RSA (CONTINUED): COMPUTATIONAL ISSUES

MATH 195

Still to be discussed:

- Computing $a^e \pmod n$ for given a, e, n.
- How to generate prime numbers of a given order of magnitude, how to tell whether a given number is prime (primality testing), and how to factor the number if it is not prime.
- "Leaking d enables one to factor n."

Computing powers

There is a 'fast' algorithm that given $n \in \mathbb{Z}_{>0}$, $a \in \mathbb{Z}/n\mathbb{Z}$, $m \in \mathbb{Z}_{>0}$, computes a^m as an element of $\mathbb{Z}/n\mathbb{Z}$.

Here is a really slow way to do it: View a as an element of $\mathbb{Z}$, compute $a^m = a \cdot a \cdot \cdots \cdot a$ using $m-1$ multiplications in $\mathbb{Z}$, and compute $a^m \pmod n$ by division with remainder. This is dreadful, because when $n \approx 10^{200}$, and $m \approx 10^{50}$, $a = 2$, and $a^m = 2^{10^{50}}$: writing $100000\ldots$ will take forever.

One can improve this by doing multiplications in $\mathbb{Z}/n\mathbb{Z}$ (replace every element by its remainder after division with n). But this would still take 10^{50} multiplications! Therefore we group the terms, since squaring will double our exponent.

Therefore when $m = 2^i$, then $a^m = a^{2^i} \in \mathbb{Z}/n\mathbb{Z}$ can be computed from a by performing i squarings in $\mathbb{Z}/n\mathbb{Z}$: at the ith step, we obtain $(a^{2^{i-1}})^2 = a^{2^i}$. For general m, write $m = \sum_{i=0}^{k} b_i 2^i = b_k b_{k-1} \ldots b_1 b_0$ where $b_i \in \{0, 1\}$, $b_k = 1$, $k = \lfloor \log m / \log 2 \rfloor$ (round down). Then

$$a^m = \prod_{\substack{i=0 \\ b_i=1}}^{k} a^{2^i}.$$

For example, if $m = 57 = (111001)_2$, we compute a^1, a^2, a^4, a^8, then multiply to get a^9, then a^{16} and multiply to get a^{25}, then compute a^{32} and multiply to get a^{57}. One can also read this in the other order, which requires the same number of squarings and multiplications: we follow the chain $a^1, a^2, a^3, a^6, a^7, a^{14}, a^{28}, a^{56}, a^{57}$.

Notice that the number of squarings is $k = \lfloor \log m / \log 2 \rfloor$, and the number of multiplications is $\#\{i : b_i = 1\} - 1$.

One can also compute in base 64, with digits between 0 and 63, and then one precomputes a^c for $0 \le c \le 63$.

If one needs to compute many a all to the same exponent (say 57), then we arrive at the theory of *addition chains*: in the chain $1, 2, 3, 6, 7, 14, 28, 56, 57$, or $1, 2, 4, 8, 9, 16, 25, 32, 57$, we insist that every number can be written as the sum of

---

This is some of the material covered February 28, in Math 195: Cryptography, taught by Hendrik Lenstra, prepared by John Voight `jvoight@math.berkeley.edu`.

two of the previous numbers. For example, we could also take $1, 2, 3, 4, 8, 16, 19, 38, 57$ (see Knuth, *The Art of Computer Programming*).

The time taken by this algorithm is $c(\log m)$ multiplications in $\mathbb{Z}/n\mathbb{Z}$. If $t \approx \log n / \log 2$ (the number of bits), and we assume multiplication is proportional to this, then the multiplication (with taking remainder) takes t^2 time, so the total is $c(\log m)(\log n)^2$. (There are actually fast multiplications which take roughly $c(\log n)(\log\log n)$, relying upon the Fast Fourier Transform.) When m is roughly n, then this is $c(\log n)^3$.

As a comparison, consider the (extended) Euclidean algorithm: Suppose a and n have t bits (t proportional to $c \log n$). Then the Euclidean algorithm takes at most ct steps, each step a division-with-remainder of two numbers of $< t$ bits each (up to a constant). Therefore the total time is $\approx ct^2$. With the FFT methods, we can obtain $\approx ct^{1+\epsilon}$.

Leaking d

There is a fast algorithm that given an odd integer $n > 1$, as well as an integer $m > 0$ with the property that

$$\text{for all } a \in (\mathbb{Z}/n\mathbb{Z})^*,\ a^m = 1 \pmod{n},$$

in practice completely factors n into prime factors.

You may assume this fact for the homework.