Module 4: Number Theory

Integers and Division

Basic definitions

- **Number theory** is the branch of mathematics that deals with properties of integers.
- For $a, b \in \mathbb{Z}$ such that $a \neq 0$, a **divides** b if there exists $c \in \mathbb{Z}$ such that $b = ac$. We write $a|b$.
- If $a|b$, then a is a **factor** of b and b is a **multiple** of a.
- Let $a, b \in \mathbb{Z}$ and $d \in \mathbb{Z}^+$. a is **congruent** to b **modulo** m if $m|(a-b)$. We write $a \equiv b \pmod{m}$.

Theorem 1. Let $a, b, c \in \mathbb{Z}$. Then the following hold:

$$\begin{array}{ll} (1) & \text{if } a|b \text{ and } a|c, \text{ then } a|(b+c); \\ (2) & \text{if } a|b, \text{ then } a|bc\ \forall c \in \mathbb{Z}; \\ (3) & \text{if } a|b \text{ and } b|c, \text{ then } a|c. \end{array}$$

Proof of (1): Assume $a|b$ and $a|c$. Then $\exists x, y \in \mathbb{Z}$ such that $b = ax$ and $c = ay$. So $b+c = ax+ay = a(x+y)$. Clearly, $x + y \in \mathbb{Z}$, so $a|(b+c)$. $\square$

Corollary. If $a, b, c \in \mathbb{Z}$ such that $a|b$ and $a|c$, then $a|mb + nc$ whenever $m, n \in \mathbb{Z}$.

The Division "Algorithm"

Theorem 2 (Division "Algorithm"). Let $a \in \mathbb{Z}$ and $d \in \mathbb{Z}^+$. Then there are unique integers q and r, with $0 \leq r < d$, such that $a = dq + r$.

- d is called the **divisor**.
- a is called the **dividend**.
- q is called the **quotient**.
- r is called the **remainder**.

There are a number of alternate ways to write the equation $a = dq + r$ in the Division Algorithm.

$$\begin{array}{rl} q = a \textbf{ div } d & \quad q = a/d \\ r = a \textbf{ mod } d & \quad r = a\%d \end{array}$$

The Division Algorithm formalizes basic division that you have seen since elementary school. For instance, if you divide 23/6, you get a quotient of 3 with a remainder of 5. In terms of the Division Algorithm, we write $23 = 6 \cdot 3 + 5$.

Modular arithmetic

Many applications in computer science rely on **modular arithmetic**. The following theorems illustrate some key properties of modular arithmetic.

Theorem 3. Let $a, b \in \mathbb{Z}$ and $m \in \mathbb{Z}^+$. Then $a \equiv b \pmod{m}$ if and only if $a \pmod{m} = b \pmod{m}$.

Theorem 4. Let $m \in \mathbb{Z}^+$. Then $a \equiv b \pmod{m}$ if and only if $\exists k \in \mathbb{Z}$ such that $a = b + km$.

Theorem 5. Let $m \in \mathbb{Z}^+$. If $a \equiv b \pmod{m}$ and $c \equiv d \pmod{m}$, then

$$a + c \equiv b + d \pmod{m}$$
$$ac \equiv bd \pmod{m}$$

Corollary. Let $a, b \in \mathbb{Z}$ and $m \in \mathbb{Z}^+$. Then

$$a + b \equiv (a \bmod m) + (b \bmod m) \bmod m$$
$$ab \equiv (a \bmod m)(b \bmod m) \bmod m$$

Informally, Theorem 5 and its Corollary tell us that the order of arithmetic and modulo operations is irrelevant. That is, to compute $((a + b) \cdot c) \pmod{m}$, we can first calculate $(a + b) \cdot c$ normally and then perform the modulo congruence. Alternatively, we can start by performing the modulo congruence on each of a, b, and c individually, then perform the arithmetic. Provided all operations are performed correctly, the result will be identical.

Hash functions

A **hash function** defines a procedure for taking an unbounded value and translating it into a small subset of bounded values. One use of a hash function is to take a key k that uniquely identifies a data record, and convert the key to a memory location that is represented by $h(k)$. Modular arithmetic offers a simple method for creating a hash function, assuming there are m possible memory locations:

$$h(k) = k \pmod{m}$$

For example, we could use an array of size 75 to store information for students in CS 18200. To find a student's record, we use Purdue ID numbers as the keys and $m = 75$. The student whose Purdue ID is 01234-56789 (which we use as the number 0123456789) has their record stored at location 0123456789 (mod 75) $\equiv 39$, while the record for 00112-23344 would be at location 30.

