

Module 11: Introduction to Number Theory and Cryptography

Introduction

The Internet is in ever-increasing use for a variety of applications that involve sensitive data and demand secure communication. In this module, we study some of the basic number theoretic results and then apply it to introduce cryptography.

Contents

1. Prime Numbers
2. Greatest Common Divisor (GCD)
3. Euclidean GCD Algorithm
4. Modulo Arithmetic
5. Inverse of Integers Modulo n
6. Exponentiation Modulo n
7. Number-Theoretic Background
 - (a) Fermat's Little Theorem
 - (b) Euler's Theorem
 - (c) A Generalization of Euler's Theorem
8. The RSA Public-Key Cryptosystem

1. Prime Numbers

Definitions:

- Given positive integers a and b , we say a **divides** b , or a is a **divisor** of b , and denote it as

$$a|b$$

if there is a positive integer q such that $b = qa$. We also say b is **divisible** by a .

- Given positive integers a, b , and d . If $d|a$ and $d|b$, we say d is a **common divisor** of a and b .


Theorem 1: If $d|a$ and $d|b$, then for any integers s and t ,

$$d | (sa + tb)$$

We call $(sa + tb)$ a **linear sum**, or **linear combination**, of a and b .

Proof: Since $d|a$ and $d|b$, then $a = pd$ and $b = qd$ for some integers p and q . So

$$sa + tb = spd + tqd = (sp + tq)d$$

Since $(sp + tq)$ is an integer, this means $d | (sa + tb)$. 

Definition: A positive integer greater than 1 is **prime** if it is divisible only by 1 and itself.

For example, the first few prime numbers are: 2, 3, 5, 7, 11, 13, 17, 19, 23, and 29.

An integer greater than 1 that is not prime is called **composite**. For example, 514250 is a composite number and it may be written as product of its prime factors.

$$514250 = 2 * 5^3 * 11^2 * 17$$

A positive integer $n > 1$ has a **unique prime factorization**

$$n = p_1^{t_1} p_2^{t_2} \cdots p_k^{t_k}$$

for some integer k , where $p_1 < p_2 < \cdots < p_k$ are prime factors (divisors) and t_1, t_2, \dots, t_k are positive integer exponents.

If integer n is not too large, then it is fairly simple to find its prime factors.

Lemma 1: A composite integer $n > 1$, has a prime factor $p \leq \sqrt{n}$.

Proof: Suppose to the contrary, the smallest prime factor is $p > \sqrt{n}$. Then, $n = pq$, for some integer $q < \sqrt{n}$. This means n has a prime factor $< \sqrt{n}$, which is a contradiction. Therefore, the smallest prime factor is $p \leq \sqrt{n}$. ■

The following algorithm determines if an integer $n > 1$ is composite, and if so finds the smallest prime factor of n .

```
int Factor (int n) {
  for p = 1 to  $\sqrt{n}$ 
    if (n mod p == 0) return p;    \ prime factor
  return 0 }                     \ n is prime
```

Note that the first integer found by the algorithm is the smallest prime factor. (If p is composite, the algorithm would find its prime factors before it gets to p .)

The time complexity of the above algorithm is $O(\sqrt{n})$. The algorithm is efficient if n is not very large. But if n is very large, then it becomes very difficult to find the factorization.

Later, we study a cryptosystem known as RSA. This system is based on an extremely large composite n , relying on the fact that it would be nearly impossible to “break the code” by decomposing n . Suppose n is about 100 decimal digits. Then,

$$\sqrt{n} \cong 10^{50}$$

To appreciate how large this exponential time becomes, suppose an iteration of the loop in the above algorithm takes 10 nanoseconds. Then, the running time of the algorithm is 10^{42} seconds. The number of seconds in one year is $365 * 24 * 3600 \cong 3.15 * 10^7$. So the running time becomes about $3 * 10^{34}$ years! In other words, exponential time means forever!

2. Greatest Common Divisors (GCD)

The **greatest common divisor** of two positive integers a and b , denoted as $\gcd(a, b)$, is the largest number that divides both of them. Note that $\gcd(a, 0) = a$. Integers a and b are said to be **relatively prime** if they have no common divisor > 1 , so $\gcd(a, b) = 1$.

If the prime factorization is known, then finding GCD is trivial. For example, suppose

$$\begin{aligned}a &= 6825 = 3 * 5^2 * 7 * 13 \\b &= 90 = 2 * 3^2 * 5\end{aligned}$$

Then, $\gcd(a, b) = 15 = 3 * 5$. For very large numbers, however, finding prime factorization is not a feasible approach, and we will see an alternative method for finding the GCD, known as Euclid's algorithm.

