Simple facts about modular arithmetic, the Chinese Remainder Theorem and RSA implementation

The Chinese Remainder Theorem (CRT) is an important tool for modular arithmetic computations. It underlies both some simple attacks on careless implementations of RSA (textbook 6.3.4.3) and an important optimization of the exponentiation with the private exponent (6.3.4.4).

While you are reading this, please keep in mind the Fundamental Theorem of Arithmetic that states that the factorization of an integer down to a product of primes is unique, that is, no matter how you factor $n$ to get

$$n = p_1^{\alpha_1} \cdots p_k^{\alpha_k},$$

the set of primes $\{p_1, \ldots, p_k\}$ and their respective powers $\alpha_i$ will be the same. Consequently, the greatest common divisor of two numbers will be exactly the product of the primes their factorizations have in common, each of these primes in the smallest of the two powers in which the prime occurs in these factorizations. Thus it is really easy to find the $\gcd(n_1, n_2)$ when their factorizations are known – but we know no fast way to do factorization, the premise on which RSA is based. Luckily, GCDs can be easily and efficiently found with Euclid’s algorithm, and the same algorithm serves to find multiplicative inverses in modular arithmetic.

Two numbers are called relatively prime or co-prime when their GCD is 1. In view of the Fundamental Theorem of Arithmetic, this means that their (unique) factorizations have no primes in common.

Modular reduction

When we perform modular arithmetic, we do it with regular integers, and some intermediary results may exceed the modulus $n$. However, since the result will have be in the set $\{0, \ldots, n - 1\}$, it pays to reduce the intermediary results as soon as possible. One can think of mod $n$ reduction as performing integer division by $n$ and taking the remainder, or as subtracting multiples of $n$ as many times as necessary for the result to get into the above range (these are really the same thing).

Observe that the results of addition and multiplication mod $n$ do not change whether we reduce the operands before the operation, or the result after the operation (but we’ll save time if we work with reduced operands). For multiplication, let us say we have $a = k \cdot n + \alpha$ and $b = l \cdot n + \beta$ where $\alpha, \beta \in \{0, \ldots, n - 1\}$. Then

$$a \cdot b = (k \cdot n + \alpha) \cdot (b = l \cdot n + \beta) = kl \cdot n + \alpha l \cdot n + \beta k \cdot n + \alpha \beta$$
and all members in this sum except for the last one are multiples of \( n \) and will go away when we reduce the result. We might as well reduce the operands first to \( a \) and \( \beta \) and multiply these to get \( a \cdot b \mod n \). It goes similarly for addition.

It is harder with exponentiation, because we cannot reduce the exponent the same way (find an example of \( a^b \mod n \) not being the same modulo \( n \) as \( \alpha^\beta \mod n \) – indeed, reducing the power \( b \) will not make multiples of \( n \) somehow appear in the base \( a \) or in the result so that they could be conveniently reduced away as above). But Euler’s Theorem helps to reduce the exponents, too – as it happens, the reduction is possible using a different modulus, which is related to \( n \).

**Euler’s Theorem** Let \( a \) and \( n \) be relatively prime. Then

\[
a^{\phi(n)} = 1 \mod n.
\]

Thus we can reduce the exponent by multiples of \( \phi(n) \) without changing the result of the operation, but only when the base \( a \) and the modulus \( n \) are relatively prime. In other words, if \( b = \gamma + k \cdot \phi(n) \), where \( \gamma \in \{0, \ldots, \phi(n) - 1 \} \) then

\[
a^b = a^{\gamma+k \cdot \phi(n)} = a^\gamma \cdot (a^{\phi(n)})^k = a^\gamma \mod n
\]

so we can use the reduced exponent and get the same result of exponentiation, provided that \( \gcd(a, n) = 1 \). We will use that in

