**MODULAR ARITHMETIC**

The Modulus

If *a*is an integer and *n*is a positive integer, we define *a*mod *n*to be the remainder when *a*is divided by *n*. The integer *n*is called the **modulus**. Thus, for any integer *a*, we can rewrite Equation (4.1) as  follows:

*a*= *qn*+ *r*0 <= *r*< *n*; *q*=   [*a*/*n*]

*a*=  [*a*/*n*] \* *n*+ (*a*mod *n*)

**11 mod 7 = 4;      - 11 mod 7 = 3**

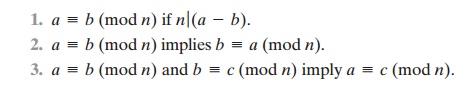
Two  integers *a*and  *b*are said to  be  **congruent modulo *n***, if  (*a*mod *n*) = (*b*mod *n*). This is written as *a*K *b*(mod *n*).2

**73 ‚ 4 (mod 23);        21 ‚ -9 (mod 10)**

Note that if *a*K 0 (mod *n*), then *n*| *a*.

Properties of Congruences

Congruences have the following properties:



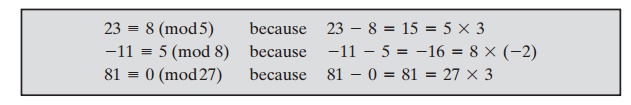
To demonstrate the first point, if *n*| (*a*- *b*), then (*a*- *b*)  = *kn*for some *k*.

So we can write *a*= *b*+ *kn*. Therefore, (*a*mod *n*) = (remainder when *b*+ *kn*is divided by *n*)  =  (remainder when *b*is divided by *n*)  =  (*b*mod *n*).

**23  = = 8 (mod 5)   because       23 - 8 = 15 = 5 \*  3**

**-11 = = 5 (mod 8)   because       -11 - 5 =  -16 = 8 \* (-2)**

**81  ==  0 (mod 27) because       81 - 0 = 81 = 27 \*  3**



The remaining points are as easily proved.

Modular Arithmetic Operations

Note that, by definition (Figure 4.1), the (mod *n*) operator maps all integers into the set of integers {0, 1, ... , (*n*- 1)}. This suggests the question: Can we perform arithmetic operations within the confines of this set? It turns out that we can; this technique is known as **modular arithmetic**.

Modular arithmetic exhibits the following properties:

***1.***                                                   [(*a*mod *n*)  + (*b*mod *n*)] mod *n*= (*a*+ *b*) mod *n*

***2.***                                                   [(*a*mod *n*) - (*b*mod *n*)] mod *n*= (*a*- *b*) mod *n*

***3.***                                                   [(*a*mod *n*) \* (*b*mod *n*)] mod *n*= (*a*\* *b*) mod *n*

We demonstrate the first property. Define (*a*mod *n*)  =  *ra*and (*b*mod *n*)  =  *rb*.

Then we can write *a*= *ra*+ *jn*for some integer *j*and *b*= *rb*+   *kn*for some integer

*k.*             Then

(*a*+ *b*) mod *n*= (*ra*+ *jn*+ *rb*+    *kn*) mod *n*

= (*ra*+ *rb*+ (*k*+ *j*)*n*) mod *n*

= (*ra*+ *rb*) mod  *n*

= [(*a*mod *n*)  + (*b*mod *n*)]mod *n*

The remaining properties are proven as easily. Here are examples of the three properties:

**11 mod 8  =  3; 15 mod 8  = 7**

**[(11 mod 8)  + (15 mod 8)] mod 8  = 10 mod 8  = 2**

**(11  + 15) mod 8  = 26 mod 8  = 2**

**[(11 mod 8) - (15 mod 8)] mod 8 = -4 mod 8 = 4**

**(11 - 15) mod 8 = -4 mod 8 = 4**

**[(11 mod 8) \* (15 mod 8)] mod 8 = 21 mod 8 = 5**

**(11 \* 15) mod 8 = 165 mod 8 =  5**

Exponentiation is performed by repeated multiplication, as in ordinary arith- metic. (We have more to say about exponentiation in Chapter 8.)