Pseudo-random number generators

A number of computer programs need a way to generate random numbers. However, in practice, creating truly random numbers is extremely difficult. Instead, computer programs rely on **pseudo-random numbers**, which are sequences of numbers that *seem* to be randomly chosen, even though there is a very precise algorithm at work.

One way to construct a sequence of random numbers $\{x_n\}$ is to start with four integers: a **modulus** m, **multiplier** a, **increment** c, and a **seed** x_0, with $2 \le a < m$, $0 \le c < m$, $0 \le x_0 < m$, and $0 \le n \le m$. The sequence of pseudo-random numbers is generated by computing:

$$x_{n+1} = (ax_n + c) \pmod{m}$$

If m, a, c, and x_0 are not chosen carefully, the sequence will not be good. For instance, let $x_0 = 2, a = 3, c = 0$, and $m = 6$, the sequence would consist of:

$$x_0 = 2, x_1 = 3 \cdot 2 + 0 \pmod{6} \equiv 0, x_2 = 0, x_3 = 0, \ldots$$

A good pseudo-random sequence with a **full period** (*i.e.,* no repeats until all m possible numbers have been used) can be created if and only the following three conditions hold:

1. Both m and c are relatively prime, *i.e.,* $\gcd(m, c) = 1$.
2. If q is prime and $q|m$, then $q|(a - 1)$.
3. If $4|m$, then $4|(a - 1)$.

Cryptology

Cryptology is the study of secret messages. One of the earliest algorithms for encrypting messages was invented by Julius Caesar. The simple encryption process can be mathematically defined by treating each letter as a number ('a' = 0, 'b' = 1, etc.) and applying the following function:

$$f(p) = (p + 3) \pmod{26}$$

To decrypt the message, apply the inverse function:

$$f^{-1}(c) = (c - 3) \pmod{26}$$

Hence, the encrypted version of "hello to you world," would be "khoor wr brx zruog." This scheme is far too simple and easy to break for modern use. At the end of this module, we will explore a modern **public key cryptosystem** that is also based on modular arithmetic, but is significantly more secure.

Primes and Greatest Common Divisors

Basic definitions

- A positive integer $p > 1$ is **prime** if its only factors are 1 and p.
- A positive integer $n > 1$ is **composite** if it is not prime.
- Note that 1 is neither prime nor composite.

Theorem 1 (Fundamental Theorem of Arithmetic). Every positive integer $n > 1$ can be written *uniquely* as a product of primes, with the primes written in increasing order. That is,

$$n = p_1^{e_1} p_2^{e_2} \dots p_m^{e_m}$$

where $p_1 < p_2 < \dots p_m$ are all primes, and e_i is the exponent of each prime.

Theorem 2. If n is a composite integer, then n has a prime divisor less than or equal to $\sqrt{n}$.
Proof: By definition, n has a factor a such that $1 < a < n$. Furthermore, there must be a b such that $1 < b < n$ and $n = ab$. If either $a \leq \sqrt{n}$ or $b \leq \sqrt{n}$, then the theorem holds. Assume $a > \sqrt{n}$ and $b > \sqrt{n}$. Then, $ab > \sqrt{n} \cdot \sqrt{n} = n$, which is a contradiction. Thus, either a or b must be less than or equal to $\sqrt{n}$. Assume, without loss of generality, $a < \sqrt{n}$. Either a is prime or it can be written as a product of primes (all of which must be less than a. Hence, there must be a prime factor less than or equal to $\sqrt{n}$. □

Theorem 3. There are infinitely many primes.
Proof: Assume the number of primes is finite. That is, all primes can be enumerated as $p_1, p_2, \dots, p_m$. Let $n = p_1 \cdot p_2 \cdot \dots \cdot p_m + 1$. As all primes must be greater than or equal to 2, clearly, n has no prime factor. Hence, n must be prime. However, this contradicts the claim that $p_1, p_2, \dots, p_m$ was the entire list of primes. Thus, by contradiction, there are infinitely many primes. □

A special type of prime, called a **Mersenne prime**, can be written as the form $2^p - 1$, where p is prime.

Greatest common divisor

- For $a, b \in \mathbb{Z}$, not both zero, the **greatest common divisor** d, denoted $\gcd(a, b)$, is the largest integer such that $d|a$ and $d|b$.
- If $\gcd(a, b) = 1$, then a and b are **relatively prime**.
- A list of elements $a_1, a_2, \dots, a_k$ are **pairwise relatively prime** if $\gcd(a_i, a_j) = 1$ for $1 \leq i < j \leq k$.
- For $a, b \in \mathbb{Z}$, not both zero, the **least common multiple** m, denoted $\operatorname{lcm}(a, b)$, is the smallest integer such that $a|m$ and $b|m$.

