COMP 8920: Cryptography Winter 2025

Lecture 19 — Mar 18, 2025

Prof. Curtis Bright *Scribe: Aaron Barnoff*

1 RSA Implementation

Recall: Implementing RSA on numbers that will be secure in practice involves multiprecision arithmetic.

Say x has k bit and y has l bits, and $k \geq l$. Then:

- $x \neq y$ uses $O(k)$ bit ops.
- xy uses $O(kl)$ bit ops.
- x/y uses $O(l(k-l))$ bit ops.
- $\gcd(x, y)$ uses $O(kl)$ bit ops (using Euclid's algorithm).

Here, "bit op" (bit operation) means one basic CPU instruction, like adding or multiplying a number that fits in the CPU's word size. Technically a bit op would be a single logic gate. Each word op can be reduced to a constant number of bit ops.

RSA will use modular multiprecision arithmetic. If $m_1, m_2 \in \mathbb{Z}_n^*$, where n is a k-bit integer, then:

- $m_1 \pm m_2 \bmod n$ uses $O(k)$ bit ops.
- $m_1 \cdot m_2 \bmod n$ uses $O(k^2)$ bit ops.
- $m_1^{-1} \bmod n$ uses $O(k^2)$ bit ops (using the Extended Euclidean Algorithm).
- $m_1^c \bmod n$ involves $c-1$ multiplications and $c-1$ divisions (due to the mod$n$ reduction), yielding $O(c \cdot k^2)$ bit operations.
  - For RSA, c will be the exponent of the private or public key.
  - The public key exponent can be as small as $c = 3$, which is acceptable.
  - Decryption, however, will involve a very large c (e.g., $c = 1024$ bits) to prevent a brute force attack.

If n is a 1024-bit number (so $k = 1024$) and the decryption exponent in RSA is a number (with $\phi(n) < n$), then:

- Two random numbers, e and d, are chosen such that they are inverses modulo $\phi(n)$.
- This means that $c < n$, so c is a k-bit number and $c \approx 2^k$, where $k \approx \log(n)$.
- This gives a cost of $2^k \cdot k^2$, which is infeasible for $k \approx 1024$.
- Note that having c with k bits is different from c being of size k; in fact, it is of size $O(2^k)$.

1.1 Computing m_1^c

We can compute m_1^c using the square-and-multiply (fast exponentiation) algorithm in $O\big(\log(c) \cdot k^2\big)$ bit operations:

- That is, $O\big(\log(c) \cdot (\log n)^2\big)$.

- We want arithmetic to be proportional to the number of bits in a number (its logarithm), not the magnitude of the number itself.
- Since $k = \log(n)$, the arithmetic cost is proportional to the bit-length already.

This algorithm reduces the number of multiplications needed to compute x^c from c to at most $2l$, where l is the number of bits in c:

$$l = \lfloor \log_2(c) \rfloor + 1.$$

- If $c < n$, as in RSA, exponentiation will take $O(\log(n)^3)$ bit operations.

Using faster multiplication algorithms (such as those based on the fast Fourier transform), this can be brought down to nearly $O(\log(n)^2)$ bit operations—effectively reducing the cost from c multiplications to about $2\log(c)$ multiplications.

1.2 Computing $m_1^c \bmod n$

To compute $x^c \bmod n$, the trick is to write c in binary:

$$c = \sum_{i=0}^{l-1} c_i \cdot 2^i, \quad \text{where } c_i \in \mathbb{Z}_2.$$

Then, precompute exponentiations of the form:

- $x^{2^i} \bmod n$ for $i \in \{1, \ldots, l-1\}$.
- These are computed by repeated squaring:

$$x^{2^i} \bmod n = \left(x^{2^{i-1}} \bmod n\right)^2.$$

We can do the calculation quickly, but why do we want to do it? Because:

$$x^c = \prod_{i \in S} x^{2^i}, \quad \text{where } S = \{\, i \mid 0 \le i \le l-1,\ c_i = 1 \,\}.$$

Thus, computing x^c amounts to multiplying together all the precomputed values x^{2^i} for which $c_i = 1$.

This algorithm makes RSA encryption and decryption efficient (although not as efficient as secret key cryptosystems like DES, it is still polynomial time).

RSA key generation is even faster since it involves computing two random primes of k bits:

- Multiplying them takes $O(k^2)$ bit ops.
- One must compute a random pair (a, b), where $a \equiv b^{-1} \bmod \phi(n)$.
- Since $\phi(n)$ is a k-bit number, this modular inversion is also $O(k^2)$.

Thus, once the primes are found, key generation is quadratic in the bit size.

1.3 Finding and Testing Large Primes

But how efficient is finding primes (i.e., finding two large primes)?

In practice, to generate large random primes, one generates large random numbers and tests if they are prime. There exist deterministic primality tests (such as the AKS algorithm) that run in polynomial time without randomness, but these are less efficient than the randomized tests we usually employ:

- It is more efficient to run randomized algorithms several times than to run a deterministic algorithm once.

The primality tests are of "Monte Carlo" type; they are randomized algorithms with a small chance of error:

- If the test reports that a number is *not prime*, the result can be trusted ("No-biased").
- If it reports that a number is prime, there is a small chance it might be wrong; such numbers are called "probable primes".
- A negative result ("not prime") serves as a proof, whereas a positive result is only probabilistic.

If repeated tests continue to report a number as a probable prime, then it is very likely to be prime. One could prove primality deterministically, but the error probability can be made exponentially small with enough random tests.

1.3.1 Prime Number Theorem

How many numbers must be tested before finding a prime? The prime number theorem tells us that the number of primes up to n is approximately

$$\frac{n}{\ln(n)},$$

so one expects to test roughly $\ln(n)$ numbers before finding a prime.

For example, for $n = 2^{1024}$, note that

$$\ln(n) = 1024 \ln(2) \approx 710,$$

so selecting random 1024-bit numbers would, on average, yield a prime after roughly 710 trials. (The odds improve if one avoids even numbers.)

1.3.2 Quadratic Residues

The first Monte Carlo algorithm we will look at uses quadratic residues (QR). Suppose:

- p is an odd prime and $a \in \mathbb{Z}_p^*$ (so a has an inverse modulo p).
- Then a is a quadratic residue modulo p if the congruence

$$y^2 \equiv a \pmod{p}$$

  has a solution (i.e., if a has a square root modulo p); otherwise, a is a non-quadratic residue.

For example, in $\mathbb{Z}_{11}$, we have $4^2 \equiv 5 \pmod{11}$, so 5 is a quadratic residue modulo 11, while 2 is not, as there is no y such that $y^2 \equiv 2 \pmod{11}$.

In general, the congruence

$$x^2 \equiv a \pmod{p}$$

has exactly two solutions. If y is one solution, then $-y$ is the other, so that $x^2 - a$ factors as

$$(x - y)(x + y)$$

in $\mathbb{Z}_p$.

Finally, Euler's criterion provides an efficient method to determine whether a is a quadratic residue modulo p.