The GCD is useful for reducing a ratio to an equivalent form so that the numerator and denominator become relatively prime. For example, for the above numbers

$$\frac{b}{a} = \frac{90}{6825} = \frac{90/15}{6825/15} = \frac{6}{455}$$

The GCD plays a very important role in finding the inverse of an integer and in RSA cryptosystem.

Next, we study an ancient efficient algorithm, Euclidean Algorithm, for computing the GCD and finding the integers s and t so that $\gcd(a, b) = sa + tb$.

3. Euclidean GCD Algorithm

This ancient algorithm is based on the following observation.

Lemma 2: Let a and b be two positive integers, where $a > b$. Then,

$$\gcd(a, b) = \gcd(b, a \bmod b).$$

Proof:

(1) From Theorem 1, we know that a common divisor of (a, b) also divides

$$r = a \bmod b = a - qb.$$

So, every common divisor of (a, b) is also a common divisor of (b, r) .

(2) The converse is also true. That is, $a = r + qb$. So, a is a linear sum of b and r . Thus, by Theorem 1, every common divisor of (b, r) is a common divisor of (a, b) .

Therefore, $\gcd(a, b) = \gcd(b, r)$. ■

Euclid's algorithm applies this rule repeatedly to reduce the size of the numbers until the smaller number becomes 0. An iterative version of the algorithm follows.

```
int GCD(int a, int b) { // Assume  $a > b \geq 0$ 
  while ( $b > 0$ ) {
     $r = a \bmod b$ ;
     $a = b$ ;  $b = r$ ;
  }
  return  $a$  // When  $b = 0$ ;  $\gcd(a, 0) = a$ ;
}
```

Below is a numerical illustration of the algorithm, which finds $\gcd(7650, 285) = 15$.

Table 1: Illustration of Euclid's algorithm

Iteration	a	b	$r = a \bmod b$
1	7650	285	240
2	285	240	45
3	240	45	15
4	45	15	0
5	15	0	

Theorem 2: Let a and b be positive integers, $a > b$. There exist integers s and t so that

$$\gcd(a, b) = sa + tb$$

Furthermore, $\gcd(a, b)$ is the **smallest** positive integer which is a linear combination of a and b .

Proof: Since each iteration of the algorithm performs a mod operation, which is a linear combination of (a, b) , the final GCD is also a linear combination of the inputs (a, b) .

Next, we prove that $\gcd(a, b)$ is the smallest positive linear combination of (a, b) .

Suppose to the contrary, there is a smaller integer d' ,

$$0 < d' < \gcd(a, b)$$

where $d' = s'a + t'b$ for some integers s', t' . Since d' is a linear combination of (a, b) , by Theorem1, $\gcd(a, b)$ divides d' . Thus, $\gcd(a, b) \leq d'$, which is a contradiction. ■

It is easy to modify the iterative Euclid's algorithm to compute the integers (s, t) , so that $\gcd(a, b) = sa + tb$. Let (a, b) be the value of the inputs at the beginning. Let a_i, b_i denote the variables at the start of iteration i , and let $q_i = \lfloor a_i/b_i \rfloor$ and $r_i = a_i - q_i b_i = a_i \bmod b_i$. The algorithm will maintain two variables (s^a, t^a) associated with a_i , which show the value of a_i in terms of the original inputs (a, b) . That is, $a_i = s^a * a + t^a * b$. Similarly, the algorithm maintains variables (s^b, t^b) associated with b_i , and variables (s^r, t^r) associated with r_i . Then, when iteration i performs the mod operation, it updates the variables r_i and (s^r, t^r) accordingly. That is,

$$(s^r, t^r) = (s^a, t^a) - q * (s^b, t^b)$$

Below is an illustration of this implementation for the same pair of inputs a, b . (The pseudo-code is left to the student as an exercise.)

Table 2: Computation of integers s, t by the iterative Euclid's algorithm

Iteration	a	b	$r = a \bmod b$
1	7650 (1,0)	285 (0,1)	$240 = a_1 - 26b_1$ (1, -26)
2	285 (0,1)	240 (1, -26)	$45 = a_2 - b_2$ (-1, 27)
3	240 (1, -26)	45 (-1, 27)	$15 = a_3 - 5b_3$ (6, -161)
4	45 (-1, 27)	15 (6, -161)	0

The algorithm returns $\gcd = 6$, $s = 6, t = -161$. That is,

$$\gcd(7650, 285) = 15 = 6 * 7650 - 161 * 285.$$

Next, we present a recursive version of the GCD algorithm, which also computes the integers (s, t) .

```

int RGCD(int a, int b, int s, int t)
{ // Inputs: a, b. It is assumed that a > b ≥ 0;
  // Outputs: Returned parameters s, t; and GCD returned as the value of the
    function,
  if (b == 0) { s = 1; t = 0; return a };
  q = ⌊a/b⌋;
  G = RGCD(b, a - qb, s2, t2);
  // Now GCD = s2b + t2(a - qb) = t2a + (s2 - t2q) b
  s = t2; t = s2 - t2q;
  return (G); }

```

Analysis of Euclid's Algorithm

Lemma 3: Let a be the larger of the two inputs to GCD algorithm. The number of iterations (thus the number of mod operations) is at most $2 \log_2 a$.