Theorem 4. Let $a, b \in \mathbb{Z}^+$. Then $ab = \gcd(a, b) \cdot \operatorname{lcm}(a, b)$.

Lemma. Let $a = bq + r$, where a, b, c, and r are integers. Then $\gcd(a, b) = \gcd(b, r)$.

The **Euclidean algorithm** uses this lemma to create a very efficient means of computing the GCD.

Algorithm 1: The Euclidean Algorithm

```
Input: $m, n \in \mathbb{Z}^+$
Output: $x = \gcd(m, n)$
1 $x \leftarrow m$
2 $y \leftarrow n$
3 while $y \neq 0$ do
4     $r \leftarrow x \pmod{y}$
5     $x \leftarrow y$
6     $y \leftarrow r$
7 end
8 return $x$
```

As an example, consider $\gcd(662, 414)$. By the Euclidean algorithm, we have:

$$\begin{array}{rcll} \gcd(662,414) & = & \gcd(414,248) & [662 = 414 \cdot 1 + 248] \\ & = & \gcd(248,166) & [414 = 248 \cdot 1 + 166] \\ & = & \gcd(166,82) & [248 = 166 \cdot 1 + 82] \\ & = & \gcd(82,2) & [166 = 82 \cdot 2 + 2] \\ & = & \gcd(2,0) & [82 = 2 \cdot 41 + 0] \\ & = & 2 & \end{array}$$

Integers and Algorithms

Numeric representation

Observe that $6172 = 6 \cdot 10^3 + 1 \cdot 10^2 + 7 \cdot 10 + 2$. This can be generalized such that any integer n can be written such that $n = a_k b^k + a_{k-1} b^{k-1} + \ldots + a_1 b + a_0$, where $0 < a_k < b$ and $0 \leq a_i < b$ for $0 \leq i < k$. The value b is called the **base** or **radix**. This representation is called the **base b expansion of n**.

- The **binary expansion** of an integer uses the base 2. A **bit** is a single binary digit.
- The **octal expansion** of an integer uses the base 8.
- The **hexadecimal expansion** of an integer uses the base 16. In that case, the coefficients are from the list $\{0, 1, \ldots, 9, A, B, \ldots, F\}$. A single hexidecimal digit is equivalent to four bits.
- A **byte** is two hexidecimal digits.

To make the base explicit, we use a subscript. For instance, $A4_{16} = 164_{10} = 244_8$. The following algorithm describes the process of converting an integer to a base b representation.

Algorithm 2: Conversion to base b representation of n

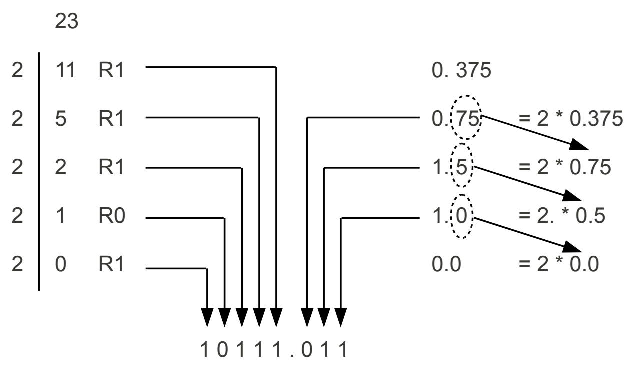
```
Input: $n \in \mathbb{Z}, b \in \mathbb{Z}^+$ such that $b > 1$
Output: The base $b$ expansion of $n$ as $(a_k, a_{k-1}, \ldots, a_0)$
1 $q \leftarrow n$
2 $i \leftarrow 0$
3 while $q \neq 0$ do
4     $a_i \leftarrow q \pmod{b}$
5     $q \leftarrow \lfloor q/b \rfloor$
6     $i \leftarrow i + 1$
7 end
8 return $(a_k, a_{k-1}, \ldots, a_0)$
```

We can expand this process to facilitate non-integer real numbers, too. For simplicity, we will current explore only the conversion of decimal to binary representation. The process can be adapted for other bases as needed. Let $n = a_k a_{k-1} \dots a_0.d_1 d_2 \dots d_j = a_k 10^k + a_{k-1} 10^{k-1} + \dots + a_0 + d_1 10^{-1} + d_2 10^{-2} + \dots + d_j 10^{-j}$. We convert the a_i values as above, ending with a decimal point. In the following algorithm, we ignore the a_i values and treat n as if $0 < n < 1$.

