e-PGPathshala

Subject : Computer Science

Paper: Cryptography and Network Security

Module: Prime's Euler and Fermat's Theorem

Module No: CS/CNS/10

Quadrant 1 – e-text

Cryptography and Network Security

Module 10- Prime's Euler and Fermat's Theorem

Learning Objectives

- To discuss about Euler's and Fermat's Theorem.
- To discuss various examples Euler's and Fermat's methods.
- To discus about generating primes
- To discuss about Primality testing and different Algorithms with the various examples.

10.1 Fermat's and Euler's Theorems

Two theorems that play important roles in public-key cryptography are Fermat's theorem and Euler's theorem.

Fermat's Theorem

This is sometimes referred to as **Fermat's little theorem**.

First version

Fermat's theorem states the following: If p is prime and a is a positive integer not divisible by p, then

$$a^{p-1} \equiv 1(\mathrm{mod}\ p)$$

Proof: Consider the set of positive integers less than *p*:{1,2,..., *p* 1} and multiply each element by *a*, modulo *p*, to get the set X = {*a* mod *p*, 2*a* mod *p*, . . . (*p*1)a mod *p*}. None of the elements of X is equal to zero because p does not divide *a*. Furthermore no two of the integers in X are equal. To see this, assume that (j*a* ≡ *p*) where 1 ≤ *p* 1. Because [5] to *p*, we can eliminate *a* from both sides of the equation [see Equation (4.3)] resulting in: j ≡ p). This last equality is impossible because *j* and *k* are both positive integers less than *p*. Therefore, we know that the (*p 1*) elements of X are all positive integers, with no two elements equal. We can conclude the X consists of the set of integers {1,2,..., *p1*} in some order. Multiplying the numbers in both sets and taking the result mod p yields

[5] Recall from Chapter 4 that two numbers are relatively prime if they have no prime factors in common; that is, their only common divisor is 1. This is

equivalent to saying that two numbers are relatively prime if their greatest common divisor is 1.

Second Version

An alternative form of Fermat's theorem is also useful: If p is prime and a is a positive integer, then

$$a^p \equiv a \pmod p$$

Exponentiation

Fermats little theorem sometimes is helpful for quickly finding a solution to some exponentiations. The following examples show the idea.

Example 10.1

Find the result of 6^{10} mod 11.

Solution

We have 6^{10} mod 11 = 1. This is the first version of Fermat's little theorem where p = 11.

Example 10.2

Find the result of 3^{12} mod 11

Solution

Here the exponent (12) and the modulus (11) are not the same. With substitution this can be solved using Fermat's little theorem.

$$3^{12} \bmod 11 = (3^{11} \times 3) \bmod 11 = (3^{11} \bmod 11)\,(3 \bmod 11) = (3 \times 3) \bmod 11 = 9$$

Multiplicative Inverses

A very interesting application of fermat's theorem is in finding some Multiplicative Inverses quickly if the modules is a prime. If *p* is a prime and a is an integer such that *p* does not divide *a(p|a)*,then a^{-1} mod p= a^{p-2} mod *p.*

This can be easily proved if we multiply both sides of the equality by *a* and use the first version of fermat's theorem.

$$a^{-1} \bmod p = a^{p-2} \bmod p$$

This application eliminates the use of extended Euclidean algorithm for finding some multiplicative inverse.

Example 10.3

The answers to multiplicative inverses modulo a prime can be found without using the extended Euclidean algorithm:

a. $8^{-1} \bmod 17 = 8^{17-2} \bmod 17 = 8^{15} \bmod 17 = 15 \bmod 17$
b. $5^{-1} \bmod 23 = 5^{23-2} \bmod 23 = 5^{21} \bmod 23 = 14 \bmod 23$
c. $60^{-1} \bmod 101 = 60^{101-2} \bmod 101 = 60^{99} \bmod 101 = 32 \bmod 101$
d. $22^{-1} \bmod 211 = 22^{211-2} \bmod 211 = 22^{209} \bmod 211 = 48 \bmod 211$

10.2 Euler's Theorem

Euler's Theorem can be thought of as a generalization of Fermat's little theorem. The modules in the Fermat theorem is a prime, the modulus in Euler's theorem is an integer. we introduce two versions of this theorem.

First version

The first version of Euler's theorem is similar to the first version of the Fermat's little theorem. if a and n are coprime,

Then Let a and m be coprime. Then $a^{\varphi(n)} = 1 \ (mod\ n)$.

The derivation of the Euler's formula for $\varphi(n)$ proceeds in two steps. First, we consider the next simplest case $\varphi(p^a)$, where p is prime.

Next, we establish the multiplicative property of φ:

$$\varphi(n_1 n_2) = \varphi(n_1)\varphi(n_2)$$

for coprime m_1 and m_2.

Since any integer can be (uniquely) represented in the form

$$n = p_1^{a}{}_1 p_2^{a}{}_2 \dots p_k^{a}{}_k,$$

with distinct p_i's, these two steps combined will lead to a closed form expression for φ.

Second version

The Second version of Euler's theorem is similar to the second version of Fermat's little theorem; it removes the condition that a and n should be coprime.