**To find 117 mod 13, we can proceed as follows:**

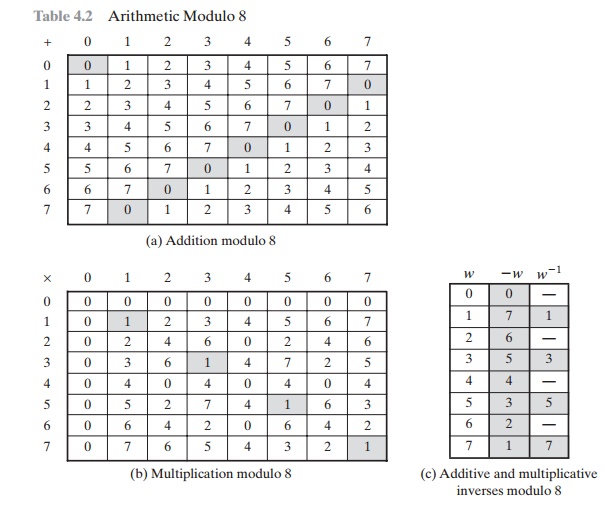
**112  =  121  K  4 (mod 13)**

**114  =  (112)2  K  42  K 3 (mod 13)**

**117  K 11  \* 4  \* 3  K 132  K 2 (mod 13)**

Thus, the rules for ordinary arithmetic involving addition, subtraction, and multiplication carry over into modular arithmetic.

Table 4.2 provides an illustration of modular addition and multiplication modulo 8. Looking at addition, the results are straightforward, and there is a regular pattern to the matrix. Both matrices are symmetric about the main diagonal in conformance to the commutative property of addition and multiplication. As in ordinary addition, there is an additive inverse, or negative, to each integer in modu- lar arithmetic. In this case, the negative of an integer *x*is the integer *y*such   that



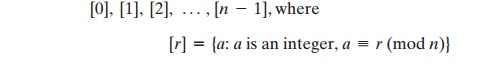
(*x*+ *y*) mod 8 = 0. To find the additive inverse of an integer in the left-hand column, scan across the corresponding row of the matrix to find the value 0; the integer at the top of that column is the additive inverse; thus, (2 + 6) mod 8 = 0. Similarly, the entries in the multiplication table are straightforward. In ordinary arithmetic, there is a multiplicative inverse, or reciprocal, to each integer. In modular arithmetic mod 8, the multiplicative inverse of *x*is the integer *y*such that (*x*\* *y*) mod 8 = 1 mod 8. Now, to find the multiplicative inverse of an integer from the multiplication table, scan across the matrix in the row for that integer to find the value 1; the integer at the top of that column is the multiplicative inverse; thus, (3 \* 3) mod 8 = 1. Note that not all integers mod 8 have a multiplicative inverse; more about that later.

Properties of Modular Arithmetic

Define the set Z*n*as the set of nonnegative integers less than *n*:



This is referred to as the **set of residues**, or **residue classes**(mod *n*). To be more precise, each integer in Z*n*represents a residue class. We can label the residue classes (mod *n*) as



The residue classes (mod 4) are

**[0] = { ... , -16, -12, -8, -4, 0, 4, 8, 12, 16, ... }**

**[1] = { ... , -15, -11, -7, -3, 1, 5, 9, 13, 17, ... }**

**[2] = { ... , -14, -10, -6, -2, 2, 6, 10, 14, 18, ... }**

**[3] = { ... , -13, -9, -5, -1, 3, 7, 11, 15, 19, ... }**

Of all the integers in a residue class, the smallest nonnegative integer is the one used to represent the residue class. Finding the smallest nonnegative integer to which *k*is congruent modulo *n*is called **reducing *k*modulo *n***.

If we perform modular arithmetic within Z*n*, the properties shown in Table 4.3 hold for integers in Z*n*. We show in the next section that this implies that Z*n*is a com- mutative ring with a multiplicative identity element.

