3. Arithmetic modulo N

Now that we have studied the basic arithmetic operations we can perform computations in the ring $\mathbb{Z}/N\mathbb{Z}$. The objects in this ring are cosets of the form $a+N\mathbb{Z}$ for integers b. By the process of division we can write $a = d \cdot N + r$ for r between 0 and $N-1$. Thus $b+N\mathbb{Z} = r+N\mathbb{Z}$ and so the objects of $\mathbb{Z}/N\mathbb{Z}$ can be identified with integers between 0 and $N=1$. Thus one way to perform arithmetic operations in this ring is to perform them with integers and then, by division, reduce to this canonical form. However, this is uses more space and time than required.

3.1. **Faster arithmetic.** Given a and b in reduced form in $\mathbb{Z}/N\mathbb{Z}$, we note that $a+b$ is between 0 and $2N-2$, while $a-b$ is between $-N+1$ and $N-1$. Thus a simple subtraction (respectively addition) of N can perform the necessary reduction. Thus, for example, addition is performed by comparing $c = a+b$ with N; if it is less than N then we return c otherwise we return $c-N$.

When we multiply a and b we can get a number of size comparable to N^2. Thus it would appear that there is no way faster than to perform division. However, there are alternatives that can be made faster as we shall see. A similar operation is a^b. The following procedure can be applied to additive and multiplicative structures in $\mathbb{Z}/N\mathbb{Z}$ to speed up both operations in some cases.

Let G be a group (or semi-group) and g and element of G. Let k be a natural number. We wish to compute $u = g^k$. We have the recursive identity $g^k = (g^{[k/2]})^2 \cdot g^{(k\%2)}$, where $[k/2]$ is the integer part of $k/2$ and $k\%2$ is 0 or 1 according as k is even or odd.

If G has an identity element 1 (i. e. G is monad). We can proceed as follows. Set $u = 1$, $p = g$. We induct on k, outputting u when k becomes zero. When k is greater than zero, if k is odd, we replace u by $u \cdot p$. Then we cwreplace k by $[k/2]$ and replace p by p^2, the square of p, before induction on k. This is known as right-left powering.

We can use the recursion directly to define g^k in terms of $g^{[k/2]}$ and $g^{(k\%2)}$. Now, the latter is either g or 1, thus we only need to take squares and multiply by a fixed number g. In particular, when g easy to multiply with there is a substantial saving of time and space.

The main hurdle for arithmetic modulo N is division (as usual). To divide by a, we apply the extended GCD to a and N (by one of the earlier algorithms) to get a solution of $xa + yN = d$, where d is the GCD of a and N. Moreover, by appropriately choosing the initialisation we can also ensure that $0 \leq x < N$. Now let $z = xb$. From the above identity we get $az = db$ in $\mathbb{Z}/N\mathbb{Z}$. In particular, if $d = 1$, we see that z is the quotient a/b in this ring.

If k is the order of the group of units (invertible elements) in $\mathbb{Z}/N\mathbb{Z}$, then $a^k = 1$ for any a which is coprime to N. Thus, we see that $a^{k-1} = x$ and this could sometimes be a faster way of computing x. One situation, when k can be computed easily is when we have a factorisation of N.

More generally, in case we have a factorisation $N = m_1 \cdots m_r$ with m_i mutually coprime, then a computationally effective form of the Chinese Remainder theorem allows us to make *all* arithmetic operations faster.

Lemma 5. *Let q_i denote the inverse of the product $(m_1 \cdots m_{i-1})$ in the ring $\mathbb{Z}/m_i\mathbb{Z}$. Given x_i in the ring $\mathbb{Z}/m_i\mathbb{Z}$, we inductively define y_i as $y_1 = x_1$ and*

$$y_j = q_j \left(x_j - \sum_{i<j} (m_1 \cdots m_{i-1}) y_i \right) \pmod{m_j}$$

Then $x = \sum_i (m_1 \cdots m_{i-1}) y_i$ is the unique number between 0 and $N-1$ such that $x = x_i \pmod{m_i}$.

Now that we can compute q_i once and for all independent of the x_i; secondly, q_j. Thus we can do many conversions quickly. Secondly, the sums and products have a structure that permits easy inductive computation (exercise for the reader).

Proof. By construction, we have

$$(m_1 \cdots m_{j-1}) y_j = \left(x_j - \sum_{i<j} (m_1 \cdots m_{i-1}) y_i \right) \pmod{m_j}$$

□

3.2. **Encrypting and Decrypting RSA messages.** We assume for the moment that someone (an "oracle") gives us a modulus N and two numbers p (the public key) and s (the secret key) with the information that in m is any (invertible) element in $\mathbb{Z}/N\mathbb{Z}$, then $m^{ps} = m$ in $\mathbb{Z}/N\mathbb{Z}$. In order to send a message, the message is broken into chunks m of size $N-1$ and each block is encrypted as $c = m^p$ (we will see shortly why we do not expect a problem from non-invertibility of m). This is then sent over the public channel. At the receiving end the operation $m = c^s$ recovers the original message blocks. Thus, the algorithms of the earlier subsection (especially the one about taking powers) can usefully be applied in implementing this encryption/decryption process.

One question is how the triple (N, p, s) is chosen. Clearly, if M is the order of the group of units in $\mathbb{Z}/N\mathbb{Z}$, then one way to recover s given p is by the relation $ps = 1 \pmod{M}$. Thus, one consideration is that given only N and p it should not be easy to determine M (or else s is not secret at all!). Clearly, if a complete factorisation of N is known then this M can be easily determined. However, note that if k is the *exponent* of the group of units in $\mathbb{Z}/N\mathbb{Z}$ (i. e. the least common multiple of the order of all elements) then it is enough to know k rather than M, since solving $ps = 1 \pmod{k}$ is adequate.

A related consideration is that there should not be too many elements in $\mathbb{Z}/N\mathbb{Z}$ which are *not* units. The encrypter/decrypter should not have to worry about checking for this while constructing messages.

Both these considerations lead us to take for $N = ab$ a product of two large primes a and b (we will see some additional constraints on the pair (a, b) as we go along). It is generally expected (and in the next section we will see some evidence of this) that factoring a number which has *all* factors of size (in terms of "log") r or more is much harder than checking that a number of size r is prime. Thus, it should be easier to construct key triples than it is to "break" keys (by determining s given N and p). Note that $M = (p-1)(q-1)$ in this case and in general there does not seem to be a way to determine M (or the exponent k for that matter) without determining a factoring of N. The only elements m in $\mathbb{Z}/N\mathbb{Z}$ that are not units are p and q, thus this choice is also good from this point of view.