RSA Cryptography: Factorization

Kevin Chu

3/11/10

Contents

Abstract

The purpose of this paper is to explore the topic of factorization in RSA cryptography in greater detail as well as to bring up other related problems and applications. This paper is organized according to several questions I came up with regarding the topic.

1 Background

1.1 What is RSA?

RSA is a public key cryptography algorithm first introduced in 1978. It is an interesting mathematical problem because the algorithm relies on principles in number theory, making it an application of "pure" math. It is also interesting because despite its simplicity, no one has managed to prove that RSA or the underlying integer factorization problem cannot be cracked. RSA cryptography has become the standard crypto-system in many areas due to the great demand for encryption and certification on the internet. The basis for RSA cryptography is the apparent difficulty in factoring large semi-primes. Although there are many algorithms that can factor very large numbers of a certain form, a general purpose algorithm is still unknown.

1.2 How it works

The general scheme of RSA is this:

1. Pick two large prime numbers p and q which are somewhat close to each other.
2. Take $n = p * q$ the product.
3. Take $\phi(pq) = (p-1)(q-1)$.
4. Choose an integer e such that $1 < e < \phi(pq)$, and $\gcd(e, \phi(pq)) = 1$.
5. Compute d such that $de \equiv 1 \pmod{\phi(pq)}$

We then release the public key (n, e) and keep the private key (n, d). A message m, an integer $0 < m < n$, can then be encrypted by computing $s \equiv m^e \pmod{n}$. To decrypt an encrypted message s, we can compute $m \equiv s^d \pmod{n}$ which gives us our original message m.

From this scheme we can easily see that if we can factor n we will get $n = pq$. Using this we can then compute $de \equiv 1 \pmod{\phi(pq)}$ which gives us the private key (n, d).

2 Why is factorization hard?

It seems strange that something as simple as factorization should be difficult to solve. The reason factorization remains a challenging problem is the size of numbers that are used in crypto-systems such as RSA. Currently the largest number that has been factored is 768 bits (232 decimal digits). RSA keys are generally at least 1024 bits long (309 decimal digits). What makes RSA an ideal algorithm for crypto-systems is the inherent asymmetry between generating primes (polynomial time) and factoring large semiprimes. As long as there is no general polynomial time algorithm for factoring large numbers, RSA may remain secure.

The factor() function in Sage can be used to show how difficult it is to factor large numbers on a personal computer. Using the following code we can see how long it takes to factor random numbers of various lengths.

```
for a in range(1,20):
    n = ZZ.random_element(a * 5)
    timeit(‘factor(n)’)
```

The following output shows the runtime for factoring numbers from 50 digits to 100 digits on a laptop with a 2 GHz processor:

```
sage: n= ZZ.random_element(10^50)
sage: timeit('factor(n)')
25 loops, best of 3: 21.5 ms per loop

sage: n= ZZ.random_element(10^55)
sage: timeit('factor(n)')
5 loops, best of 3: 2.68 s per loop

sage: n= ZZ.random_element(10^60)
sage: timeit('factor(n)')
5 loops, best of 3: 2.13 s per loop

sage: n= ZZ.random_element(10^65)
sage: timeit('factor(n)')
5 loops, best of 3: 20.3 s per loop

sage: n= ZZ.random_element(10^70)
sage: timeit('factor(n)')
5 loops, best of 3: 3.93 s per loop

sage: n= ZZ.random_element(10^75)
sage: timeit('factor(n)')
```

```
5 loops, best of 3: 13.6 s per loop

sage: n= ZZ.random_element(10^80)
sage: timeit('factor(n)')
5 loops, best of 3: 12.1 s per loop

sage: n= ZZ.random_element(10^85)
sage: timeit('factor(n)')
5 loops, best of 3: 86.8 ms per loop

sage: n= ZZ.random_element(10^90)
sage: timeit('factor(n)')
5 loops, best of 3: 22.7 s per loop

sage: n= ZZ.random_element(10^95)
sage: timeit('factor(n)')
5 loops, best of 3: 1.87 s per loop

sage: n= ZZ.random_element(10^100)
sage: timeit('factor(n)')
5 loops, best of 3: 160 ms per loop
```

From this we can see that not all numbers of a given length take the same amount of time to factor. For example a 100 digit number took just 160 milliseconds to factor while a 65 digit number took 20.3 seconds. This suggests that certain numbers are harder to factor. In practice, large semiprimes are the most difficult to factor. While this data does not give an accurate heuristic of the runtime of a factoring algorithm, it does give some insight as to the difficulty of factoring very large numbers. If a 65 digit number takes 20 seconds to factor, we can see that in practice, very large numbers (ie. 2048 bit or 617 digits) cannot be factored in a reasonable amount of time.

3 What are some common factoring techniques?

3.1 Pollard's $p-1$ algorithm

One well known algorithm used to factor large numbers quickly is Pollard's $(p-1)$ algorithm. Pollard's method relies on the fact that a Number N with prime divisor p can be factored quickly if $p-1$ is "smooth" (ie, has small prime factors). In modern cryptography, Pollard's algorithm is not necessarily useful. Well designed crypto-systems

will choose numbers that do not work well.

3.2 Elliptic curve factorization

Elliptic curve factorization is an improvement on Pollard $(p-1)$ algorithm. It replaces the restriction of only using the group $(\mathbb{Z}/p\mathbb{Z})^*$ which always has order $p-1$ and thus depends on having a factor p such that $p-1$ is smooth. Instead elliptic curve factorization uses the group of points on a random elliptic curve.

4 Alternatives to RSA?

It is feasible that eventually someone will come up with an efficient algorithm for factoring large numbers as this has not yet been proven impossible. In that case RSA will be broken and there must be a replacement. There are currently relatively few alternatives which are as versatile as RSA in part due to the fact that RSA has withstood years of testing in the real world. On the other hand it is also possible that factorization is in the class *np-hard*. In this case, a polynomial time algorithm for factorization is very unlikely as it would imply a polynomial time algorithm to the whole class of problems.