Fall 2006
Chris Christensen
MAT/CSC 483

Mathematical attack on RSA

If we know $\phi(n)$ and the public key (the modulus *n* and the encryption exponent *e*), then we can determine *d* because *d* is the inverse of *e* mod *n*. We can use the Extended Euclidean algorithm (in *Mathematica*, **ExtendedGCD[**integer, integer**]**) to determine *e*. Then we can read the message.

Now knowing $\phi(n)$ is mathematically equivalent to knowing *p* and *q* – the two prime factors of *n*. Why? Well, certainly is we know *p* and *q* we know $\phi(n)$ because $\phi(n) = (p-1)(q-1)$. Conversely, if we know *n* and $\phi(n)$, then

$$n - \phi(n) + 1 = pq - (p-1)(q-1) + 1 = pq - pq + p + q - 1 + 1 = p + q.$$

So, we know $n = pq$ and $n - \phi(n) + 1 = p + q$. This suggests a quadratic equation which has *p* and *q* as its roots.

$$X^2 - (n - \phi(n) + 1)X + n = (X - p)(X - q)$$

For example, if we know that $n = 27153383$ and $\phi(n) = 27142080$, then solving (by using the quadratic formula, TI-92, or *Mathematica*)

$$X^2 - 11304X + 27153383 = (X - 7841)(X - 3463)$$

$n = 7841*3463$.

So, knowing $\phi(n)$ and the public key (which allows us to break any message encrypted with that key) is equivalent to factoring *n*. There is no known efficient algorithm to factor large integers.

So, how might we factor *n*?

We know that *n* is composite (in fact we know that it is the product of 2 large primes). One of the factors of *n* must be less than or equal to $\sqrt{n}$. The brute force attack is to try division by all positive integers less than or equal to $\sqrt{n}$. This is not recommended.

There is a factoring algorithm due to Fermat (1601 – 1665) that helps if the 2 primes are nearly the same size. Here is how it works.

Fermat factoring algorithm

The algorithm is based upon the being able to factor the difference of 2 squares.

$$x^2 - y^2 = (x+y)(x-y)$$

If $n = x^2 - y^2$, then n factors: $n = (x+y)(x-y)$. But, every positive odd integer can be written as the difference of two squares. In particular for the integers that we use of RSA moduli $n = pq$,

$$n = pq = \left(\frac{p+q}{2}\right)^2 - \left(\frac{p-q}{2}\right)^2$$

Let k be the smallest positive integer so that $k^2 > n$, and consider $k^2 - n$. If this is a square, we can factor n: if $k^2 - n = h^2$, then $n = (k+h)(k-h)$. If $k^2 - n$ is not a square, increase the term on the left by one and consider $(k+1)^2 - n$. If this is a square, n factors. If $(k+1)^2 - n$ is not a square, consider $(k+2)^2 - n$. Etc. Eventually, we will find an h so that $(k+h)^2 - n$ factors. That is so because $\left(\frac{n+1}{2}\right)^2 - n = \left(\frac{n-1}{2}\right)^2$.

In this case, n factors as $n = n \times 1$. $k \le k+h \le \frac{n+1}{2}$.

Here is an example. n = 6699557. $\sqrt{n} \approx 2588.35$; so, k = 2589.
$k^2 - n^2 = 2589^2 - 6699557 = 58^2$. So,

$$6688557 = 2589^2 - 58^2 = (2589+58)(2589-58) = 2647 \times 2531$$

Fermat's factorization algorithm works well if the factors are roughly the same size.

Here is another example. $n = 26504551$. $\sqrt{26504551} \approx 5148.26$; so, $k = 5149$.

$5149^2 - 26504551$ is not a square.
$5150^2 - 26504551$ is not a square.
$5151^2 - 26504551$ is not a square.

.
.
.

$5840^2 - 26504551 = 2757^2$

So, $26504551 = 5840^2 - 2757^2 = (5840 + 2757)(5840 - 2757) = 8597 \times 3083$.

The Pollard *p* – 1 factorization algorithm

Let's just consider the case of interest – factoring $n = pq$ where p and q are large primes. This algorithm works well if either $p - 1$ or $q - 1$ is a product of relatively small primes. Let's assume that $p - 1$ is the product of small primes.

First, we guess an r so that $p - 1$ divides r. Of course, in practice we will not know p, but for the moment assume that we do know p. It might be convenient to take r to be a factorial that is large enough that $p - 1$ divides r.

For example, $7001 - 1 = 2^3 \times 5^3 \times 7$; so, 15! Would work because 2^3, 5^3, and 7 each divide 15!. $4536 - 1 = 2^3 \times 3^4 \times 7$; so, 9! would work. But, $5869 - 1 = 2^2 \times 3^2 \times 163$; so, we would need to use at least 163!. This is what we mean by saying that we want to take $r = k!$ sufficiently large so that $p - 1$ divides r. But, remember that because we won't know p we are guessing that we have chosen r large enough. If r is not large enough, the algorithm will fail to find a factor. Assuming that $p - 1$ divides r, we can write $r = (p - 1)\, j$.

Also choose a positive integer a so that $1 < a < p$-1. Once again, we're guessing that our a satisfies this inequality.

Say, we have r sufficiently large so that $p - 1$ divides r and a so that $1 < a < p - 1$. Then notice that

$$a^r = a^{(p-1)j} = \left(a^{p-1}\right)^j = 1^j = 1 \bmod p$$

What this tells us is that p divides $a^r - 1$, and because $n = pq$, p divides $\gcd\left(a^r - 1, \quad n\right)$. In fact, $\gcd\left(a^r - 1, \quad n\right) = p$. (Unless, by chance, $a^r - 1 = 0 \bmod n$. In that case we choose another a.)

So, that's the algorithm. After choosing r large enough so that $p - 1$ divides r and a so that $1 < a < p$, we calculate $\gcd\left(a^r - 1 \bmod n, \quad n\right)$. (We have good algorithms for gcd and for modular exponentiation.) If we have chosen r and a correctly, $\gcd\left(a^r - 1 \bmod n, \quad n\right) = p$.

Example: Let n = 70348807, a = 2, and r = 13!.

PowerMod[2, 13!, 70348807**]**

17662502

GCD[% - 1, 70348807**]**

7723

Which is one of the factors of n = 70348807. $70348807 = 7723 \times 9109$.

If we took r too small here's what happens. Say, r = 10!. (After the fact we can see that $7723 - 1 = 2 \times 3^3 \times 11 \times 13$; so 10! is too small.)

PowerMod[2, 10!, 70348807**]**

60592434

GCD[% - 1, 70348807**]**

1

Factoring

RSA's public key consists of the modulus *n* (which we know is the product of two large primes) and the encryption exponent *e*. The private key is the decryption exponent *d*. Recall that *e* and *d* are inverses mod $\phi(n)$. Knowing $\phi(n)$ and *n* is equivalent to knowing the factors of *n*.

One attack on RSA is to try to factor the modulus *n*. If we could factor *n*, we could calculate $\phi(n)$ and (by using the extended Euclidean algorithm) determine *d*.

Here are some factoring techniques:

Trial division: Try all the primes that are $\leq \sqrt{n}$. It's not very elegant, but "in theory" it would work. The problem is that, like other brute force techniques, it's not practical.

Fermat (1601 – 1665) factorization: Not a bad technique if *p* and *q* are relatively equidistant from $\sqrt{n}$.

(1974) Pollard *p*-1 algorithm: Not bad if *p* – 1 or *q*– 1 is the product of small primes.

(1975) Pollard ρ -algorithm: The book discusses this algorithm.

(1981) Pomerance quadratic sieve algorithm QSA: Still fast for up to around 110 decimal digits.

(c 1993) Number field sieve NFS: The most efficient; based on work of Pollard (1988).

(c 1987) Elliptic curve method ECM: H. Lenstra.

(1994) Schor's algorithm: Needs a quantum computer. The existing 7 cubit quantum computer has factored 15.

The RSA factoring challenge numbers
http://www.rsasecurity.com/rsalabs/node.asp?id=2092