**The Chinese Remainder Theorem**

This theorem states that a solution to a system of equations, each of which specifies the remainder \( a_i \) from dividing a number by some \( m_k \) always exists and is unique modulo \( m_1 \cdot \ldots \cdot m_k \), provided that these numbers \( m_k \) are mutually relatively prime, i.e. \( \forall i, j \leq k, i \neq j \gcd(m_i, m_k) = 1 \). That is, there exists an integer \( x \) such that

\[
x = a_1 \mod m_1
\]
\[
x = a_2 \mod m_2
\]
\[
\ldots
\]
\[
x = a_k \mod m_k
\]

and all such integers differ by a multiple of \( n = m_1 \cdot \ldots \cdot m_k \). The proof of CRT is constructive, that is, we write down the formula for computing this \( x \).

The construction goes as follows. Define \( n_i = n/m_i \), that is, \( n_i \) is the product of all \( m_j, j \neq i \). It is easy to see that \( n_i \) is relatively prime with \( m_i \), i.e. \( \gcd(n_i, m_i) = 1 \). This means that \( m_i \) has a multiplicative inverse \( y_i \) modulo \( n_i \), which can be found by running Euclid’s algorithm. Then we have

\[
\forall i = 1, \ldots, k \quad y_i \cdot n_i = 1 \mod m_i,
\]

and also

\[
\forall i, i = 1, \ldots, k, j \neq i \quad y_i \cdot n_i = 0 \mod m_j.
\]
because $a_j$ is a multiple of $m_i$ for all $j \neq i$. We can multiply both parts of these congruences by $a_i$ and we'll get

$$\forall i \quad a_i y_i n_i = a_i \mod m_i, \quad a_i y_i n_i = 0 \mod m_j, \quad j \neq i$$

From these two observations it follows that

$$x = \sum_{i=1}^{k} a_i y_i n_i$$

solves the above system of equations.

Here is an ancient arithmetic problem that CRT can be applied to solve:

An old woman goes to market and a horse steps on her basket and crashes the eggs. The rider offers to pay for the damages and asks her how many eggs she had brought. She does not remember the exact number, but when she had taken them out two at a time, there was one egg left. The same happened when she picked them out three, four, five, and six at a time, but when she took them seven at a time they came out even. What is the smallest number of eggs she could have had? (From http://www.cut-the-knot.org/blue/chinese.shtml)

The application of CRT to modular arithmetic is straightforward. We can break the modulus $n$ into a product of relatively primes numbers $m_k$ (primes or prime powers). If we can compute our expression of interest under smaller moduli $m_k$ (and thus save time by operating with smaller numbers), then we can combine these results $a_i$ to get the result modulo $n = m_1 \cdot \ldots \cdot m_k$. That final step will cost us one addition mod $n$, but all other operations will be cheaper, dealing with smaller moduli.

**Use of CRT to speed up private exponentiation**

Suppose the RSA key length in 512 bytes, that is, we are dealing with 512 integer arithmetic modulo $n = p \cdot q$. Since $p$ and $q$, being prime, are clearly also mutually prime, we can use CRT, to perform exponentiations modulo the smaller $p$ and $q$, and with smaller exponents, too. Supposing that $p$ and $q$ are around 256 bits each, this is a big win, no matter what exponentiation and multiplication algorithms we use. Here is how.

Observe that by the CRT, we can compute $c^d \mod n$ uniquely once we know $c^d \mod p$ and $c^d \mod q$, which will be faster to compute. Also, now that we are using smaller moduli (and less bits), we can use smaller exponents because of Euler’s theorem:

$$c^d \mod p = c^{d \mod \phi(p)} \mod p = c^{d \mod (p-1)} \mod p$$

and analogously for $q$. We can, of course, precompute these smaller exponents $d_p = d \mod (p-1)$ and $d_q = d \mod (q-1)$ to save time.
Supposing that $p$ and $q$ are 256 bit numbers, we can trade one 512 bit exponentiation for 2 256 bit exponentiations as above ($\mod p$ and $\mod q$), 2 256 bit multiplications and one 512 bit addition (from the CTR formula for the solution). This is a big win.