University College London
Department of Computer Science

# Cryptanalysis Lab 7

P. Spacek

April 4, 2018

peter.spacek@stuba.sk
Version 1.0

## Implementing the Coppersmith's Algorithm

Write an implementation of the Coppersmith's method for solving discrete logarithm. This algorithm was the first algorithm in computational number theory to have heuristic subexponential complexity of the form $L_q(1/3, c + o(1))$. The method uses a polynomial basis for $F_{2^n}$ of the form $F_2[x]/(F(x))$ for $F(x) = x^n + F_1(x)$ where $F_1(x)$ has very small degree. Let $b \in N$ be such that $b = cn^{1/3}log(n)^{2/3}$ for a suitable constant c.

This lab was based on "Mathematics of Public Key Cryptography" by Steven Galbraith, available from https://www.math.auckland.ac.nz/~sgal018/crypto-book/crypto-book.ht

Click on the green letter before each question to get a full solution. Click on the green square to go back to the questions.

Exercise 1.

(a) Let $g = x^{11} + x^7 + x^5 + x^2 + 1$ and $h = x^{14} + x^{11} + x^{10} + x^9 + 1$

be the DLP instance. Our task is to find such m that $h = g^m$ Let $F_{2^{15}} = F_2[x]/(F(x))$, where $F(x) = x^{15} + x + 1$. We consider the subgroup of $F^*_{2^{15}}$ of order r = 151 (note that $(2^{15} - 1)/r = 731 = 217$).

(b) First compute in sage $n^{1/3}$ and $n^{2/3}$ Wrt n we choose b=3.

(c) Let $B = \{A(x) \in F_2[x] : deg(A(x)) \leq b, A(x) - irreducible\}$. Note that $\#B \approx 2^b/b$ . Create factor base $B$. You can list all the elements in the base.

(d) Now, we do Coppersmith's method.
Let $k \in N$ be such that $2^k \approx \sqrt{n/b} \approx \frac{1}{\sqrt{c}}(n/log(n))^{1/3}$,
let $l = \lceil n/2^k \rceil \approx \sqrt{nb} \approx \sqrt{c}n^{2/3}log(n)^{1/3}$.
Compute k, l.

(e) Suppose $A(x), B(x) \in F_2[x]$ are such that $deg(A(x)) = d_A \approx b$ and $deg(B(x)) = d_B \approx b$ and define $C(x) = A(x)x^l + B(x)$.
In practice one restricts to pairs $(A(x), B(x))$ such that $gcd(A(x), B(x)$
1.
Write function to generate such A(x) and B(x), construct C(x)

(f) The crucial observation is that

$C(x)^{2^k} = A(x^{2^k})(x^{2^k})^l + B(x^{2^k}) \cong A(x^{2^k})x^{2^k l ? n} F_1(x) + B(x^{2^k})$ $(modF(x))$.

Let D(x) be the right hand hand side of equation.

$D(x) = A(x^{2^k})x^{2^k l ? n} F_1(x) + B(x^{2^k})(modF(x))$.

We have $deg(C(x)) \leq max\{d_A + l, d_B\} \approx l \approx n^{2/3} log(n)^{1/3}$ and $deg(D(x)) \leq max\{2^k d_A + (2^k l - n) + deg(F_1(x)), 2^k d_B\} \approx 2^k b \approx n^{2/3} log(n)^{1/3}$.

Compute D(x).

(g) We have two polynomials C(x),D(x) of $degree \approx n^{2/3}$ that we wish to be b-smooth where $b \approx n^{1/3} log(n)^{2/3}$. Write function to test smoothness over factor base. The function should return $false$ if polynomial is not smooth, or its decomposition in form of list of exponents (how many times is the factor base polynomial in the testing polynomial.) Test polynomials for smoothness over B.

(h) We will also assume that the resulting relations are essentially random (and so with high probability there is a non-trivial linear dependence once $\#B + 1$ relations have been collected). Create relation matrix. Relations have a form of $C(x)^{2^k} = D(x)$

(i) Having generated enough relations among elements of the factor base, it is necessary to find some relations involving the elements g and h of the DLP instance. The easiest way is to find such exponents i, j of $g$ and $h$ that $g^i$ and $h^j$ are smooth. This is fast enough for our subgroup, but may not be easy for larger groups. Write function for finding such i and j. Add decomposition of $g^i$ and $h^j$ to the relation matrix.