Proof: We show that a will reduce by at least a factor of 2 after one or two iterations. Let a_i, b_i denote the value of the variables at the beginning of iteration i of the algorithm, and let r_i be the mod value computed during this iteration. There are two cases:

1. $b_i \leq a_i/2$: Then after one iteration, $a_{i+1} = b_i \leq a_i/2$.
2. $b_i > a_i/2$: Then $r_i = a_i - b_i < a_i/2$. So, $a_{i+2} = b_{i+1} = r_i < a_i/2$.

So the size of the larger input a is reduced by at least a factor of 2 after at most two iterations. Therefore, the number of iterations is at most $2 \log_2 a$.

Finally, we may also express the running time in terms of the smaller input b . Since after one iteration, $a_{i+1} = b_i$, then the number of iterations is at most $1 + 2 \log_2 b$. ■

Next, we present an interesting alternative analysis in terms of Fibonacci sequence, which is defined as $F_1 = F_2 = 1$ and $F_n = F_{n-1} + F_{n-2}$, $n \geq 3$. This alternative analysis improves the worst-case number of iterations to $1.44 \log_2 a$.

Lemma 4: If the larger input to GCD algorithm is $a < F_n$ for some integer n , then the number of mod operations of the algorithm is at most $n - 3$.

Proof: This is proved by (strong) induction on n . For base cases, we use $n = 3, n = 4$.

- For $n = 3$, the larger input is $a < F_3 = 2$, so $a \leq 1$ and clearly no mod operation is needed, which is $n - 3 = 0$.
- For $n = 4$, the larger input is $a < F_4 = 3$, so $a \leq 2$. In this case, at most one mod operation is needed, which is $n - 3 = 1$. So the two base cases are correct.

Now suppose $a < F_n$ for some $n \geq 5$. Let a_i, b_i denote the variables at the beginning of iteration i of the algorithm, and let r_i be the mod computed in this iteration. Initially, $a_1 < F_n$. We consider two cases:

1. If $b_1 < F_{n-1}$: Then, after one iteration, the larger number is $a_2 = b_1 < F_{n-1}$.
2. If $b_1 \geq F_{n-1}$: Then, $r_1 = a_1 - b_1 < F_n - F_{n-1} = F_{n-2}$. And r_1 becomes the larger input after two iterations. So, $a_3 = b_2 = r_1 < F_{n-2}$.

In simple words, if initially the larger input is $< F_n$, then

- after one iteration the larger input will be $< F_{n-1}$, or
- after two iterations the larger input will be $< F_{n-2}$.

Therefore, by induction, the total number of mod operations is at most $n - 3$. ■

Corollary 1: Let a be the larger input to GCD algorithm. The number of mod operations is at most $(1.4405) \log_2 a$.

Proof: Suppose the larger input a is $F_{n-1} \leq a < F_n$ for some integer n . The above lemma established the number of mod operations is at most $n - 3$. And, it is easy to prove by induction that

$$F_n \geq (1.618)^{n-2}, \quad \forall n \geq 2$$

(This induction proof is left to the reader.) Therefore, $a \geq F_{n-1} \geq (1.618)^{n-3}$. Taking the log of both sides yields the worst-case number of mod operations:

$$n - 3 \leq \log_{1.618} a = (\log_2 a)(\log_{1.618} 2) < (1.4405) \log_2 a. \quad \blacksquare$$

It is interesting to note that if initially $a = F_n$ and $b = F_{n-1}$ (the exact equality) then the number of iterations will be exactly $n - 2$. Below is an example for $a = F_8$ and $b = F_7$. Note that iteration 6 is the last mod operation.

Table 3: The working of Euclid's algorithm with inputs $a = F_8, b = F_7$.

Iteration	a	b	$r = a \bmod b$
1	$F_8 = 21$	$F_7 = 13$	$F_6 = 8$
2	$F_7 = 13$	8	5
3	$F_6 = 8$	5	3
4	$F_5 = 5$	3	2
5	$F_4 = 3$	2	1
6	$F_3 = 2$	1	0
	1	0	

4. Modulo Arithmetic

The following lemma is useful for handling a series of multiplications, modulo n .

Lemma 5: Let x, y, z , and n be positive integers.