Algorithm 3: Conversion of decimal portion of a real number to binary

```
  Input: $n \in \mathbb{R}$, such that $n = 0.d_1 d_2 \dots d_j = d_1 10^{-1} + d_2 10^{-2} + \dots + d_j 10^{-j}$
  Output: The binary expansion of $n$ as $(a_1, a_2, \dots a_m)$ where
          $n = 0.a_1 a_2 \dots a_m = a_1 2^{-1} + a_2 2^{-2} + \dots + a_m 2^{-m}$
1 $q \leftarrow n$
2 $i \leftarrow 1$
3 while $q \neq 0$ do
      // Multiply $n$ by 2 to get a new decimal value
4     $a_i \leftarrow \lfloor q \cdot 2 \rfloor$        // $a_i$ is the part of the product to the left of the decimal
5     $q \leftarrow q \cdot 2 - a_i$                                  // $q$ gets what is leftover
6     $i \leftarrow i + 1$
7 end
8 return $(a_1, a_2, \dots, a_m)$
```

The next figure shows these two algorithms at work, converting a decimal number to binary.



$$(23.375)_{10} = (10{,}111.011)_2$$

Integer operations

Basic arithmetic operations work exactly the same, regardless of the base used. For instance, consider $123_{10} + 58_{10} = 1{,}111{,}011_2 + 111{,}010_2$:

$$\begin{array}{cccc} & & 1 & \\ & 1 & 2 & 3 \\ + & & 5 & 8 \\ \hline & 1 & 8 & 1 \end{array} \quad \leftarrow \quad \textit{carries} \quad \rightarrow \quad \begin{array}{cccccccc} 1 & 1 & 1 & 1 & & 1 & & \\ & 1 & 1 & 1 & 1 & 0 & 1 & 1 \\ + & & 1 & 1 & 1 & 0 & 1 & 0 \\ \hline 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 \end{array}$$

On your own, you can verify that subtraction, multiplication, and division also work identically.

Modular exponentiation

Modular exponentiation is the process of applying modular arithmetic to computing a value raised to a power. That is, we want to compute $x = b^n \pmod{m}$. The naïve approach would be to first compute the power b^n normally, then apply the modulo congruence. However, as n increases, this approach becomes infeasible. Consider, for example, that 16^{100} produces an integer that is 121 digits long. Imagine how large the number would be if n were 10,000.

To get reasonable performance in modular exponentiation, we rely on two tricks. First, recall that the order of arithmetic and modulo operations does not matter. That is, we can alternate between performing a step of multiplication and a step of modulo congruence. In this approach, we are using a sort of "running value" that never exceeds m^2. This improves the amount of space in memory required for the computation.

The second optimization, which improves the time required for the computation, manipulates the binary expansion of the exponent $n = a_{k-1} \cdot 2^{k-1} + a_{k-2} \cdot 2^{k-2} + \ldots + a_1 \cdot 2 + a_0$, where each a_i is either 1 or 0. That is,

$$x = b^n \pmod{m} = b^{a_{k-1} \cdot 2^{k-1} + a_{k-2} \cdot 2^{k-2} + \ldots + a_1 \cdot 2 + a_0} \bmod m = b^{a_{k-1} \cdot 2^{k-1}} \cdot b^{a_{k-2} \cdot 2^{k-2}} \cdot \ldots \cdot b^2 \cdot b \bmod m$$

Note that all of the factors where $a_i = 0$ just "disappear." *I.e.,* $b^{0 \cdot 2^i} = b^0 = 1$. As a result, we do not have to perform n computations. Instead, we start with b and repeatedly square the value k times. If $a_i = 1$, then we multiply the current squared value against a running product. The following algorithm describes the process.

