1 Preliminaries

1.1 SAGE

Download and install the Sage library [1].

1.2 Basic Coppersmith Attack

The following code generates an RSA key with a modulus $N$ of $n$ bits, generates a random polynomial:

$$f(x) = x^2 + ax + b \mod N$$

with a small root $|x_0| < 2^{n/3}$, and recovers this root using Coppersmith’s technique.

```python
def keyGen(n=256):
    "Generates an RSA key"
    while True:
        p=random_prime(2^(n//2)); q=random_prime(2^(n//2)); e=3
        if gcd(e,(p-1)*(q-1))==1: break
        d=inverse_mod(e,(p-1)*(q-1))
        Nn=p*q
        print "p",p,"q",q
        print "N",Nn
        print "Size of N:",Nn.nbits()
    return Nn,p,q,e,d

def CopPolyDeg2(a,b,Nn):
    "Finds a small root of polynomial x^2+ax+b=0 mod N"
    n=Nn.nbits()
    X=2^(n//3-5)
    M=matrix(ZZ,[[X^2,a*X,b],
                [0,Nn*X,0],
                [0,0,Nn]])
    V=M.LLL()
    v=V[0]
    return [v[i]/X^(2-i) for i in range(3)]

def test():
    "Generates a random polynomial with a small root $x_0$ modulo $N$ and recovers his root."
    Nn,p,q,e,d=keyGen()
    n=Nn.nbits()
    x0=ZZ.random_element(2^((n/3)-10))
    a=ZZ.random_element(Nn)
    b=mod(-x0^2-a*x0,Nn)
    print "x0=",x0
```
2 Application to breaking RSA

2.1 Polynomials of degree 3

Modify the previous code to find small roots of polynomials of degree 3. What is the new bound on $x_0$?

2.2 Application to breaking RSA encryption

Let

$$N = 2122840968903324034467344329510307845524745715398875789936591447337206598081$$

be an RSA modulus of size 251-bits. Let $m$ be a message with $m < 2^{36}$. Let

$$c = (2^{250} + m)^3 \mod N$$

We have:

$$c = 392293632962225871356015460642971309021740757869377374897362323056543628$$

Recover the message $m$ using Coppersmith’s technique.

2.3 Extension

Extend the previous attack to handle larger messages $m$, by using lattices of higher dimension.

References