CHAPTER 6

Where All Roads Meet: Modular Exponentiation

For a long time on that spot I stood,
Where two roads converged in the wood and I thought:
"Someone going the other way
Might someday stop here for the sake
Of deciding which path to take."
But my direction lay where it lay.
And walking on, I felt a sense
Of wonder at that difference.

—Ilya Bernstein, *Attention and Man*

IN ADDITION TO THE CALCULATIONAL rules for addition, subtraction, and multiplication in residue classes we can also define an operation of exponentiation, where the exponent specifies how many times the base is to be multiplied by itself. Exponentiation is carried out, as usual, by means of recursive calls to multiplication: For a in $\mathbb{Z}_m$ we have $a^0 := \overline{1}$ and $a^{e+1} := a \cdot a^e$.

It is easy to see that for exponentiation in $\mathbb{Z}_m$ the usual rules apply (cf. Chapter 1):

$$a^e \cdot a^f = a^{e+f}, \qquad a^e \cdot b^e = (a \cdot b)^e, \qquad (a^e)^f = a^{ef}.$$

6.1 First Approaches

The simplest approach to exponentiation consists in following the recursive rule defined above and multiplying the base a by itself e times. This requires $e - 1$ modular multiplications, and for our purposes that is simply too many.

A more efficient way of proceeding is demonstrated in the following examples, in which we consider the binary representation of the exponent:

$$a^{15} = a^{2^3+2^2+2+1} = \Big(\Big(\big(a^2\big)\,a\Big)^2\,a\Big)^2\,a, \qquad a^{16} = a^{2^4} = \Big(\Big(\big(a^2\big)^2\Big)^2\Big)^2.$$

Here raising the base to the fifteenth power requires only six multiplications, as opposed to fourteen in the first method. Half of these are squarings, which, as we know, require only about half as much CPU time as regular multiplications. Exponentiation to the sixteenth power is accomplished with only four squarings.

Algorithms for exponentiation of a^e modulo m that calculate with the binary representation of the exponent are in general much more favorable than the first approach, as we are about to see. But first we must observe that the intermediate results of many integer multiplications one after the other quickly occupy more storage space than can be supplied by all the computer memory in the world, for from $p = a^b$ follows $\log p = b \log a$, and thus the number of digits of the exponentiated a^b is the product of the exponent and the number of digits of the base. However, if we carry out the calculation of a^e in a residue class ring $\mathbb{Z}_m$, that is, by means of *modular* multiplication, then we avoid this problem. In fact, most applications require exponentiation modulo m, so we may as well focus our attention on this case.

Let $e = (e_{n-1}e_{n-2}\ldots e_0)_2$ with $e_{n-1} > 0$ be the binary representation of the exponent e. Then the following *binary algorithm* requires $\lfloor \log_2 e \rfloor = n$ modular squarings and $\delta(e) - 1$ modular multiplications, where

$$\delta(e) := \sum_{i=0}^{n-1} e_i$$

is the number of ones in the binary representation of e. If we assume that each digit takes on the value 0 or 1 with equal probability, then we may conclude that $\delta(e)$ has the expected value $\delta(e) = n/2$, and altogether we have $\frac{3}{2}\lfloor \log_2 e \rfloor$ multiplications for the algorithm.

Binary algorithm for exponentiation of a^e modulo m

1. Set $p \leftarrow a^{e_{n-1}}$ and $i \leftarrow n-2$.
2. Set $p \leftarrow p^2 \bmod m$.
3. If $e_i = 1$, set $p \leftarrow p \cdot a \bmod m$.
4. Set $i \leftarrow i - 1$; if $i \geq 0$, go to step 2.
5. Output p.

The following implementation of this algorithm gives good results already for small exponents, those that can be represented by the USHORT type.