this byte sequence is d"""
        try:
            num_len=self.num_len
        except KeyError:
            print ('You need to get the public key from the server first')
            return
        if len(c)>num_len:
            print ("The message is too long")
            return

        hex_c=c.hex().encode()
        self.s.sendall(b'flag '+hex_c+b'\n',)
        response=self.recv_until(b'\n').decode()

        if show:
            print (response)

        if response.find('Wrong')!=-1:
            print('Wrong signature')
            x=self.recv_until()
            if show:
                print (x)
            return
        flag=re.search('CRYPTOTRAINING\{.*\}',response).group(0)
        print ('FLAG: ',flag)

    def __del__(self):
        self.s.close()
```

[6]:
```python
vs=VulnServerClient()
(e,N)=vs.get_public_key()
```

```
Welcome to RSA CRT Decryption Faults task
Available commands:
help - print this help
public - show public key
decrypt <hex(data)> - decrypt ciphertext
faulty_decrypt <hex(data)> - decrypt with fault
flag <hex(d))> - print flag
quit - quit
>
e: 65537
N: 2015971766318676420084248263832914243247937675568128643256140001120775156877O
23937873504239055098886463647821209788938254180637863281345152201173477839435246
47506954302364591564396569321085369361070927857591871209155591733213020275252290
18106368725032056109022369913503577023942696069608771010384365856481001383579432
```

```
84411223121576763032862701509742254008778946240450869708632121399086803127321961
48979014368449994422593874530212706423955318848486976509334781242540719122324457
08062597679170291021925633789812405697682134528381868778865376836541179591638312
152472136313757252384761293684336082840137773984575947459061
>
```

```
[ ]:
```