# Bleichenbacher Signature Exponent 3 Attack ENG

February 13, 2020

## 1 Bleichenbacher's signature attack on RSA with exponent 3

Public exponent 3 is often used due to the fact that computation for it is simpler. Obviously, it can be a problem if the message $M$ you are trying to encrypt doesn't wrap the modulus. To mitigate this there were created several padding schemes. Most noteworthy from the point of attacks is PKCS#1 v1.5, which we will learn further down the way. It uses a somewhat similar format both for signatures and encryption, but today we'll be looking into signatures. A valid signature under PKCS encoding has the following format:

`00 | 01 | FF | .. | FF | 00 | HASH`

If $N$ is $k$ bytes long (usually 128,256 or 384), then the number of FF bytes is supposed to be $(k - 3 - len(HASH))$.

Unfortunately, certain implementations checked just for sequence

`00 | 01 | FF | .. | FF | 00`

with arbitrary number of bytes FF and then took the hash after those. The problem is that what initially was just one possible message has now way too many possibilities, since all the bytes after hash can be whatever you want. With $e = 3$ this becomes a real issue, since we can forge the signature. What you can do is try to take the approximate cubic root of $M =$

`00 | 01 | FF | 00 | HASH | 00 | .. | 00`

$S' \approx M^{1/3}$ then find signature $S = S' + 1$. Since this will change just the bytes after the hash, the signature will pass the check.

Forge a signature for the b'flag' and send it to the server, only the server is not using the hash of 'flag', instead it just looks for 'flag' right after padding. Good luck!

[1]:
```python
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',1341))
        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 checkSignature(self,c,show=True):
        """Check if this number is a valid signature for flag"""
        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.checkSignatureBytes(signature_bytes,show)

    def checkSignatureBytes(self,c,show=True):
        """Check if this byte sequence is a valid signature for flag"""
        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)
```

```python
        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()
```

[2]:
```python
vs=VulnServerClient()
(e,N)=vs.get_public_key()
```

```
Welcome to Bleichenbacher's signature exponent 3 attack task
Available commands:
help - print this help
public - show public key
flag <hex(signature(b'flag'))> - print flag
quit - quit
>
e: 3
N: 2345172163845073583719293651270528508414801698643772620413957930569232314355O
81964345787672240794572490003697517555394747462447654353297912996510869465982681
36790820845336349648480663904722604204569306361101850465776320279299837628291806
99297454189397229979451189326493503464132097157965882767371766584791340102547650
80712744161589731316860006929661017332925231984135770311579228265777992033597729
80378140505908286573965733673226813533790453648080805499611914254159619378643824
95328668937920235775400914630813078535281943770942554463414529423069019686396596
201141371706207268929145987762222661753151969189517471009277
>
```

[ ]: