Attacks on RSA, some using LLL

Recall RSA: $N = pq$ hard to factor. Choose e with $\gcd(e, \phi(N)) = 1$, where $\phi(N) = (p-1)(q-1)$. Via extended Euclid, find d with $ed \equiv 1 \pmod{\phi(N)}$. Discard p and q. Public key is N, e. Private key is d. Encipher m as $c = m^e \bmod N$. Decipher c as $m = c^d \bmod N$.

1. RSA problem: Given N, e, c, find m.

2. Compute d: Given N, e, find d.

3. Factor N: Given N, find p and q.

Clearly, $3 \to 2 \to 1$.

In fact, $3 \equiv 2$. It is not known whether $3 \equiv 1$. All three problems seem hard, although Shor showed that one can solve 3 quickly on a quantum computer.

Relax the problems. Assume we know some hint about p or q or d, either because they are limited or we have an oracle for them..

Coppersmith proved that if $f(x)$ is a monic polynomial and p is an unknown factor of a given integer N, then one can find all "small" solutions x to $f(x) \equiv 0 \pmod{p}$ quickly via LLL.

Let $N = pq$, where $p > q$. Suppose we know N and a bit more than half of the high-order bits of p. (Oracle or limit on p.)

Specifically, suppose we are give $\tilde{p}$ with $|p - \tilde{p}| < N^{1/4}$.

Let $f(x) = x + \tilde{p}$. Then $x_0 = p - \tilde{p}$ is a small zero of $f(x) \equiv 0 \pmod{p}$ so we can find it with Coppersmith's method.

Define polynomials for $i = 0$ to h-1 by

$$f_i(x) = N^{h-i} f^i(x) = N^{h-i}(x + \tilde{p})^i$$

$$f_{h+i}(x) = x^i f^h(x) = x^i (x + \tilde{p})^h.$$

Then p^h divides all $f_i(x_0)$, $i = 0, 1, \ldots, 2h - 1$.

Let $X = N^{1/4-\epsilon}$.

The coefficient vector of a polynomial $h(x) = a_0 + a_1 x + a_2 x^2 + \cdots a_n x^n$ is the vector $v = (a_0, a_1, \ldots, a_n)$.

Define a lattice L of dimension $2h$ by the basis of coefficient vectors of $f_i(xX)$.

Apply LLL to get a reduced basis, including a shortest vector $v =$ the coefficient vector of a polynomial $g(xX)$.

We have $|g(x_0)| < p^h$ and p^h divides $g(x_0)$. Therefore, $g(x_0) = 0$. Solve $g(x) = 0$ in integer x. Then $p = x_0 + \tilde{p}$.

Another result of Coppersmith is that one can find all of a message m, provided one knows 2/3 of its bits and $e = 3$. This might happen when m is a standard message, like, “The password for today is wxyz.”

We are given N, $c = m^3 \bmod N$ and $\tilde{m}$ with $|m - \tilde{m}| < N^{1/3}$. We must find m.

Write $m = \tilde{m} + x_0$, where $|x_0| < N^{1/3}$. Then we must find a small root ($x = x_0$) of

$$f(x) = ((\tilde{m} + x)^3 - m^3) \bmod N.$$

This is done with the same technique just discussed.

In a similar way, one can find d, given N and e provided d is small, say, $0 < d < N^{1/4}$.

Write $ed = 1 + k\phi(N)$. Since $\phi(N) = N - (p + q) + 1$, we can write $\phi(N) = N - z$ with $z <$ a small multiple of $\sqrt{N}$. This leads to

$$ed - kN - kz - 1 = 0.$$

Since kz is much smaller than ed or kN, we have $e/N \approx k/d$. Continued fractions allow us to recover k and d from e and N when $d < N^{1/4}$.

Rewrite $ed - kN - kz - 1 = 0$ as $kN + kz + 1 \equiv 0 \pmod{e}$. A variation of Coppersmith's method using lattices and LLL lets one find solutions to this congruence with z near $\sqrt{N}$ and k near N^{α} for α up to about 0.29.

Now suppose that the *same* message is encrypted with RSA and sent to k receivers, each with different RSA moduli, but with the same (small) enciphering exponent e.

An eavesdropper would know the k public moduli N_1, ..., N_k and the k ciphertexts $c_i = m^e \bmod N_i$ for $1 \leq i \leq k$.

If two N_i were not relatively prime, then the eavesdropper could factor both of them (since they could have only one of their two prime factors in common), compute both d and decipher c to get m.

And if every pair of N_i were relatively prime, then the eavesdropper could use the CRT to compute m^e modulo the product of all k moduli. If the value of m^e is less than this product, then it can be computed. Finally, m can be computed by taking the e-th root of the integer m^e.

A special case is $e = 3$.

We now show that computing d in RSA is equivalent to factoring N. Clearly, one can compute d quickly given the factors of N.

It is easy to see that we can factor N given N and $\phi(N)$ by solving a quadratic equation. We show that one can find $\phi(N)$ quickly from N, e and d.