There is one peculiarity of modular arithmetic that sets it apart from ordinary arithmetic. First, observe that (as in ordinary arithmetic) we can write the following:

**if**(*a*+ *b*)  K (*a*+ *c*) (mod *n*)   **then***b*K *c*(mod *n*)              **(4.4)**

**(5  + 23)  K (5  + 7) (mod 8);  23  K   7(mod 8)**

Equation (4.4) is consistent with the existence of an additive inverse. Adding the additive inverse of *a*to both sides of Equation (4.4), we have

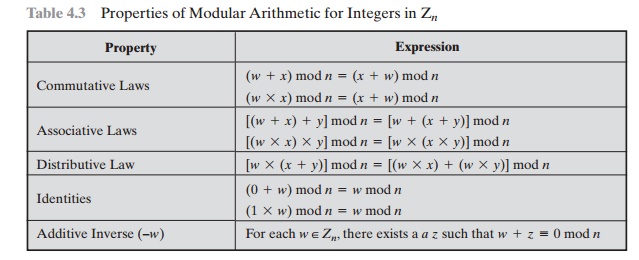
((-*a*)  +  *a*+  *b*)  K  ((-*a*)  +  *a*+  *c*) (mod *n*) *b*K *c*(mod *n*)

However, the following statement is true only with the attached condition:

**if**(*a*\* *b*)  K (*a*\* *c*) (mod *n*) **then***b*K *c*(mod *n*)   **if***a*is relatively prime to *n***(4.5)**

Recall that two integers are **relatively prime**if their only common positive integer factor is 1. Similar to the case of Equation (4.4), we can say that Equation (4.5) is

**Table 4.3**Properties of Modular Arithmetic for Integers in Z*n*



consistent with the existence of a multiplicative inverse. Applying the multiplicative inverse of *a*to both sides of Equation (4.5), we  have

((*a*- 1)*ab*) K ((*a*- 1)*ac*) (mod *n*) *b*K *c*(mod *n*)

**To see this, consider an example in which the condition of Equation (4.5) does not hold. The integers 6 and 8 are not relatively prime, since they have the common factor 2. We have the following:**

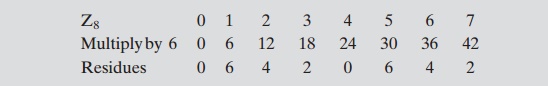
**6 \* 3 = 18 K 2 (mod 8)**

**6 \* 7 = 42 K 2 (mod 8)**

**Yet 3 [ 7 (mod 8).**

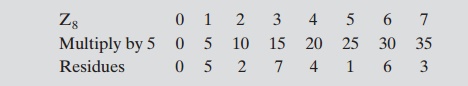
The reason for this strange result is that for any general modulus *n*, a multiplier *a*that is applied in turn to the integers 0 through (*n*- 1) will fail to produce a complete set of residues if *a*and *n*have any factors in common.

**With a  =  6 and n  = 8,**



**Because  we  do  not  have  a  complete  set  of  residues  when  multiplying  by 6, more  than  one  integer  in  Z8  maps  into  the  same  residue.  Specifically, 6 \* 0 mod 8 = 6 \* 4 mod 8; 6 \* 1 mod 8 = 6 \* 5 mod 8; and so on. Because this is a many-to-one mapping, there is not a unique inverse to the multiply operation. However, if we take a  =  5 and n  =     8, whose only common factor is 1,**

**However, if we take a  =  5 and n  =     8, whose only common factor is 1,**



**The line of residues contains all the integers in Z8, in a different order.**

In general, an integer has a multiplicative inverse in Z*n*if that integer is relatively prime to *n*. Table 4.2c shows that the integers 1, 3, 5, and 7 have a multiplicative inverse in Z8; but 2, 4, and 6 do not.

