Cryptography in the real World - Homework

Jean-Sébastien Coron

Université du Luxembourg

1 Homework

Please answer the questions below. You must provide:

1. A PDF document providing the answer to the questions below. The PDF document can include some relevant parts of your source code.
2. The source code.

2 Attack on variants of RSA

2.1 Secret modulus

Assume that Alice wants to keep her RSA modulus N secret to everybody except to Bob. Alice uses $e = 3$ as public exponent. To encrypt a message m, Bob computes $c = m^3 \mod N$ and sends c to Alice. Assume that Eve gets $c_1 = m_1^3 \mod N$ and $c_2 = m_2^3 \mod N$ and already knows m_1 and m_2; explain how Eve can recover N.

2.2 Common modulus

Assume that Alice and Bob want to share the same modulus N but use different public exponent. Alice uses $e_A = 3$ and Bob uses $e_B = 5$. Let d_A and d_B be the corresponding private exponents. Explain how Alice can recover d_B from d_A.

2.3 Common modulus, cont.

Assume that Alice and Bob want to share the same modulus N but use different public exponent. Alice uses $e_A = 3$ and Bob uses $e_B = 5$. Now Charlie wants to encrypt a message m for Alice and Bob. He sends:

$$c_A = m^3 \mod N$$

to Alice and

$$c_B = m^5 \mod N$$

to Bob. Explain how Eve can recover m from N, c_A and c_B.

2.4 Implementation

Download and install the NTL number theory library available at `www.shoup.net`. Check that the previous attacks work by implementing them with NTL.

3 Coppersmith Attack

3.1 SAGE

Download and install the Sage library [1].

3.2 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.

```
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 x0 modulo Nn
```

```
    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
  v=CopPolyDeg2(a,b,Nn)
  R.<x> = ZZ[]
  f = v[0]*x^2+v[1]*x+v[2]
  print find_root(f, 0,2^(n//3-10))
```

3.3 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 ?

3.4 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:

$c = 3922936329622225871353601546064297130902174075788693773748973623230565 43628$

Recover the message m using Coppersmith's technique.

3.5 Extension

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

4 Fault attacks

1. Implement the plain RSA signature scheme using the NTL library available at `www.shoup.net`, with a modulus size of 1024 bits, and using the Chinese Remainder Theorem (CRT) : to compute $s = m^d \mod N$, compute

$$s_p = s \mod p = H(m)^{d \mod p-1} \mod p$$

   and

$$s_q = s \mod q = H(m)^{d \mod q-1} \mod q$$

   Recover $s \mod N$ from s_p and s_q using the CRT.

2. Assume that an error occurs during the computation of s_p, that is, an incorrect value $s'_p \neq s_p$ is computed while s_q is correctly computed. Show how to recover the factorization of N from s. How could such error be detected ? Propose and implement a simple method to detect such error.

5 DGHV Somowhat Homormorphic Encryption Scheme

Implement the basic DGHV encryption scheme [4], without the squashed decryption and without the bootstrapping, but using the compression of the public-key as described in [3]. You can use the SAGE library [1].

5.1 Optional: DGHV with Squashed Decryption

Implement DGHV with squashed decryption, as described in [4, 2].

5.2 Optional: full DGHV

Implement the fully homomorphic DGHV encryption scheme, including the bootstrapping procedure, as described in [4, 3].

References

1. Sage Mathematical Library, Available at `http://www.sagemath.org/`
2. Jean-Sebastien Coron, Avradip Mandal, David Naccache, Mehdi Tibouchi: Fully Homomorphic Encryption over the Integers with Shorter Public Keys. CRYPTO 2011: 487-504.
3. Jean-Sebastien Coron, David Naccache, Mehdi Tibouchi: Public Key Compression and Modulus Switching for Fully Homomorphic Encryption over the Integers. EUROCRYPT 2012: 446-464
4. Marten van Dijk, Craig Gentry, Shai Halevi, Vinod Vaikuntanathan: Fully Homomorphic Encryption over the Integers. EUROCRYPT 2010: 24-43.