Generating Trapdoor Primes

A short take on generating SNFS primes

Nikolai Rozanov

UCL - Computer Science

Table of contents - Introduction

Importance

Importance

- Look Up
- Benchmarking
- Deliberate Weakening

Method and Code

Algorithm

Step 1. Generating the prime q of the corresponding size.

Step 2. Generating the coefficients for polynomials f and g, according to the size suggestions in [1]

Step 3. Setting up a new polynomial G, which is the resultant of f and g and the variable is the leading coefficient of g.

Step 4. Finding roots of G-1 modulo q, if there are no roots then going back to Step 1.

Step 5. Then letting p=$|G|$, and checking if p is prime, (an addition is to check whether q divides p-1).

Main Script

Main Script

```
# Generating required Rings
X.<x>      = ZZ['x']
G.<x,g1>  = ZZ['x,g1']
# Main Loop
    #generating prime and Associated Field
    q = get_prime(bits_q)
    T.<g2>     = Integers(q)['g2']
        ##while loop
        f_poly,norm_f = get_f(X,degree_f,bits_q)
        g_poly,g0 = get_g(g1,x,bits_p,degree_f, norm_f)
        G_poly = get_G(f_poly,g_poly)
        temp = list(G_poly.coefficients())
        temp.reverse()
        T2 = T(temp)
    r     = T2.roots()
    root = r[0][0]
    rt    = int(root)
#    while(rt<int(2^(bits_p/degree_f)/norm_f)):
#        rt+=q
    p     = X([G_poly(1,rt)+1])
```

Helper Functions

Prime Generation

```
def get_prime(bits_q):
    q = random_prime(2^bits_q -1,False,2^(bits_q -1))
    while(not is_prime(q)):
        q = random_prime(2^bits_q -1,False,2^(bits_q -1))
    return q
```

Poly f

```
def get_f(X,degree_f, bits_q):
    flag_irreducible = False
    while (not flag_irreducible):
        #f_vec = [ZZ.random_element(-int(2^(10)-1),int(2^(10)-1)) for _ in range(degree_f+1)]
        f_vec = [ZZ.random_element(-int(2^(bits_q/(2*(degree_f+1)))),int(2^(bits_q/(2*(degree_f
            +1))))) for _ in range(degree_f+1)]
        #f_vec = [ZZ.random_element(1,int(2^(bits_q/(2*(degree_f+1))))) for _ in range(degree_f
            +1)]

        norm_f = max(map(abs,f_vec))
        f_poly = X(list(f_vec))
        if f_poly.is_irreducible():
            flag_irreducible = True
    return f_poly,norm_f
```

Helper Functions

Poly g

```
def get_g(g1,x,bits_p,degree_f,norm_f):
    g0 = ZZ.random_element(-int(2^(bits_p/degree_f)/norm_f),int(2^(bits_p/degree_f)/norm_f))
    #g0 = ZZ.random_element(1,int(2^(bits_p/degree_f)/norm_f))
    g_poly = g1*x+g0
    return g_poly,g0
```

Poly G

```
def get_G(f_poly,g_poly):
    G_temp = f_poly.sylvester_matrix(g_poly,variable=x)
    G_poly = G_temp.determinant()-1
    return G_poly
```

Some Analysis

References I

J. Fried, P. Gaudry, N. Heninger, and E. Thomé.
A kilobit hidden snfs discrete logarithm computation.
arXiv preprint arXiv:1610.02874, 2016.