Algorithm 4: Modular Exponentiation Algorithm

```
  Input: $b, n, m \in \mathbb{Z}$, where $n = (a_{k-1}a_{k-2} \ldots a_1 a_0)_2$ and $k = \lceil \log_2 n \rceil$
  Output: $x = b^n \pmod{m}$
1 $x \leftarrow 1$
2 $power \leftarrow b \pmod{m}$
3 for $i \leftarrow 0$ to $k - 1$ do
4     if $a_i = 1$ then
5         $x \leftarrow (x \cdot power) \pmod{m}$
6         $power \leftarrow (power \cdot power) \pmod{m}$
7     end
8 end
9 return $x$
```

As an illustration, consider $x = 7^{13} \pmod{17}$. Since $13 = 8 + 4 + 1 = 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0$, we can compute the modular exponentiation as:

$$x = 7^{8+4+1} \pmod{17} = 7^8 \cdot 7^4 \cdot 7 \pmod{17} = 16 \cdot 4 \cdot 7 \pmod{17} \equiv 6$$

This works because

$$\begin{aligned}
&7^1 = 7 \\
&7^2 = 49 \equiv 15 \pmod{17} \\
&7^4 = (7^2)^2 \equiv 15^2 \pmod{17} = 225 \pmod{17} \equiv 4 \pmod{17} \\
&7^8 = (7^4)^2 \equiv 4^2 \pmod{17} = 16 \pmod{17}.
\end{aligned}$$

In contrast, without the two optimization tricks, we would have to first compute that $7^{13} = 96,889,010,407$, then do the modulo congruence. Also, note that we did not have to multiple 7 by itself 13 times.

Applications of Number Theory

Linear Combinations

Before delving into the RSA cryptosystem, we need to explore some useful mathematical ideas.

Theorem 1. *If $a, b \in \mathbb{Z}^+$, then there exist $s, t \in \mathbb{Z}$ such that $\gcd(a, b) = sa + tb$. That is, the GCD can be expressed as a* **linear combination***.*

To apply Theorem 1, consider the use of the Euclidean algorithm to compute $\gcd(396, 504)$:

$$\begin{aligned}
504 &= 396 + 108 \\
396 &= 3 \cdot 108 + 72 \\
108 &= 72 + 36 \\
72 &= 2 \cdot 36
\end{aligned}$$

Thus, gcd(396,504) = 36. To find the desired s and t for the linear combination, we use the equations above (starting with the second to last) and work backward:

$$\begin{aligned}
\text{gcd}(396{,}504) \;= 36 &= 108 - 72 \\
&= 108 - (396 - 3 \cdot 108) = 4 \cdot 108 - 396 \\
&= 4\,(504 - 396) - 396 = 4 \cdot 504 - 5 \cdot 396 \\
&= (-5)a + (4)b
\end{aligned}$$

Therefore, if $a = 396, b = 504$, then $s = -5$ and $t = 4$ produces the desired linear combination.

Lemma 1. If a, b, c are positive integers such that $\gcd(a, b) = 1$ and $a|bc$, then $a|c$.
Proof: Since $\gcd(a, b) = 1$, there exist s, t such that $sa + tb = 1$. If we multiply both sides by c, we get $sac + tbc = c$. Since $a|bc$, $a|tbc$. Furthermore, $a|(sac + tbc)$. Thus, $a|c$. □

Lemma 2. If p is a prime and $p|a_1a_2 \ldots a_n$, where each a_i is an integer, then $p|a_i$ for some i.

Theorem 2. Let $m \in \mathbb{Z}^+$ and let $a, b, c \in \mathbb{Z}$. If $ac \equiv bc \pmod m$ and $\gcd(c, m) = 1$, then $a \equiv b \pmod m$.
Proof: Since $ac \equiv bc \pmod m$, $m|ac - bc = c(a - b)$. By Lemma 1 and the assumption that $\gcd(c, m) = 1$, $m|(a - b)$. Hence, $a \equiv b \pmod m$. □

<u>**Linear congruences**</u>
A congruence of the form $ax \equiv b \pmod m$ is called a **linear congruence**. One important such linear congruence is $\overline{a}a \equiv 1 \pmod m$. The integer $\overline{a}$ is called the **inverse** of a modulo m.

Theorem 3. If a and m are relatively prime integers and $m > 1$, then there exists a unique inverse such that $\overline{a}a \equiv 1 \pmod m$.
Proof: Since $\gcd(a, m) = 1$, there exist $s, t \in \mathbb{Z}$ such that $sa + tm = 1$. That is $sa + tm \equiv 1 \pmod m$. Since $tm \equiv 0 \pmod m$, $sa \equiv 1 \pmod m$. Thus, s is an inverse of a. We will show that s is unique by contradiction. Assume s is not unique and there exists another $s' \not\equiv s \pmod m$ such that $s'a \equiv 1 \pmod m$. That is, $sa \equiv s'a \pmod m$. By Theorem 2, $s' \equiv s \pmod m$, which contradicts our assumption. Thus, s is a unique inverse.

