CS 198 Fall 2022 | Codebreaking | Ryan Cottone, Imran Khaliq-Baporia | Note 3

Modular Arithmetic

Modular arithmetic is perhaps the most important subject in all of cryptography. The concept is very simple at face value, but allows for almost every modern cryptosystem to function. It's importance cannot be understated.

If you have taken CS 70, you will have covered this topic (as well as RSA) in some detail already. Unfortunately, this note will not go into as much theory/proofs behind the results as a math class – if you are interested, please do reach out to the instructors or use CS 70's notes.

So, what is modular arithmetic? First we must define the **modulo** operator:

$$(\text{mod } n)$$

Evaluating an expression modulo n is roughly equivalent to taking its remainder divided by n. For example, $5 \pmod 3 \equiv 2 \pmod 3$, since 5 divided by 3 yields a remainder of 2. Note that the mod operator **applies to the entire equation**.

This also applies to negative numbers: $-6 \pmod 5 \equiv -1 \pmod 5 \equiv 4 \pmod 5$. We can only have numbers $0, ..., n-1$ when taking modulo n, so to convert a negative result to a positive one, simply add an n.

Another possibly helpful visualization is thinking of modular arithmetic as "wrapping" around a number, or like a clock face. When the clock reaches 12, it starts over at the beginning. (In this case, 12 would technically be 0).

Some other helpful properties of modular arithmetic:

- $(x \equiv an + x \pmod n)(\forall a \in \mathbb{Z})$
  - Basically, you can add as many multiples of n as you want to the expression – since they reduce to 0 modulo n, it does not affect the value.
- $(x^k \equiv (x \pmod n)^k \pmod n)(\forall k \in \mathbb{Z})$
  - This means you can reduce the base of the exponent modulo n before taking the exponent. For example, $9^{1000} \equiv -1^{1000} \equiv 1 \pmod{10}$
    Note that multiplication and the exponent **are not reducible**.

Multiplicative Inverses

Unfortunately, modular arithmetic is a bit messy and unintuitive when it comes to division. Division, as you know it from grade school, does not exist in most modular fields. While you can

do other algebraic operations like addition, subtraction, multiplication, and exponentiation to both sides of a modular equation freely, you **cannot** divide.

Definition 1: Multiplicative Inverse

The **multiplicative inverse** of a integer $x \pmod{n}$ is an integer x^{-1} such that $x \cdot x^{-1} \equiv 1 \pmod{n}$.

For normal algebra, the multiplicative inverse of any number is its reciprocal – that is, for any number x, $x^{-1} = \frac{1}{x}$. This is what *division* actually is – multiplying by its reciprocal.

In modular arithmetic, this does not work, because we are only working with integers (within the scope of this class). Thus, a fraction cannot exist modulo another integer. However, all hope is not lost – we can still find multiplicative inverses for numbers quite efficiently.

By its definition, to inverse an integer, we need to find another integer that when multiplied by the original evaluates to 1. For example, when we want to solve for x in this modular equation:

$$3x \equiv 5 \pmod{11}$$

Notice that $4 \cdot 3 = 12 \equiv 1 \pmod{11}$
Thus, we use it as our multiplicative inverse:

$$\begin{aligned} 4 \cdot 3x &\equiv 4 \cdot 5 \pmod{11} \\ (4 \cdot 3)x &\equiv 20 \pmod{11} \\ x &\equiv 9 \pmod{11} \end{aligned}$$

It may seem counterintuitive that multiplying by 4 was how we isolated the x. Modular arithmetic can be tricky like that – but you will soon get used to it.

However, there are cases where no multiplicative inverse exists modulo n. For example:

$$2x \equiv 3 \pmod{4}$$

As you can see, there does not exist an integer a such that $2a \equiv 1 \pmod{4}$. Thus, there exists no solution to this modular equation (notice that there is no multiple of two that is equivalent to 4a + 3 for some multiple of 4).

Notably, it turns out that a modular inverse exists for $x \pmod{n}$ *if and only if* $\gcd(x,n) = 1$. What that means is x and n must be relatively prime (aka coprime), or have a greatest common divisor of 1. Put another way, no integer greater than 1 divides both x and n.

Quick reminder of predicate logic: if and only if means that if either side of the equation holds, the other is true as well (aka the two are equivalent). If the gcd of x and n is 1, then a modular inverse exists, and if a modular inverse exists, then the gcd is 1.

Euclid's Extended Algorithm

Thankfully for us, there exists a very efficient method to calculate the gcd of two integers, and also find the modular inverse while we're at it. Enter **Euclid's Extended Algorithm**, with our function extended_gcd(x,y) that returns the gcd and two integers a,b such that $ax+by=gcd(x,y)$.

The full algorithm is outside the scope of this class, though we recommend interested readers to check the CS70 notes or online resources to learn more. For now, we will take this function for granted as being very efficient.

To compute a modular inverse for $x \pmod{n}$, we call extended_gcd$(x,n)=(d,a,b)$. Assuming d=1, we know that $ax+bn=1$. Taking both sides modulo n, $ax=1 \pmod{n}$. Therefore, our modular inverse is a as returned by the extended_gcd function.

Fermat's Little Theorem

One of the biggest results in modular arithmetic is **Fermat's Little Theorem**, hereafter referred to as FLT.

Definition 2: Fermat's Little Theorem

Fermat's Little Theorem states that for prime p,

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

and, by multiplying by x^{-1}..

$$x^{p-1} \equiv 1 \pmod{p}$$

The proof is sadly outside the scope of this class (though it is really quite cool).

This theorem is what underpins all of RSA, one of the most prolific encryption schemes in the world.

The Chinese Remainder Theorem

Definition 3: Chinese Remainder Theorem

The **Chinese Remainder Theorem** (aka CRT) states that for coprime $n_1, n_2, \cdots n_k$, the following system of modular equations has a unique solution for $x \pmod{n_1 \cdot n_2 \cdot \cdots \cdot n_k}$.

$$\begin{aligned} x &\equiv a_1 \pmod{n_1} \\ x &\equiv a_2 \pmod{n_2} \\ &\vdots \\ x &\equiv a_k \pmod{n_k} \end{aligned}$$

such that $x \equiv b \pmod{\prod_{i=1}^{k} n_i}$ for some unique b.

At first glance, this may seem niche or hard to apply. However, we will shortly see just how powerful this theorem can be for working with modular systems.

An example of CRT is as follows:

$$\begin{aligned} x &\equiv 3 \mod 5 \\ x &\equiv 4 \mod 7 \end{aligned}$$

We want to find x_1 such that $x_1 \equiv 1 \mod 5$ and $x_1 \equiv 0 \mod 7$. The former is accomplished by finding the inverse of m_2 (in this case, 7) mod m_1 (5) (such that $7 \cdot 7^{-1} \equiv 1 \mod 5$), the latter by scaling the former by 7. Remember, anything multiplied by m is $0 \mod m$.

We find x_2 in a similar fashion, but this time $0 \mod 5$ and $1 \mod 7$. Finally, we setup our final expression $3x_1 + 4x_2$. We can see that this value fulfills both of our original equations. Therefore, it is the unique value for the system $\mod 5 \cdot 7$.

In our case, that is $3(7 \cdot 3) + 4(5 \cdot 3) = 18 \mod 35$