Math 571-01, Cryptography Project 02
Quadratic Sieve
University of Massachusetts Amherst

Matthew Gramigna
Barry Greengus
Wei Xie

April 5, 2017

1 Introduction

The quadratic sieve is an efficient method to find many numbers greater $\sqrt{N}$ for some given N whose squares mod N are "B-smooth" for a given positive integer B. i.e. Let $N, B \in \mathbb{Z}$. Find $a^2 \pmod{N}$ s.t. $\forall p_i$ in $a^2 = p_1^{e_1} * p_2^{e_2} * ... * p_k^{e_k}$, $p_i \leq B$

1.1 Difference by Squares Factoring

Consider the problem of factoring, specifically the method of factoring using difference of squares. If a number N is known to be the difference of two squares, say $N = X^2 - Y^2$, then $N = (X + Y)(X - Y)$. So all we have to do to factor N is to find a number b such that $N + b^2$ is a perfect square. Then $N + b^2 = a^2$, so

$$N = a^2 - b^2 = (a + b)(a - b)$$

and we have just factored N.

A random value of b is unlikely to produce a perfect square, but it is fairly likely for a multiple k of N to equal the difference of two squares

$$kN = a^2 - b^2 = (a + b)(a - b)$$

such that $(a+b)$ or $(a-b)$, besides for being a factor of kN, is also a non-trivial factor of N. This means we only need to find a difference of two squares that equals a mutiple of N, which is the equivalent of finding a and b such that $a^2 \equiv b^2 \pmod{N}$. This fact enables a three step factoring algorithm comprised of:

Step1:
Step2:
Step3:

Quadratic sieve solves the first step of this algorithm.

2 Overview of Quadratic Sieve

Explain how quadratic sieve works TODO more info +equations + fix "=" to congruent
Setup Step: Given number N and set of primes P, where all elemenet in P ¿= B, set a=floor of the sqrt(N), set a quadratic polynomial. we will use F(T) = T2̂ - 221.

Step 1: build a list of F(a) to F(L(a)). TODO define L(), explain why we use it. explain why we start at a.

Step 2: For i=2 to B, where i=some p in P or is prime factor of some p in P:

Step 3: Predict where division of elem in list by i CAN happen.
if p — F(T), then T$\hat{2}$ = N mod p has a solution, else no solution so you cant divide by p.
So, if p odd and T$\hat{2}$ = N mod p has two solutions, a and b. all mulitples of those solutions can also be divided by the p

Step 4: Divide all multiples of the solutions a and b in the list by p

Step 5: Whenever the quotient of a list element is 1, it's prime factors are clearly only primes ¡= B and is thus B-smooth

3 Implementation

We used gp-pari for our implementation, mainly due to its power and useful built-in mathematical functions. The source code is in section 5 of this document.

3.1 Initial Approach

Initially, we wanted to get our code to work for a small example such as the one in the book, and then expand it to see how it worked for the larger examples. We also were not considering efficiency at this time, and wanted the algorithm to work before we tried any optimization. Naively, this approach involved iterating through every number in the list and checking divisibility by all primes in our list of primes then dividing each number as much as possible.

This approach works, but not efficiently. In fact, we need not consider primes p where the congruence $a^2 \equiv N \pmod{p}$ has no solutions. Also, once we know that some number a is divisible by p, we can also divide multiples of p from that number a in the list and skip over ones we know will not be divisible by p. This leads to our more efficient and final implementation.

3.2 Final Implementation

For our final implementation, we used the aforementioned techniques to improve the efficiency of the sieve by only considering numbers and primes that we know have a chance of being divisible by said prime.

As seen in the code, we use a guard for testing if a number is a quadratic residue modulo some prime in the list p, and if not then we don't consider that prime in the sieve. Moreover, we use the square roots modulo p to determine where to start the divisions for the sieve in the `forstep` loop. Once we have the sieved list, we find the matrix kernel to form the squares as mentioned in section 2, and return the factorization of N.

3.3 Interesting Details/Pitfalls

We struggled with determining appropriate values for b, the size of the list of numbers, and B, the bound on the size of primes to use in the sieve. We found that these values can be quite different depending on the number being factored.

3.4 Testing

In order to ensure that the algorithm works as desired, we added a simple check to make sure the factorization is correct:

```
{
    validate(factors,N)=
        if(factors[1]*factors[2]==N,  return(True));
        return(False);
}
```

This verifies that the factorization found by the algorithm is indeed the correct factorization. The string “True” or “False” is present in the return value of the `QSfactor` function.

4 Efficiency

5 Source Code

6 Group Organization/Administrative

6.1 Git

TODO, we used git bc useful for XYZ

6.2 Meetings

met how often? helpful bc why?