Euclidean Algorithm Revisited

The Euclidean algorithm can be based on the following theorem: For any nonnegative integer *a*and any positive integer *b*,

gcd(*a*, *b*)  =   gcd(*b*, *a*mod *b*)                **(4.6)**

**gcd(55, 22) =  gcd(22, 55 mod 22) =  gcd(22, 11) =   11**

To see that Equation (4.6) works, let  *d*=  gcd(*a*, *b*). Then, by the definition of  gcd, *d*|  *a*and *d*|       *b*. For any positive integer *b*, we can express *a*as

*a*= *kb*+ *r*K *r*(mod *b*)  *a*mod *b*= *r*

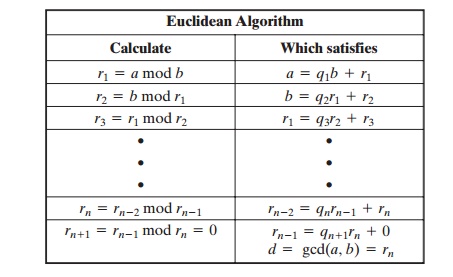
with *k*, *r*integers.Therefore, (*a*mod *b*) = *a*- *kb*for some integer *k*. But because *d*| *b*, it also divides *kb*.We also have *d*| *a*.Therefore, *d*| (*a*mod *b*).This shows that *d*is a common divisor of *b*and (*a*mod *b*). Conversely, if *d*is a common divisor of *b*and (*a*mod *b*), then *d*| *kb*and thus *d*| [*kb*+ (*a*mod *b*)], which is equivalent to *d*| *a*.Thus, the set of common divisors of *a*and *b*is equal to the set of common divisors of *b*and (*a*mod *b*).Therefore, the gcd of one pair is the same as the gcd of the other pair, proving the  theorem.

Equation (4.6) can be used repetitively to determine the greatest common divisor.

**gcd(18, 12)  = gcd(12, 6)  = gcd(6, 0)  = 6**

**gcd(11, 10)  = gcd(10, 1)  = gcd(1, 0)  = 1**

This is the same scheme shown in Equation (4.3), which can be rewritten in the following way.



We can define the Euclidean algorithm concisely as the following recursive function.

Euclid(a,b)

**if**(b=0) **then return**a;

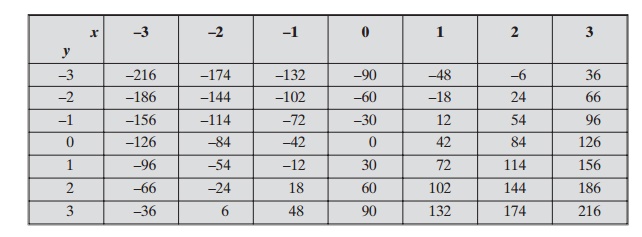
**else return**Euclid(b, a mod b);

The Extended Euclidean Algorithm

We now proceed to look at an extension to the Euclidean algorithm that will be impor- tant for later computations in the area of finite fields and in encryption algorithms, such as RSA. For given integers *a*and *b*, the extended Euclidean algorithm not only calculate the greatest common divisor *d*but also two additional integers *x*and *y*that satisfy the following equation.

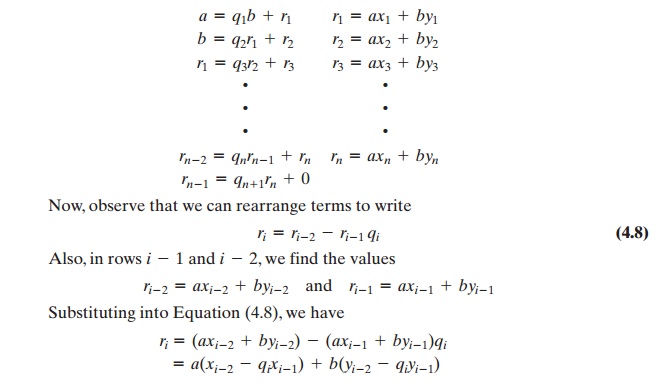
*ax*+ *by*= *d*=  gcd(*a*,  *b*)            **(4.7)**