<u>**Chinese Remainder Theorem**</u>
For a very large value of m, computing the modulo congruence of a value x can be very time-consuming. The **Chinese Remainder Theorem** offers a technique that significantly improves the performance of this computation.

Theorem 4 (Chinese Remainder Theorem). Let $m_1, m_2, \ldots, m_n$ be pairwise relatively prime positive integers and $a_1, a_2, \ldots, a_n$ be arbitrary integers. Then the system of linear congruences:

$$\begin{aligned}
x &\equiv a_1 \pmod{m_1} \\
x &\equiv a_2 \pmod{m_2} \\
&\quad\cdots \\
x &\equiv a_n \pmod{m_n}
\end{aligned}$$

has a unique solution modulo $m = m_1m_2 \ldots m_n$.
Proof: For $k = 1, 2, \ldots, n$, define

$$M_k = \frac{m}{m_k} = m_1 \ldots m_{k-1}m_{k+1} \ldots m_n.$$

Note that $\gcd(M_k, m_k) = 1$ (since all m_i are pairwise relatively prime). By Theorem 3, each M_k has a unique inverse $y_k \pmod{m_k}$. That is, $M_k y_k \equiv 1 \pmod{m_k}$. To construct the solution, define

$$x = a_1 M_1 y_1 + a_2 M_2 y_2 + \ldots + a_n M_n y_n \pmod{m}.$$

Observe that $M_j \equiv 0 \pmod{m_k}$ for $j \neq k$. But $x \equiv a_k M_k y_k \equiv a_k \pmod{m_k}$ since $M_k y_k \equiv 1 \pmod{m_k}$. Thus, x is a simultaneous solution of all of the given linear congruences.

As an illustration, consider $m = 99 \cdot 98 \cdot 97 \cdot 95 = 89,403,930$. We can represent $x = 123,684$ (mod 89,403,930) using the Chinese Remainder Theorem as:

$$\begin{aligned} 123{,}684 &\equiv 33 \pmod{99} \\ 123{,}684 &\equiv 8 \pmod{98} \\ 123{,}684 &\equiv 9 \pmod{97} \\ 123{,}684 &\equiv 89 \pmod{95} \end{aligned}$$

That is, we can say that the tuple (33,8,9,89) represents the value 123,684 (mod 89,403,930). Similarly, (32,92,42,16) represents 413,456 (mod 89,403,930). If we wanted to add these two values, we could first calculate 123,684 + 413,456 = 537,140, or we could simply do modular arithmetic on each corresponding entry in the tuple to get (65,2,51,10). Note that the second entry is 2, because $8 + 92 = 100 \equiv 2 \pmod{98}$.

Theorem 5 (Fermat's Little Theorem). If p is prime and a is an integer not divisible by p, then $a^{p-1} \equiv 1 \pmod{p}$. Furthermore, for every integer a, $a^p \equiv a \pmod{p}$.

The RSA Cryptosystem

We now have all of the tools that we need to explore the **RSA public key cryptosystem**. The name RSA is built from the initials of the inventors (Ron Rivest, Adi Shamir, Len Adleman). RSA is based on modular exponentiation. Specifically, the modulus $n = pq$, where p and q are both prime numbers.

While many cryptosystems use a single key (*i.e.,* the same key that is used to encrypt a message is used for decryption, too), RSA uses two: a **private key** that must be kept securely and a **public key** that can be made available to anyone in the world. These keys are constructed as follows.

- Select the private key e that is relatively prime to $\phi = (p-1)(q-1)$. That is,

$$\gcd(e, \phi) = 1$$

- The public key d is e's inverse modulo ϕ. That is,

$$de \equiv 1 \pmod{\phi}$$

Note that anyone who knows p and q can compute d using the Euclidean algorithm. Hence, these values must be kept secret. As it turns out, if you do not know p and q, computing d will take so long that you are just as likely to guess it with blind luck. That is, without p and q, you essentilly cannot find d. Using these keys, a user could encrypt the message M as follows:

$$C \equiv M^e \pmod{n}$$

Common mistake: Be careful with the modulus. Above, we had $de \equiv 1 \pmod{\phi}$, but here we have $C \equiv M^e \pmod{n}$.