- (1) $(x * y) \bmod n = ((x \bmod n) * (y \bmod n)) \bmod n$
- (2) $(x * y * z) \bmod n = ((x * y) \bmod n) * z \bmod n$

Proof: We prove only (1), since (2) basically follows from (1). Let $x \bmod n = d_1$ and $y \bmod n = d_2$. This means $x = d_1 + r_1n$, and $y = d_2 + r_2n$, for some integers r_1, r_2 . Then,

$$\begin{aligned} (x * y) \bmod n &= (d_1 + r_1n)(d_2 + r_2n) \bmod n \\ &= ((d_1d_2) + (d_1r_2 + r_1d_2)n + r_1r_2n^2) \bmod n = (d_1d_2) \bmod n. \end{aligned}$$

\blacksquare

Suppose we want to compute a series of multiplications $(x_1 * x_2 * x_3 * \cdots * x_p) \bmod n$.

If we first perform the entire series of multiplications and then perform the mod operation at the end, the intermediate results may get too large. Instead, we perform a mod operation after each multiplication, to prevent the intermediate results from becoming too large: $x_1 * x_2 \bmod n * x_3 \bmod n * \cdots * x_p \bmod n$

(The above lemma provides the justification for reordering the mod operations.)

5. Inverse of Integers Modulo n

Definition: Let x and n be two positive integers. The **inverse** of x modulo n , if it exists, is a positive integer $s < n$ such that

$$x * s \bmod n = 1.$$

The inverse of x is denoted as x^{-1} . ■

For example, inverse of 5 mod 13 is 8, since $5 * 8 \bmod 13 = 40 \bmod 13 = 1$. The inverse does not always exist. For example, inverse of 6 mod 15 does not exist. (This may be easily verified by testing all integers 1 to 14.) The following theorem proves the necessary and sufficient condition for the inverse to exist.

Theorem 3: Let x and n be two positive integers. The inverse of x modulo n exists, and is unique, if and only if $\gcd(x, n) = 1$.

Proof:

1. Suppose $\gcd(x, n) = 1$. By Theorem 2, there are integers s, t such that

$$sx + tn = 1$$

This means $sx \bmod n = 1$, which means $(s \bmod n) x \bmod n = 1$. Therefore,

$$x^{-1} = s \bmod n.$$

2. Suppose the inverse of $x \bmod n$ exists. Let s be the inverse. Then, $sx \bmod n = 1$.

Thus, there is an integer t such that $sx + tn = 1$. By Theorem 2,

$$\gcd(x, n) = sx + tn = 1.$$

The uniqueness may be established by a simple proof by contradiction. Suppose an integer x has inverses s and s' , where $s < s' < n$. Then,

$$xs \bmod n = xs' \bmod n = 1,$$

$$x(s' - s) = rn, \text{ for some integer } r.$$

Since x is prime relative to n , and $s' - s < n$, the latter is true only if $r = 0, s' = s$. ■

Computation of Inverse using GCD

The above theorem suggests how to compute the inverse efficiently by computing GCD. To compute the inverse of an integer $x \bmod n$, we do:

Find $\gcd(x, n)$, and integers (s, t) so that $\gcd(x, n) = sx + tn$.
 If $\gcd(x, n) = 1$, then $x^{-1} = s \bmod n$;
 Else, the inverse does not exist.

The table below shows the inverse of integers modulo 13. Since 13 is prime, all integers 1 to 12 are prime relative to 13 and thus have inverse. For example, to compute inverse of 6, we use GCD to find $\gcd(13, 6) = 1 = 1 * 13 - 2 * 6$. Thus, $6^{-1} = -2 \bmod 13 = 11$.

i	1	2	3	4	5	6	7	8	9	10	11	12
$i^{-1} \bmod 13$	1	7	9	10	8	11	2	5	3	4	6	12

The next table shows the inverse of integers 1 to 9 modulo 10. The only integers which are prime relative to 10 are $\{1, 3, 7, 9\}$ and thus have inverse. Integers $\{2, 4, 5, 6, 8\}$ are not prime relative to 10 and so they do not have inverse.

i	1	2	3	4	5	6	7	8	9
$\gcd(10, i)$	1	2	1	2	5	2	1	2	1
$i^{-1} \bmod 10$	1	-	7	-	-	-	3	-	9

The inverse plays a critical role in secure communications. In such a communication, the sender of a message encodes the message by applying some transformation so that an unauthorized third party will not be able to read it. Then, the receiver applies the inverse transformation to recover the original message.

