CS181A Notes #3 Basic RSA

Here is the famous RSA algorithm in several simple steps. Suppose that the input is a positive integer k (also called the security parameter).

1. Choose two distinct random k-bit prime numbers p and q.
2. Compute $N = pq$ and $\phi = (p-1)(q-1)$.
3. Choose two integers e and d so that $ed \equiv 1 \pmod{\phi}$.

Now, to encrypt a message $x \in \mathbb{Z}_N$ we use $\mathrm{Enc}(x) \equiv x^e \pmod{N}$ whereas to decrypt a cipher $y \in \mathbb{Z}_N$ we use $\mathrm{Dec}(y) \equiv y^d \pmod{N}$. We will discuss the first and third steps as the second step requires (only) integer multiplication.

Generating random primes To generate (or choose) random k-bit prime numbers, we use the simple two-step idea of generating a random k-bit number and testing if it is prime. If not, we repeat the process (and show that there is an abundance of prime so the expected waiting time is short); otherwise, we have found our random prime number. The only interesting part here is how to test if a number is prime or not. This is the so-called **primality testing** problem.

The simplest and fastest algorithm for primality testing is Miller-Rabin's algorithm. It is based on two simple observations:

- Fermat's Little Theorem (contrapositive version):
  If there is a number $a \not\equiv 0 \pmod{m}$ so that $a^{m-1} \not\equiv 1 \pmod{m}$, then m is not prime.
- Primes Have Only Trivial Roots of Unity:
  If there is a number $a \not\equiv \pm 1 \pmod{m}$ so that $a^2 \equiv 1 \pmod{m}$, then m is not prime.

These two observations can be combined a compact way into a simple algorithm.

Miller-Rabin's algorithm: Let the input be an odd integer m

1. Choose a random integer a from $\{1, 2, \ldots, m-1\}$.
2. If $a^m \not\equiv a \pmod{m}$ then output **no** else output **yes**.

First, note that if the output is **no** then this is always correct. We will show below that the probability of error in the **yes** answer is *only* $1/2$. This yields a bounded one-sided error probabilistic algorithm for primality testing (actually, for detecting non-primality).

Next, note that to compute $a^m \bmod m$, we may use the following recursive formulation of (modular) exponentiation.

$$a^b \bmod m = \begin{cases} 1 & \text{if } b = 0 \\ (a^{b/2})^2 \bmod m & \text{if } b > 0 \text{ is even} \\ a^{b-1} a \bmod m & \text{if } b > 0 \text{ is odd} \end{cases} \tag{1}$$

So, when we compute $y = a^{b/2} \bmod m$ in the even case and must return $z = y^2 \bmod m$, we can insert the following check: if $z = 1 \bmod m$ but $y \neq \pm 1 \bmod m$ then return 0 (failure code) else return z (normal return). Returning 0 in the case of y being a non-trivial square root of one will cause the test $a^m \not\equiv a \pmod m$ in Step 3 to succeed (and hence the output NO is corect).

Theorem 1. *The error probability of the Miller-Rabin primality testing algorithm is at most* $1/2$.

Proof. Assume that the input to the algorithm is an odd integer m. Thus, $m - 1$ is even and can be written as $m - 1 = 2^u v$, where v is odd and $u \geq 1$.

Suppose there is an $a \in \mathbb{Z}_m^\star$ so that $a^{m-1} \not\equiv 1 \pmod m$. Then, the subgroup $B = \{a \in \mathbb{Z}_m^\star : a^{m-1} \equiv 1 \pmod m\}$ is proper and hence $|B| \leq \frac{1}{2}|\mathbb{Z}_m^\star|$ by Lagrange's theorem. This implies that for more than half of the a's, we will observe $a^{m-1} \not\equiv 1 \pmod m$.

On the other hand, now suppose that $a^{m-1} \equiv 1 \pmod m$ for all $a \in \mathbb{Z}_m^\star$. Let j be the largest integer so that there is an a with $a^{2^j v} \equiv -1 \pmod m$. There is always such an integer j since $a \equiv -1 \pmod m$ with $j = 0$ satisfies the condition. Fix an integer a so that $a^{2^j v} \equiv -1 \pmod m$ for the largest $j < u$. Assume first that $m = m_1 m_2$ for two relatively prime integers $m_1, m_2 > 1$. Then, we can define an integer b so that