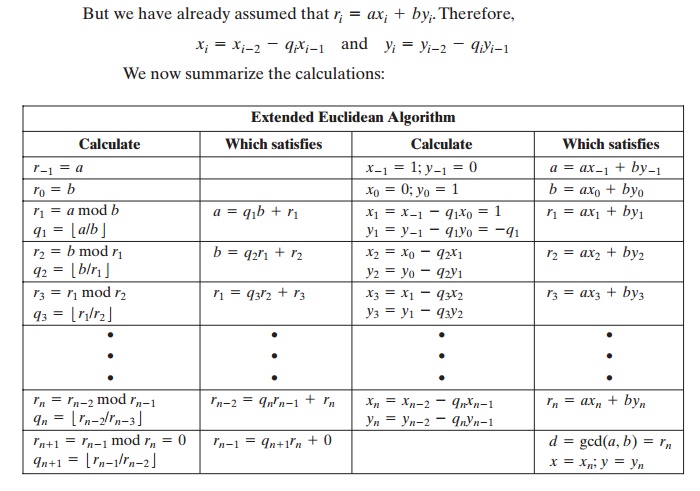
It should be clear that *x*and *y*will have opposite signs. Before examining the algorithm, let us look at some of the values of *x*and *y*when *a*= 42 and *b*=  30.  Note that gcd(42, 30)  =  6. Here is a partial table of values3 for 42*x*+ 30*y*.



Observer that all of the entries are divisible by 6. This is not surprising, because  both  42  and  30  are  divisible  by  6,  so  every  number  of  the    form 42*x*+ 30*y*= 6(7*x*+ 5*y*) is a multiple of 6. Note also that  gcd(42, 30)  = 6   appears in the table. In general, it can be shown that for given integers *a*and *b*, the smallest positive value of *ax*+  *by*is equal to gcd(*a*, *b*).

Now let us show how to extend the Euclidean algorithm to determine (*x*, *y*, *d*) given *a*and *b*. We again go through the sequence of divisions indicated in Equation (4.3), and we assume that at each step  *i*we can find integers  *xi*and  *yi*that satisfy   *ri*= *axi*+ *byi*. We end up with the following sequence.





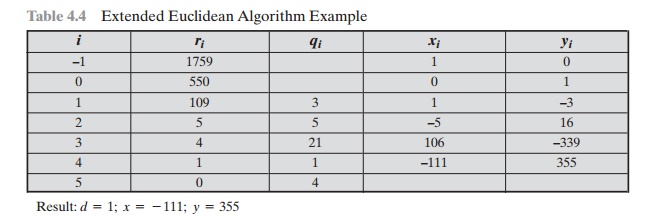
We need to make several additional comments here. In each row, we calculate a new remainder *ri*based on the remainders of the previous two rows, namely *ri*- 1 and *ri*- 2. To start the algorithm, we need values for *r*0 and *r*- 1, which are just *a*and *b*. It is then straightforward to determine the required values for *x*- 1, *y*- 1, *x*0, and *y*0.

We know from the original Euclidean algorithm that the process ends with a remainder  of  zero  and  that  the  greatest  common  divisor  of  a   and  b     is d = gcd(a, b) = rn. But we also have determined that Therefore, in Equation (4.7), x  =  xn and y  = yn.

d  = rn  = axn  + byn.

As   an   example,   let   us  use a  = 1759 and b  = 550 and   solve  for 1759x + 550y = gcd(1759, 550). The results are shown in Table 4.4. Thus, we have 1759 x (–111) + 550 x 355 = –195249 + 195250 = 1.

**Table 4.4**Extended Euclidean Algorithm Example



Result: *d*= 1; *x*=  - 111; *y*=   355

…………………………………………………………………

**Modular Arithmetic**

**Read**

Courses

Jobs

Modular arithmetic is the branch of arithmetic mathematics related with the “mod” functionality. Basically, modular arithmetic is related with computation of “mod” of expressions. Expressions may have digits and computational symbols of addition, subtraction, multiplication, division or any other. Here we will discuss briefly about all modular arithmetic operations.

[**Quotient Remainder Theorem:**](https://www.geeksforgeeks.org/quotient-remainder-theorem/)

It states that, for any pair of integers a and b (b is positive), there exist two unique integers q and r such that:

*a = b x q + r where 0 <= r < b*

**Example:** If a = 20, b = 6 then q = 3, r = 2 20 = 6 x 3 + 2

[**Modular Addition:**](https://www.geeksforgeeks.org/modular-addition/)

Rule for modular addition is:

*(a + b) mod m = ((a mod m) + (b mod m)) mod m*

**Example:**

(15 + 17) % 7

= ((15 % 7) + (17 % 7)) % 7

= (1 + 3) % 7

= 4 % 7

= 4

The same rule is to modular subtraction. We don’t require much modular subtraction but it can also be done in the same way.