(j) Find a non-trivial kernel vector modulo r. Find non zero elements u and v in two last columns of the kernel matrix. This gives us a relation:
$1 = (g^i)^u(h^j)^v$ With use of math compute m such that $h = g^m$

(k) Heureka! Lets party all night, we have a solution ;)

---

*rest is optional*

---

## Exercise 2.

(a) For generating enough relations among elements of the factor base in real life, it is necessary to find some relations involving the elements g and h of the DLP instance. This is not trivial. You cannot do it like in the previous example in the real life. All DLP algorithms having complexity $L_q(1/3, c + o(1))$ feature a process called special q-descent that achieves this. The first step is to express g (respectively, h) as a product $\prod_i Q_i Gi(x)$ of polynomials of degree at most $b_1 = n^{2/3} log(n)^{1/3}$; this can be done by multiplying g (resp. h) by random combinations of elements of B and factoring. We now have a list of around $2n^{1/3} < n$ polynomials Gi(x) of $degree \approx n^{2/3}$ that need to be 'smoothed' further. Essentially one performs the same sieving as earlier except that A(x) and B(x) are chosen so that $G_i(x)|C(x) = A(x)x^l + B(x)$ (not necessarily with the same value of l or the same degrees for A(x) and B(x)). Defining $D(x) = C(x)^{2^k}(modF(x))$ (not necessarily the same value of k as before) one hopes that C(x)/G(x) and D(x) are b-smooth. After sufficiently many trials one has a relation that expresses $G_i(x)$ in terms of elements of B. Repeating for the polynomially many values $G_i(x)$ one eventually has the values g and h expressed in terms of elements of B. One can then do linear algebra modulo the order of g to find integers $Z_1, Z_2$ such that $g^{Z_1}h^{Z_2} = 1$ and the DLP is solved.

**Exercise 3.**

(a) In exercise 1 we list all the elements for the base. Write function to generate factor base $B$

## Solutions to Exercises

**Exercise 1(a)**

n=15
$F2t15.<a>=GF(2**n, name='a', modulus=x**n+x+1)$
r=151
$g=a**11+a**7+a**5+a**2+1$
$h=a**14+a**11+a**10+a**9+1$

□

**Exercise 1(b)**

```
n13=n**(0.3333333)
print n13
n23=n**(0.6666666)
print n23
b=3
```

□

**Exercise 1(c)**

B=[a,a+1,a**2+a+1,a**3+a+1,a**3+a**2+1]

□

**Exercise 1(d)**

```
print sqrt(n/b+0.0)
k=1
l=ceil(n/2**k)
```

□

**Exercise 1(g)**

Naive solution. Try to find better one:

```
def is_smooth(px,B):
    gx=px
    exponent_list = []
    for fx in B:
        exponent = 0
        while((gx/fx)<(gx)):
            gx=gx/fx
            exponent = exponent+1
        exponent_list = exponent_list + [exponent]
    if gx>1:
        return False
    return exponent_list
```

□

**Exercise 1(h)**

One solution would be like this. Try to find better one:

```
relMat=[]
maxDeg = 2
for p in (0..1):
    for q in (0..maxDeg):
        for u in (0..1):
            for v in (0..maxDeg):
                Ax=u + a**v
                Bx=p + a**q
                Cx=Ax*a**l+Bx
                Dx=Cx**(2**k)
                if(is_smooth(Cx,B)!=false):
                    if(is_smooth(Dx,B)!=false):
                        relMat=relMat+[list((2**k)*
vector(is_smooth(Cx,B))-vector(is_smooth(Dx,B)))]
```

□

**Exercise 1(i)**

```
for i in (1..r):
    if (is_smooth(g**i,B)!=false):
        print i, ": ",is_smooth(g**i,B)
        gn=i
        relMat=relMat+[is_smooth(g**i,B)]
        break
for i in (1..r):
    if (is_smooth(h**i,B)!=false):
        print i, ": ",is_smooth(h**i,B)
        hn=i
        relMat=relMat+[is_smooth(h**i,B)]
        break
```

□

**Exercise 1(j)**

```
relMat = matrix(relMat)
M=relMat.kernel().matrix()
print M
for i in M:
    if i[-2]!=0:
        if i[-1] != 0:
            ge= i[-2]*gn% r
            he =-i[-1]*hn% r
            result = inverse_mod(he,r)*ge%r
            print result
print g**result==h
```

□

**Exercise 1(k)**

```
buy(drinks, snacks)
for f in flats:
    if (f.empty()):
        occupate( CryptanalysysCourse.Students() )
party(morning)
```

□

**Exercise 3(a)**

```
def get_base(b,n):
    B=[]
    R=ZZ['a']
    Z=GF(2**(n),'a')
    for c in GF(2**(b+1), 'a'):
        if R(c).is_irreducible():
            B=B+[Z(c)]
    return B
```

□