Since $\phi(N) = N - (p + q) + 1$ and $p \approx q$, the polynomial

$$f(x) = N - x \bmod \phi(N)$$

has a root $x_0 = p + q - 1$ of size $2N^{1/2}$. We don't know $\phi(N)$, but we do know that it divides $M = ed - 1$ and that $M < N^2$.

The LLL algorithm quickly computes all roots x_0 with $x_0 < 2N^{1/2}$.

Here is a simpler way to see that one can factor N given e and d.

Recall that $ed \equiv 1 \pmod{\phi(N)}$. Therefore, $ed - 1 = k\phi(N)$ for some integer k. If we let $r = ed - 1$, then whenever $\gcd(a, N) = 1$ we have

$$a^r = a^{ed-1} = \left(a^{\phi(N)}\right)^k \equiv 1 \pmod{N}$$

by Euler's theorem. Note that r is even because $\phi(n)$ is even for $n > 2$.

Now write $r = 2^s d$ with d odd. Choose a random a in $1 < a < N - 1$. If $\gcd(a, N) > 1$, then N has been factored and we are done. Otherwise, compute $b_i = a^{2^i d} \bmod N$ for $0 \le i \le s$. We know that $b_s = a^r \bmod N = 1$ by the congruence above.

If for some $0 < i \leq s$ we have $b_i = 1$ but $b_{i-1} \not\equiv \pm 1 \pmod{N}$, then $\gcd(b_{i-1} - 1, N)$ is a proper factor of N. If there is no such i, try a different random a. The reason this works is that $b_{i-1}^2 \equiv 1 \pmod{N}$, but $b_{i-1} \not\equiv \pm 1 \pmod{N}$, so $\gcd(b_{i-1} - 1, N)$ is a proper factor of N by an earlier theorem. In fact, each random a leads to a factorization of N with probability at least 1/2.

Timing Attack on RSA

This insidious attack was discovered by Kocher and apply to nearly all cryptographic algorithms whose execution time depends on the input value.

In order to perform the attack, you must be able to observe a cipher program running on your computer and make precise measurements of the time it takes to run on various inputs. You must also know the input value and the parameters of the cryptographic algorithm other than the secret key. Someone with an account on the victim's machine and who could observe incoming packets could easily obtain the required information.

Let us use RSA as a simple example of a timing attack. The victim has modulus n, enciphering exponent e and deciphering exponent d. The latter is secret, while n and e are public. The victim receives ciphertext messages C and deciphers them by computing $M = C^d \bmod n$.

Let $d = \sum_{i=0}^{k} b_i 2^i$ be the binary representation of d. The attacker records many ciphertexts C_j and the time t_j needed to decipher each. He deduces d one bit at a time, from b_0 to b_k. This is the order in which the bits are used in the fast exponentiation algorithm. Assume that the first r bits have been computed.

[Fast Exponentiation for RSA Deciphering]
Input: A modulus n, an exponent $d \geq 0$ and a ciphertext C.
Output: The value $M = C^d$.

```
$t = d$
$M = 1$
$z = C$
while ($t > 0$) {
        if ($t$ is odd) $M = Mz \bmod n$
        $z = z^2 \bmod n$
        $t = \lfloor t/2 \rfloor$
        }
return $M$
```

Let us suppose that the operation $M = Mz$ mod n takes longer for some pairs M, z than for other pairs and that the attacker can measure the execution time of the algorithm accurately enough to notice the difference. Because the first r bits of d have been computed, the attacker can perform the first r iterations of the `while` loop for input C_j and measure its time c_j precisely.

The attacker can also measure the precise time d_j the operation $M = Mz$ mod n would take, if it were done. He knows whether this particular modular multiplication is fast or slow compared to the time for average pairs M, z.

Using a formula from statistics, he can predict whether b_r is 0 or 1. He compares the two variances $v_1 = \mathbf{Var}(t_j - c_j)$ and $v_2 = \mathbf{Var}(t_j - c_j - d_j)$. If $v_1 > v_2$, the bit b_r is probably 1; but if $v_1 < v_2$, then b_r is probably 0.

For if the multiplication occurs, it is reasonable to assume that the time d_j it takes and the time $t_j - c_j - d_j$ for the part of the fast exponentiation after it are mutually independent, so

$$v_1 = \mathbf{Var}(t_j - c_j) = \mathbf{Var}(t_j - c_j - d_j) + \mathbf{Var}(d_j)$$

$$v_1 > \mathbf{Var}(t_j - c_j - d_j) = v_2.$$

But if the multiplication does not occur, then the time d_j it takes and the time $t_j - c_j$ for the part of the fast exponentiation after it are mutually independent, so

$$v_2 = \mathbf{Var}(t_j - c_j - d_j) = \mathbf{Var}(t_j - c_j) + \mathbf{Var}(-d_j)$$

$$v_2 > \mathbf{Var}(t_j - c_j) = v_1.$$

If a mistake is made, then no further significant differences between v_1 and v_2 will appear for larger r. In that case, the attacker will notice the error, back up and correct it.