[**Modular Multiplication:**](https://www.geeksforgeeks.org/modular-multiplication/)

The Rule for modular multiplication is:

*(a x b) mod m = ((a mod m) x (b mod m)) mod m*

**Example:**

(12 x 13) % 5

= ((12 % 5) x (13 % 5)) % 5

= (2 x 3) % 5

= 6 % 5

= 1

[**Modular Division:**](https://www.geeksforgeeks.org/modular-division/)

The modular division is totally different from modular addition, subtraction and multiplication. It also does not exist always.

(a / b) mod m is not equal to ((a mod m) / (b mod m)) mod m.

This is calculated using the following formula:

*(a / b) mod m = (a x (inverse of b if exists)) mod m*

[**Modular Inverse:**](https://www.geeksforgeeks.org/multiplicative-inverse-under-modulo-m/)

The modular inverse of a mod m exists only if a and m are relatively prime i.e. gcd(a, m) = 1. Hence, for finding the inverse of an under modulo m, if (a x b) mod m = 1 then b is the modular inverse of a.

**Example:** a = 5, m = 7 (5 x 3) % 7 = 1 hence, 3 is modulo inverse of 5 under 7.

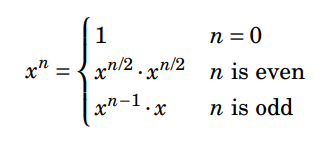
[**Modular Exponentiation:**](https://www.geeksforgeeks.org/modular-exponentiation-power-in-modular-arithmetic/)

Finding a^b mod m is the modular exponentiation. There are two approaches for this – recursive and iterative. **Example:**

a = 5, b = 2, m = 7

(5 ^ 2) % 7 = 25 % 7 = 4

There is often a need to efficiently calculate the value of xn mod m. This can be done in O(logn) time using the following recursion:



It is important that in the case of an even n, the value of xn/2 is calculated only once.

This guarantees that the time complexity of the algorithm is O(logn) because n is always halved when it is even.

The following function calculates the value of xn mod m:

…………………………………………………………………

Certainly! Let’s delve into the properties of modular arithmetic, particularly in the context of cryptography. Modular arithmetic plays a crucial role in cryptographic algorithms due to its reversible nature and the ability to handle large numbers efficiently. Here are some key properties:

1. **Modular Addition**:
   * Rule: ((a + b) \mod m = ((a \mod m) + (b \mod m)) \mod m)
   * Example: ((15 + 17) \mod 7 = ((15 \mod 7) + (17 \mod 7)) \mod 7 = (1 + 3) \mod 7 = 4 \mod 7 = 4)
2. **Modular Subtraction**:
   * Similar to addition, we apply the same rule for modular subtraction.
3. **Modular Multiplication**:
   * Rule: ((a \cdot b) \mod m = ((a \mod m) \cdot (b \mod m)) \mod m)
   * Example: ((12 \cdot 13) \mod 5 = ((12 \mod 5) \cdot (13 \mod 5)) \mod 5 = (2 \cdot 3) \mod 5 = 6 \mod 5 = 1)
4. **Modular Division**:
   * Unlike addition, subtraction, and multiplication, modular division is different and not always defined.
   * The expression (\frac{a}{b} \mod m) is not equal to (\left(\frac{a \mod m}{b \mod m}\right) \mod m).
   * Instead, it is calculated using: (\frac{a}{b} \mod m = (a \cdot \text{{inverse of }} b) \mod m)
5. **Modular Inverse**:
   * The modular inverse of (a \mod m) exists only if (a) and (m) are relatively prime (i.e., (\gcd(a, m) = 1)).
   * If ((a \cdot b) \mod m = 1), then (b) is the modular inverse of (a).
   * Example: For (a = 5) and (m = 7), ((5 \cdot 3) \mod 7 = 1), so (3) is the modular inverse of (5) under (7).
6. **Modular Exponentiation**:
   * Finding (a^b \mod m) is essential in cryptography.
   * Recursive or iterative approaches can be used.
   * Example: For (a = 5), (b = 2), and (m = 7), ((5^2) \mod 7 = 25 \mod 7 = 4).
   * Efficient calculation using recursion: (x^n \mod m) can be done in (O(\log n)) time.