A Naïve Communication Protocol

Consider the following naïve scheme for secure communications. Suppose a sender and a receiver privately agree on a secret pair of integers (s, n) , where $\gcd(s, n) = 1$. The receiver computes the integer $s^{-1} \bmod n$, and saves it in a secure place. Then:

1. The sender, who wants to transmit a message as an integer $M < n$, *encodes*, or *encrypts*, the message by computing $E = (M * s) \bmod n$, and transmits E .

2. The receiver *decodes* the received data by computing $(E * s^{-1}) \bmod n$. This produces the original message because

$$(M * s \bmod n) * s^{-1} \bmod n = (M * s * s^{-1}) \bmod n = M.$$

However, this scheme has some problems. For example, how do the sender and receiver communicate the pair of integers (s, n) in a secure way?! Later, we will see a more sophisticated and secure communication scheme which uses the idea of inverse but in conjunction with exponentiation modulo n .

6. Exponentiation Modulo n

We now describe an efficient algorithm to compute exponentiation modulo n . Suppose we have integers x, p , and n , and we want to compute $x^p \bmod n$. In an earlier module, we discussed how to perform exponentiation by repeated squaring. Here, we extend the algorithm to add the mod computation in a straightforward way.

```

Int Power (int x, int p, int n) { // Compute  $x^p \bmod n$ . Assume  $p \geq 1$ .
  if ( $p == 1$ ) return ( $x \bmod n$ );
   $T = \text{Power}(x, \lfloor p/2 \rfloor, n)$ ;
   $T = (T * T) \bmod n$ ;
  if ( $p \bmod 2 == 1$ )
     $T = (T * x) \bmod n$ ;
  return  $T$ ; }

```

To analyze the running time of the algorithm, let $f(p)$ be the worst-case number of multiplications. Then,

$$f(p) = \begin{cases} 0, & p = 1 \\ f\left(\left\lfloor \frac{p}{2} \right\rfloor\right) + 2, & p > 1 \end{cases}$$

It is easy to prove by induction that the solution is $f(p) = 2\lfloor \log p \rfloor$.

Example: Below is computation of $x^p \bmod n$, for $x = 79$, $p = 5143$, $n = 4927$.

Power p	Square	Mod n	$* x$	Mod n
5143	2,560,000	2887	228,073	1431
2571	5,121,169	2016	159,264	1600
1285	22,401,289	3147	248,613	2263
642	10,316,944	4733	4,733	4733
321	6,105,841	1288	101,752	3212
160	9,339,136	2471	2,471	2471
80	2,047,761	3056	3,056	3056
40	10,909,809	1431	1,431	1431
20	19,430,464	3303	3,303	3303
10	4,064,256	4408	4,408	4408
5	1,726,596	2146	169,534	2016
2	6,241	1314	1,314	1314
1				79

7. Number-Theoretic Background

Let $Z_n = \{0, 1, \dots, n-1\}$.

Lemma 6: Given positive integers x and n , where $\gcd(x, n) = 1$. The sequence

$$(x * i \bmod n \mid i = 1, 2, \dots, n-1)$$

is a permutation of $(1, 2, \dots, n-1)$.

Proof: We show that for any pair of positive integers i and j , where $1 \leq i < j \leq n-1$,

$$x * i \bmod n \neq x * j \bmod n$$

That is, for any integer r ,

$$x(j-i) \neq rn.$$

The latter is true since x is prime relative to n , and $0 < (j-i) < n$. ■

Below is an example for $n = 8$, $x = 3$.

i	1	2	3	4	5	6	7
$3i \bmod 8$	3	6	1	4	7	2	5

A Word of Caution About Modulo Arithmetic: Consider the following equality, where x and i are two positive integers in Z_n . Can we cancel the factor i from both sides and conclude $x = 1$?

$$x * i \bmod n = i$$

Not always! In the above example, $x = 3$, $i = 4$, $n = 8$. Here, $x \neq 1$, but

$$x * 4 \bmod 8 = 4$$

(General rule: Multiplying both sides by a term mod n is valid. Dividing both sides by a term is not valid.)

However, if integer i has an inverse mod n , then we can multiply both sides by the inverse i^{-1} , and as a result cancel i from both sides:

$$\begin{aligned} x * i \bmod n &= i \\ ((x * i \bmod n) * i^{-1}) \bmod n &= (i * i^{-1}) \bmod n \\ (x * i * i^{-1}) \bmod n &= 1 \\ x &= 1 \end{aligned}$$

We are now ready for our important theorem.