$$b \equiv \begin{cases} a & \pmod{m_1} \\ 1 & \pmod{m_2} \end{cases}$$

Note that $b^{2^j v} \not\equiv \pm 1 \pmod m$ since $b^{2^j v} \equiv -1 \pmod{m_1}$ and $b^{2^j v} \equiv 1 \pmod{m_2}$. This shows that the subgroup $B = \{a \in \mathbb{Z}_m^\star : a^{2^j v} \equiv \pm 1 \pmod m\}$ is proper and thus $|B| \leq \frac{1}{2}|\mathbb{Z}_m^\star|$. This implies that for more than half of the a's chosen, we

will see a transition from a value $a^{2^k v} \not\equiv \pm 1 \pmod{m}$ to the value $a^{2^{k+1} v} \equiv 1 \pmod{m}$, where k lies between j and $u-1$.

For the remaining case, suppose $m = p^k$, where p is an odd prime and $k \geq 2$. Then, $\phi(m) = p^{k-1}(p-1)$ and $a^{\phi(m)} \equiv 1 \pmod{m}$ by Euler-Fermat's little theorem. Since $a^{m-1} \equiv 1 \pmod{m}$, we have $p^{k-1}(p-1)|(m-1)$ or $p|(m-1)$. Thus, $m \equiv 1 \pmod{p}$ and $m \equiv 0 \pmod{p}$, which is impossible. □

Generating random keys The task of finding e and d so that $ed \equiv 1 \pmod{\phi}$ can be given to the Puliverizer of Aryabhata. Choose a random e and run the Extended Euclidean algorithm on ϕ and e. If $\gcd(e, \phi) = 1$, then the output contains two integers x and y so that

$$1 = ex + \phi y.$$

Now, we simply set $d \equiv x \pmod{\phi}$.

Speedup issues Some implementations of RSA impose the condition $e = 3$ (or $e = 17$) to speed up the encryption process. This is possible by requiring that the primes $p \neq q$ satisfy $\gcd(3, (p-1)(q-1)) = 1$. To speed up the decryption process, we may use the Chinese Remainder Theorem. For this, we first compute $d_p \equiv d \pmod{p-1}$ and $d_q \equiv d \pmod{q-1}$. Upon receiving the cipher y, and having to compute $x \equiv y^d \pmod{n}$, we instead compute $x_p \equiv y^{d_p} \pmod{p}$ and $x_q \equiv y^{d_q} \pmod{q}$. Finally, we use the Chinese Remainder Theorem to obtain x which satisfies the simultaneous congruence $x \equiv x_p \pmod{p}$ and $x \equiv x_q \pmod{q}$. The advantage of this scheme is that the two separate computations modulo p and q can be done in parallel (and with precomputed exponent values). Moreover, the constants of Chinese Remainder Theorem, α_p and α_q, can also be precomputed. In total, we need two (smaller) modular exponentiations plus two (larger) modular multiplications and one modular additions.

Weaknesses (1) The basic RSA is a *deterministic* scheme. This allows Eve to recognize if the same message had been sent again. A simple remedy for this is to use *salting* – which is the process of adding a known number of random low-order bits to the message prior to encryption. Decryption will then simply strip or ignore these random bits.

(2) RSA is *multiplicative* in the sense that $E(xy) = E(x)E(y)$ for any two messages x and y. This property can be exploited in a one-time chosen ciphertext

attack (CCA). Suppose Eve needs to decrypt the message $y = E(x)$ and she can somehow convince Bob to decrypt any message of her choice provided it is not y. So, Eve chooses a message z, so that z^{-1} exists, and then asks Bob to decrypt $E(x)E(z) = E(xz)$. After receiving xz from Bob, Eve can simply recover x by cancelling z out.

(3) RSA with low encryption exponents is susceptible to attacks. Suppose Alice sends the same message x to three Bobs, all of whom used $e = 3$ in their RSA setups, but with different moduli, namely n_1, n_2 and n_3. So, Eve observes the three pairwise relatively prime moduli (otherwise two of them share a prime factor and Eve can compute x easily) along with the following congruences:

$$x^3 \equiv \begin{cases} y_1 & (\mathrm{mod}\ n_1) \\ y_2 & (\mathrm{mod}\ n_2) \\ y_3 & (\mathrm{mod}\ n_3) \end{cases}$$

Using the Chinese Remainder Theorem, Eve computes $z \equiv x^3 \pmod{n_1 n_2 n_3}$. But, since $x < n_1$, $x < n_2$ and $x < n_3$ we have $x^3 < n_1 n_2 n_3$. Therefore, the integer cube root of z is actually x itself.