Remember that Fermat’s theorem states that (x^{m-1} \mod m = 1) when (m) is prime and (x) and (m) are coprime. This property also yields (x^k \mod m = x^{k \mod (m-1)} \mod m).

These properties form the foundation for secure cryptographic algorithms, allowing encryption, decryption, and digital signatures.

…………………………………………………………………

Certainly! Let's go through the properties of modular arithmetic binary operations, namely addition, subtraction, multiplication, and exponentiation. We will state each property and provide a brief proof.

### Addition:

1. \*\*Closure Property:\*\*

For all integers \(a\) and \(b\), if \(a \equiv c \pmod{m}\) and \(b \equiv d \pmod{m}\), then \((a + b) \equiv (c + d) \pmod{m}\).

\*Proof:\* Let \(a = c + km\) and \(b = d + jm\) for some integers \(k\) and \(j\). Then, \(a + b = (c + d) + (k + j)m\). Since \(k + j\) is an integer, \(a + b \equiv (c + d) \pmod{m}\).

2. \*\*Associative Property:\*\*

For all integers \(a\), \(b\), and \(c\), \((a + b) + c \equiv a + (b + c) \pmod{m}\).

\*Proof:\* This property holds in the integers, and since modular arithmetic involves working with integers, the associative property also holds for modular addition.

3. \*\*Commutative Property:\*\*

For all integers \(a\) and \(b\), \(a + b \equiv b + a \pmod{m}\).

\*Proof:\* This property holds in the integers, and since modular arithmetic involves working with integers, the commutative property also holds for modular addition.

### Multiplication:

1. \*\*Closure Property:\*\*

For all integers \(a\) and \(b\), if \(a \equiv c \pmod{m}\) and \(b \equiv d \pmod{m}\), then \((a \cdot b) \equiv (c \cdot d) \pmod{m}\).

\*Proof:\* Let \(a = c + km\) and \(b = d + jm\) for some integers \(k\) and \(j\). Then, \(a \cdot b = (c + km) \cdot (d + jm)\). Expanding and rearranging terms, \(a \cdot b \equiv c \cdot d \pmod{m}\).

2. \*\*Associative Property:\*\*

For all integers \(a\), \(b\), and \(c\), \((a \cdot b) \cdot c \equiv a \cdot (b \cdot c) \pmod{m}\).

\*Proof:\* This property holds in the integers, and since modular arithmetic involves working with integers, the associative property also holds for modular multiplication.

3. \*\*Commutative Property:\*\*

For all integers \(a\) and \(b\), \(a \cdot b \equiv b \cdot a \pmod{m}\).

\*Proof:\* This property holds in the integers, and since modular arithmetic involves working with integers, the commutative property also holds for modular multiplication.

### Exponentiation:

1. \*\*Power Rule:\*\*

For all integers \(a\), \(b\), and \(c\), \(a^b \equiv a^{b \pmod{\phi(m)}} \pmod{m}\), where \(\phi(m)\) is Euler's totient function.

\*Proof:\* This follows from Euler's totient theorem, which states that \(a^{\phi(m)} \equiv 1 \pmod{m}\) for \(a\) coprime to \(m\). If \(b \equiv c \pmod{\phi(m)}\), then \(b = c + k\phi(m)\) for some integer \(k\). Thus, \(a^b = a^{c + k\phi(m)} = a^c \cdot (a^{\phi(m)})^k \equiv a^c \pmod{m}\).

These properties are fundamental in modular arithmetic and are crucial in various mathematical and cryptographic applications. They provide a foundation for manipulating numbers within a modular system, ensuring consistency and predictability in calculations.