Theorem 4 (Fermat's Little Theorem): Let p be a prime integer, and let x be a positive integer with $\gcd(x, p) = 1$. (Since p is prime, this condition means $x \neq rp$ for any integer r .) Then,

$$x^{p-1} \bmod p = 1$$

Proof: Let us form the following product of $p - 1$ terms.

$$F = (x * 1 \bmod p)(x * 2 \bmod p) \cdots (x * (p - 1) \bmod p) \bmod p$$

From Lemma 6, we know the sequence $(x * i \bmod p \mid i = 1, 2, \dots, p - 1)$ is a permutation of $(1, 2, \dots, p - 1)$. Therefore, the product of the terms in the sequence results in $(p - 1)!$. That is,

$$F = (p - 1)! \bmod p$$

But we may also rearrange the product terms to separate all x factors, and thus produce

$$F = x^{p-1} * (\prod_{i=1:p-1} i) \bmod p = x^{p-1}(p - 1)! \bmod p$$

Therefore,

$$x^{p-1}(p - 1)! \bmod p = (p - 1)! \bmod p$$

Since p is prime, all integers $(1, 2, \dots, p-1)$ have inverse mod p . So we may multiply both sides by $(p-1)^{-1}(p-2)^{-1} \dots 1^{-1}$. This will result in cancelling the term $(p-1)!$ from both sides. Therefore,

$$x^{p-1} \bmod p = 1$$



The following table illustrates the computation of $x^i \bmod p$, $1 \leq i \leq p-1$, for $x = 3, p = 7$. Observe that $3^6 \bmod 7 = 1$.

i	1	2	3	4	5	6
$3^i \bmod 7$	3	2	6	4	5	1

Next we generalize the above theorem to the case when p may not be prime.

Generalization of Fermat's Little Theorem (Euler's Theorem)

Definitions: Let $\phi(n)$ denote the number of integers in $Z_n = \{0, 1, \dots, n-1\}$ which are prime relative to n . And let $Z_n^* = \{u_1, u_2, \dots, u_{\phi(n)}\}$ be the set of elements in Z_n which are prime relative to n . ■

For the case when n is product of two primes p and q , $n = pq$, it is easy to compute $\phi(n)$. First, we count the integers in Z_n which are *not* prime relative to n . There are q integers in Z_n which are a multiple of p , namely $\{0, p, 2p, \dots, (q-1)p\}$. Similarly, there are p integers which are a multiple of q . And integer 0 is common to both, so there are $p + q - 1$ integers not prime relative to n . Since $n = pq$, then

$$\phi(n) = pq - p - q + 1 = (p-1)(q-1)$$

Example: For $Z_{15} = \{0, 1, \dots, 14\}$, $n = 15 = 3 * 5$, and $\phi(n) = (3-1)(5-1) = 8$. There are 8 integers in Z_{15} which are prime relative to $n = 15$. (The integers are highlighted in yellow below.) Thus, $Z_{15}^* = \{1, 2, 4, 7, 8, 11, 13, 14\}$. ■

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
---	---	---	---	---	---	---	---	---	---	----	----	----	----	----

The set of elements in Z_n^* satisfy two important properties, proved below.

Lemma 7:

1. Every integer $i \in Z_n^*$ has an inverse mod n , and the inverse is in Z_n^* .
2. For every pair of integers i and j in Z_n^* , their product $(i * j \bmod n)$ is also in Z_n^* .
(This property is called *closure under multiplication*.)

Proof:

1. Every integer $i \in Z_n^*$ has an inverse because $\gcd(i, n) = 1$. And i^{-1} has an inverse. That is, $(i^{-1})^{-1} = i$. And this requires $\gcd(i^{-1}, n) = 1$, which means $i^{-1} \in Z_n^*$. For example, the inverse of all integers in Z_{15}^* are shown below.

$i \in Z_{15}^*$	1	2	4	7	8	11	13	14
$i^{-1} \bmod 15$	1	8	4	13	2	11	7	14

2. For every pair of integers i and j in Z_n^* , their product $(i * j \bmod n)$ is also in Z_n^* . The reason is obvious: Since i and j are both prime relative to n , their product $(i * j)$ is also prime relative to n , and therefore $(i * j \bmod n)$ is prime relative to n . For example, for integers 4 and 11 in Z_{15}^* , their product is $(4 * 11 \bmod 15 = 14)$, which is in Z_{15}^* . ■

Next, let us state a generalization of Lemma 6.

Lemma 8: Let $Z_n^* = \{u_1, u_2, \dots, u_{\phi(n)}\}$. Let x be any integer in Z_n^* . The sequence

$$(x * u_i \bmod n \mid i = 1, 2, \dots, \phi(n))$$

is a permutation of $(u_1, u_2, \dots, u_{\phi(n)})$.

Proof: The proof is very similar to the proof of Lemma 6. We need to show that for any pair u_i and u_j , where $u_i < u_j$,

$$x * u_i \bmod n \neq x * u_j \bmod n$$

That is, for any integer r ,

$$x(u_j - u_i) \neq rn$$

This is true since x and n are relatively prime, and $0 < u_j - u_i < n$. ■

Below is an illustration for Z_{15}^* , and $x = 7$.