To decrypt the encrypted text C, we apply the public key d:

$$C^d \pmod{n} \equiv M$$

As an example, we will encrypt the message *STOP* using $p = 43$ and $q = 59$. This gives us $n = 43 \cdot 59 = 2537$ and $\phi = 42 \cdot 58 = 2436$. If we choose the private key $e = 13$, we can use the Euclidean algorithm to compute its inverse $d = 937$. Then, using a numerical version of the alphabet ($A = 00, B = 01, \ldots Z = 25$), we convert *STOP* into "1819 1415." We can then encrypt the blocks as:

$$\begin{aligned} 1819^{13} \pmod{2537} &\equiv 2081 \\ 1415^{13} \pmod{2537} &\equiv 2182 \end{aligned}$$

That gives us a **ciphertext** of "2081 2182." (This message does not break up nicely into individual letters, but that is okay. Encrypted messages never do.) Now, we can use the public key d to decrypt the ciphertext and recover the message:

$$\begin{aligned} 2081^{937} \pmod{2537} &\equiv 1819 \\ 2182^{937} \pmod{2537} &\equiv 1415 \end{aligned}$$

The mathematics that we have been studying lay the foundation for why this cryptosystem works. Specifically, recall that $de \equiv 1 \pmod{\phi}$ means $de = 1 + k\phi$ for some k. That means, by applying Fermat's Little Theorem,

$$\begin{aligned} C^d &= (M^e)^d \\ &= M^{ed} \\ &= M^{1+k\phi} \\ &= M \cdot M^{k(p-1)(q-1)} \\ &= M \cdot (M^{(p-1)})^{k(q-1)} \\ &= M \cdot 1^{k(q-1)} \\ &\equiv M \pmod{p} \end{aligned}$$

and

$$\begin{aligned} C^d &= (M^e)^d \\ &= M^{ed} \\ &= M^{1+k\phi} \\ &= M \cdot M^{k(p-1)(q-1)} \\ &= M \cdot (M^{(q-1)})^{k(p-1)} \\ &= M \cdot 1^{k(p-1)} \\ &\equiv M \pmod{q} \end{aligned}$$

Furthermore, this gives us a system of equations:

$$\begin{aligned} M &\equiv C^d \pmod{p} \\ M &\equiv C^d \pmod{q} \end{aligned}$$

Since p and q are pairwise relatively prime, there must be a unique solution M modulo $pq = n$ (by the Chinese Remainder Theorem). Thus, applying the exponent d successfully decrypts the message.

Matrices

Basic definitions

- An $m \times n$ **matrix A** is a rectangular array of numbers with m rows and n columns, and looks like the following:

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & \ldots & a_{1n} \\ a_{21} & a_{22} & \ldots & a_{2n} \\ . & . & . & . \\ . & . & . & . \\ . & . & . & . \\ a_{m1} & a_{m2} & \ldots & a_{mn} \end{bmatrix}$$

- The i^{th} row of $\mathbf{A}$ is the $1 \times n$ matrix $[\ a_{i1}, a_{i2}, \ldots a_{in}\ ]$, and the j^{th} column of $\mathbf{A}$ is the $m \times 1$ matrix

$$\begin{bmatrix} a_{1j} \\ a_{2j} \\ \cdot \\ \cdot \\ \cdot \\ a_{mj} \end{bmatrix}$$

- We can use the short-hand notation $\mathbf{A} = [a_{ij}]$ to indicate the array that has the element A_{ij} in the j^{th} column of the i^{th} row.

Matrix arithmetic

Let $\mathbf{A} = [a_{ij}]$ and $\mathbf{B} = [b_{ij}]$ both be $m \times n$ matrices. Then the sum $\mathbf{A} + \mathbf{B}$ is the $m \times n$ matrix $\mathbf{A} + \mathbf{B} = [a_{ij} + b_{ij}]$. That is,

$$\mathbf{A} + \mathbf{B} = \begin{bmatrix} a_{11} + b_{11} & a_{12} + b_{12} & \ldots & a_{1n} + b_{1n} \\ a_{21} + b_{21} & a_{22} + b_{22} & \ldots & a_{2n} + b_{2n} \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ a_{m1} + b_{m1} & a_{m2} + b_{m2} & \ldots & a_{mn} + b_{mn} \end{bmatrix}$$

Let $\mathbf{A} = [a_{ij}]$ be a $m \times k$ matrix and $\mathbf{B} = [b_{ij}]$ be a $k \times n$ matrix. Then the product $\mathbf{AB}$ is the $m \times n$ matrix $\mathbf{AB} = [c_{ij}]$, where

$$c_{ij} = a_{i1}b_{1j} + a_{i2}b_{2j} + \ldots + a_{ik}b_{kj}$$

Note that the number of columns in $\mathbf{A}$ must match the number of rows in $\mathbf{B}$. Consider the following example.

$$\mathbf{A} = \begin{bmatrix} 2 & 1 & 3 \\ 0 & 0 & 1 \\ 4 & -1 & 2 \\ 1 & 1 & 1 \end{bmatrix} \quad \text{and} \quad \mathbf{B} = \begin{bmatrix} 1 & 2 \\ 3 & 1 \\ 6 & 1 \end{bmatrix}$$

To compute $\mathbf{AB} = [c_{ij}]$, we start with $c_{11} = 2 \cdot 1 + 1 \cdot 3 + 3 \cdot 6 = 23$. Next, $c_{12} = 2 \cdot 2 + 1 \cdot 1 + 3 \cdot 1 = 8$. Continuing in this way, we get

$$\mathbf{AB} = \begin{bmatrix} 23 & 8 \\ 6 & 1 \\ 13 & 9 \\ 10 & 4 \end{bmatrix}$$

Identity matrix and transposes

The **identity matrix of order** n, denoted $\mathbf{I}_n = [\delta_{ij}]$, where $\delta_{ij} = 1$ if $i = j$, and $\delta_{ij} = 0$ otherwise. That is, the identity matrix looks like this:

$$\mathbf{A} = \begin{bmatrix} 1 & 0 & 0 & \ldots & 0 \\ 0 & 1 & 0 & \ldots & 0 \\ \cdot & \cdot & \cdot & & \cdot \\ \cdot & \cdot & \cdot & & \cdot \\ \cdot & \cdot & \cdot & & \cdot \\ 0 & 0 & 0 & \ldots & 1 \end{bmatrix}$$

For any $m \times n$ matrix $\mathbf{A}$, we have

$$\mathbf{AI}_n = \mathbf{I}_m\mathbf{A} = \mathbf{A}$$

Let $\mathbf{A}$ be an $m \times n$ matrix. The **transpose** of $\mathbf{A}$, denoted $\mathbf{A}^t$, is the matrix $\mathbf{A}^t = [b_{ji}]$, where $b_{ji} = a_{ij}\ \forall i, j$. For example, if

$$\mathbf{A} = \begin{bmatrix} 2 & 1 \\ 0 & 3 \\ 1 & 1 \end{bmatrix} \quad \text{then} \quad \mathbf{A}^t = \begin{bmatrix} 2 & 0 & 1 \\ 1 & 3 & 1 \end{bmatrix}$$

A square (*i.e.,* $n \times n$) matrix $\mathbf{A}$ is **symmetric** if $\mathbf{A} = \mathbf{A}^t$.

Zero-one matrices

A **zero-one matrix** has entries that can only be either 1 or 0. We can operate on zero-one matrices using bit manipulations with Boolean operators on the operators:

$$b_1 \wedge b_2 = \begin{cases} 1 & \text{if } b_1 = b_2 = 1 \\ 0 & \text{otherwise} \end{cases}$$

$$b_1 \vee b_2 = \begin{cases} 1 & \text{if } b_1 = 1 \text{ or } b_2 = 1 \\ 0 & \text{otherwise} \end{cases}$$

Given two $m \times n$ zero-one matrices $\mathbf{A} = [a_{ij}]$ and $\mathbf{B} = [b_{ij}]$, we can compute the **join** $\mathbf{A} \vee \mathbf{B} = [a_{ij} \vee b_{ij}]$, or the **meet** $\mathbf{A} \wedge \mathbf{B} = [a_{ij} \wedge b_{ij}]$. For example, if

$$\mathbf{A} = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix} \quad \text{and} \quad \mathbf{B} = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 1 \end{bmatrix}$$

then,

$$\mathbf{A} \vee \mathbf{B} = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 1 & 1 \end{bmatrix} \quad \text{and} \quad \mathbf{A} \wedge \mathbf{B} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Let $\mathbf{A} = [a_{ij}]$ be a $m \times k$ zero-one matrix and $\mathbf{B} = [b_{ij}]$ be a $k \times n$ zero-one matrix. Then the **Boolean product** $\mathbf{A} \odot \mathbf{B}$ is the $m \times n$ zero-one matrix $\mathbf{A} \odot \mathbf{B} = [c_{ij}]$, where

$$c_{ij} = (a_{i1} \wedge b_{1j}) \vee (a_{i2} \wedge b_{2j}) \vee \ldots \vee (a_{ik} \wedge b_{kj})$$

As an example, let

$$\mathbf{A} = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 1 \end{bmatrix} \quad \text{and} \quad \mathbf{B} = \begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}$$

Then

$$\mathbf{A} \odot \mathbf{B} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{bmatrix}$$