$u_i \in Z_{15}^*$	1	2	4	7	8	11	13	14
$7u_i \bmod 15$	7	14	13	4	11	2	1	8

We are now ready for a generalization of Fermat's Little Theorem.

Theorem 5 (Euler's Theorem): Let n be a positive integer, and $Z_n^* = \{u_1, u_2, \dots, u_{\phi(n)}\}$ be the set of elements in Z_n^* which are prime relative to n . And let x be an integer in Z_n^* . Then,

$$x^{\phi(n)} \bmod n = 1$$

Proof: The proof is similar to the proof of Fermat's Little Theorem. Let's form the following product.

$$F = (\prod_{i=1:\phi(n)} (xu_i \bmod n)) \bmod n$$

On one hand, from Lemma 8, we know the sequence $(xu_i \bmod n \mid i = 1, 2, \dots, \phi(n))$ is a permutation of $(u_1, u_2, \dots, u_{\phi(n)})$. So, the product of the terms in the sequence becomes

$$F = (u_1 * u_2 * \dots * u_{\phi(n)}) \bmod n$$

On the other hand, we may rearrange the product terms to separate all x factors, and produce

$$F = x^{\phi(n)} * (u_1 * u_2 * \dots * u_{\phi(n)}) \bmod n$$

Therefore,

$$x^{\phi(n)} * (u_1 * u_2 * \dots * u_{\phi(n)}) \bmod n = (u_1 * u_2 * \dots * u_{\phi(n)}) \bmod n$$

Since every u_i term has an inverse mod n , we multiply both sides by the inverse terms. This cancels all u_i terms from both sides and gives: $x^{\phi(n)} \bmod n = 1$. ■

Note that if n is prime, all integers in $(1, 2, \dots, n-1)$ are prime relative to n , and $\phi(n) = n-1$. In this case, Euler's Theorem reduces to Fermat's Little Theorem.

Computing Inverse by Exponentiation: Euler's Theorem gives another way of computing the inverse of an integer $x \bmod n$, where $x \in Z_n^*$. Namely,

$$x^{-1} = x^{\phi(n)-1} \bmod n$$

Example: Let $x = 7 \in Z_{15}^*$. Since $n = 15 = 3 * 5$, $\phi(n) = 8$. Then

$$x^{-1} = x^{\phi(n)-1} \bmod 15 = 7^7 \bmod 15 = 13.$$



Corollary 2: Let n be a positive integer, and $Z_n^* = \{u_1, u_2, \dots, u_{\phi(n)}\}$ be the set of elements prime relative to n . And let x be an integer in Z_n^* . Then, for any integer r ,

$$x^{r\phi(n)+1} \bmod n = x$$

Proof: By Euler's Theorem, $x^{\phi(n)} \bmod n = 1$. Raise both sides to the power r to get: $x^{r\phi(n)} \bmod n = 1$. Then multiply both sides by x . ■

Generalization of Euler's Theorem

Euler's Theorem requires $x \in Z_n^*$, so x is prime relative to n . Our next theorem removes this restriction, so x may be any integer.

Theorem 6: Let $n = pq$ be product of two primes p and q , and $\phi(n) = (p-1)(q-1)$. Let x be any positive integer, $x < n$. Then, for any integer r ,

$$x^{r\phi(n)+1} \bmod n = x$$

Proof: If $\gcd(x, n) = 1$, the theorem is true by the above corollary. Now suppose x is not prime relative to n . Since $x < n$, then x must be either a multiple of p or a multiple of q , but not both. Suppose $x = tp$, for some positive integer $t < q$. Since x is prime relative to q , then by Fermat's Little Theorem,

$$x^{q-1} \bmod q = 1.$$

Raise both sides to the power of $r(p-1)$ to get

$$x^{r(p-1)(q-1)} \bmod q = 1.$$

Thus, for some integer k ,

$$x^{r\phi(n)} = kq + 1$$

Now, multiply both sides by $x = tp$.

$$x^{r\phi(n)+1} = kt(pq) + x$$

$$x^{r\phi(n)+1} = kt n + x$$

$$x^{r\phi(n)+1} \bmod n = x.$$



Example: For $n = 15 = 3 * 5$, $\phi(n) = 8$, and $x = 5$,

$$x^{\phi(n)+1} \bmod 15 = 5^9 \bmod 15 = 5.$$



Example: Let $n = 15 = 3 * 5$, $\phi(n) = 2 * 4 = 8$. The following table shows a row $(x^i \bmod n \mid i = 1, 2, \dots, 9)$ for every positive integer $x < n$. The following observations are made from the table:

- For every x , column 9 confirms Theorem 6, namely $x^{\phi(n)+1} \bmod n = x$.
- For those rows where $x \in Z_n^*$ (highlighted in yellow), column 8 shows a value 1 thus confirming Euler's Theorem. That is, $x^{\phi(n)} \bmod n = 1$. And for these rows, column 7 confirms $x^{\phi(n)-1} \bmod n = x^{-1}$. ■

i	1	2	3	4	5	6	7	$8 = \phi(n)$	$9 = \phi(n) + 1$
$1^i \bmod n$	1	1	1	1	1	1	1	1	1
$2^i \bmod n$	2	4	8	1	2	4	8	1	2
$3^i \bmod n$	3	9	12	6	3	9	12	6	3
$4^i \bmod n$	4	1	4	1	4	1	4	1	4
$5^i \bmod n$	5	10	5	10	5	10	5	10	5
$6^i \bmod n$	6	6	6	6	6	6	6	6	6
$7^i \bmod n$	7	4	13	1	7	4	13	1	7
$8^i \bmod n$	8	4	2	1	8	4	2	1	8
$9^i \bmod n$	9	6	9	6	9	6	9	6	9
$10^i \bmod n$	10	10	10	10	10	10	10	10	10
$11^i \bmod n$	11	1	11	1	11	1	11	1	11
$12^i \bmod n$	12	9	3	6	12	9	3	6	12
$13^i \bmod n$	13	4	7	1	13	4	7	1	13
$14^i \bmod n$	14	1	14	1	14	1	14	1	14

8. The RSA Public-Key Cryptosystem

We now have the theoretical background for an understanding of the cryptosystem. The RSA public-key cryptosystem is named after its inventors: Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. It enables many senders to securely communicate with one receiver.

An example we use is the commerce communication between many clients or customers of a bank (called public) and the bank. All the clients have the same **public keys** they use to encode their messages. And the bank (recipient) holds a secret **private key** used to decode the messages. Furthermore, the public keys work in such a way that although all the senders (public) share the same public keys, they still will not be able to decipher each other's messages (even if they try to eavesdrop) because they lack the private keys needed to unlock the encoded messages.

Here is how the RSA cryptosystem works. First, the bank (receiver) privately picks two very large prime numbers p and q , and computes

$$n = pq, \quad \phi(n) = (p - 1)(q - 1).$$

The bank also picks a small number s which is **prime relative to $\phi(n)$** , and privately computes

$$t = s^{-1} \bmod \phi(n).$$

Notice the inverse is mod $\phi(n)$, not mod n . Thus, $st = r\phi(n) + 1$ for some integer r .

In practice, a small prime number may be chosen for s , but it must be done carefully to make sure that $\gcd(s, \phi(n)) = 1$. For example, suppose $n = 41 * 67$, thus $\phi(n) = 40 * 66 = 2640$. Then, for example, $s = 11$ would not work since it is not prime relative to 2640.

The bank announces the two integers s and n as **public keys** to all of its clients. The integer $t = s^{-1}$ is kept secret by the bank, and is called the **private key**.

The communication between the clients and the bank proceeds as follows.

1. The client (sender), who wants to transmit an information in the form of a large integer $x < n$, uses the public keys (s, n) and computes

$$y = x^s \bmod n$$

The value y , called the **encrypted** message, is transmitted over the Internet.

2. The bank (receiver) takes the data y , and uses its private key t , together with the public key n , to decrypt the message by computing

$$z = y^t \bmod n$$

By Theorem 6, we know the result z is indeed the original message x .

$$z = y^t \bmod n = (x^s \bmod n)^t \bmod n = x^{st} \bmod n = x^{r\phi(n)+1} \bmod n = x.$$

Example: Let us pick a small example for illustration. We choose two prime numbers $p = 5$, $q = 13$.

$$n = p * q = 5 * 13 = 65$$

$$\phi(n) = (p - 1)(q - 1) = 4 * 12 = 48$$

And let us pick a small prime number s , while making sure that $\gcd(s, \phi(n)) = 1$.

$$s = 11$$

Then,

$$t = s^{-1} \bmod 48 = (-13) \bmod 48 = 35$$

So, the public keys for all senders are: $s = 11$, $n = 65$. And the receiver keeps the private key $t = 35$, and the public key $n = 65$.

Suppose a sender wants to send a message (number) $x = 53$. The message x is encrypted using the public keys (s, n) .

$$y = x^s \bmod n = 53^{11} \bmod 65 = 27$$

The computed value y is transmitted. The receiver takes the data y and decrypts it, using the private key $t = 35$ and public key n .

$$z = y^t \bmod n = 27^{35} \bmod 65 = 53$$

And the result z is the original message x . 