If $n = p \ X \ q$, $a<n$, and k an integer, then $a^{k \times f(n) + 1} \equiv a \ (mod\ n)$

Let us give an informal proof of the second version based on the first version. because $a<n$, three cases are possible:

1. If a is neither a multiple of p nor a multiple of q ,then a and n are coprimes.

$$a^{k\times\phi(n)+1} \bmod n = (a^{\phi(n)})^k \times a \bmod n = (1)^k \times a \bmod n = a \bmod n$$

2. if a is a multiple of p(a=I x p),but not a multiple of q

$$a^{\phi(n)} \bmod q = (a^{\phi(q)} \bmod q)^{\phi(p)} \bmod q = 1 \;\rightarrow\; a^{\phi(n)} \bmod q = 1$$

$$a^{k\times\phi(n)} \bmod q = (a^{\phi(n)} \bmod q)^k \bmod q = 1 \;\rightarrow\; a^{k\times\phi(n)} \bmod q = 1$$

$$a^{k\times\phi(n)} \bmod q = 1 \;\rightarrow\; a^{k\times\phi(n)} = 1 + j\times q \quad \text{(Interpretation of congruence)}$$

$$a^{k\times\phi(n)+1} = a\times(1+j\times q) = a + j\times q\times a = a + (i\times j)\times q\times p = a + (i\times j)\times n$$

$$a^{k\times\phi(n)+1} = a + (i\times j)\times n \;\rightarrow\; a^{k\times\phi(n)+1} = a \bmod n \quad \text{(Congruence relation)}$$

3. if a is a multiple of q(a = I x q),but not a multiple of p,the proof is the same as for the second case,but the roles of p and q are changed.

The second version of Euler's theorem is used in the RSA cryptosystem.

Applications

Although we will see some applications of Euler's Later in this chapter, the theorem is very useful for solving some problems.

Exponentiation

Euler's theorem some times is helpful for quickly finding a solution to some exponentiations. The following examples shows the idea.

Example 10.4

Find the result of 6^{24} mod 35.

Solution

We have 6^{24} mod 35 = $6^{f(35)}$ mod 35 = 1.

Example 10.5

Find the result of 20^{62} mod 77.

Solution

If we let *k* = 1 on the second version, we have

$$20^{62} \bmod 77 = (20 \bmod 77)\ (20^{f(77)\ +\ 1} \bmod 77) \bmod 77$$

$$= (20)(20) \bmod 77 = 15.$$

Multiplicative inverse

Euler's theorem can be used to find multiplicative inverse modulo a prime.also with a composite.if n and a are coprime,then $a^{-1} \bmod n = a^{f(n)-1} \bmod n$

This can be easily proved if we multiply both sides of the equality by a:

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

Example 10.6

The answers to multiplicative inverses modulo a composite can be found without using the extended Euclidean algorithm if we know the factorization of the composite:

a. $8^{-1} \bmod 77 = 8^{\phi(77)-1} \bmod 77 = 8^{59} \bmod 77 = 29 \bmod 77$
b. $7^{-1} \bmod 15 = 7^{\phi(15)-1} \bmod 15 = 7^{7} \bmod 15 = 13 \bmod 15$
c. $60^{-1} \bmod 187 = 60^{\phi(187)-1} \bmod 187 = 60^{159} \bmod 187 = 53 \bmod 187$
d. $71^{-1} \bmod 100 = 71^{\phi(100)-1} \bmod 100 = 71^{39} \bmod 100 = 31 \bmod 100$

Generating Primes

Two mathematicians, Mersenne and Fermat, attempted to develop a formula that could Generate Primes.

Mersenne Primes

Mersenne defined the following formula, which is called the Mersenne numbers, that was supposed to enumerate all primes.

$$\mathbf{M_p = 2^p - 1}$$

If the p above formula is a prime ,then M_p was to be aprime, years later was proven that not all numbers created by the Mersenne formula are primes. The following lists some Mersenne numbers.

$M_2 = 2^2 - 1 = 3$

$M_3 = 2^3 - 1 = 7$

$M_5 = 2^5 - 1 = 31$

$M_7 = 2^7 - 1 = 127$

$M_{11} = 2^{11} - 1 = 2047$ **Not a prime ($2047 = 23 \times 89$)**

$M_{13} = 2^{13} - 1 = 8191$

$M_{17} = 2^{17} - 1 = 131071$

It turned out that M_{11} is not a prime. however, 41 Mersenne primes have been found; the latest one is M124036583, a very large number with 7,253,733 digits, the search continues.

A number in the form $M_p = 2^p - 1$ is called a Mersenne number and may or may not be a prime.

Fermat Primes

Fermat tried to find aformula to generate primes. The following formula is a Fermat number:

$$F_n = 2^{2^n} + 1$$

Fermat tested numbers up to F_4 ,but it turned out that F_5 is not a prime. no number.

$F_0 = 3$

$F_1 = 5$

$F_2 = 17$

$F_3 = 257$

$F_4 = 65537$

$F_5 = 4294967297 = 641 \times 6700417$ *Not a prime.*

Greater than F_4, has been proven to be a prime.As a m a tter of f act m any numbers up to F_{24} h a ve been proven to be composite numbers.

10.2 PRIMALITY TESTING

Finding an algorithm to correctly and efficiently test a very large integer and output a prime or a composite has always been a challenge in number theory, and consequently in cryptography. However, recent developments look very promising.

Algorithms that deal with this issue can be divided into two categories. one is deterministic algorithm and another one is Probabilistic algorithms. A deterministic algorithm always gives a correct answer; a Probabilistic algorithms gives an answer that is correct most of the time,but not all of the time. Although a deterministic algorithm is deal ,it is normally less efficient than the corresponding probabilistic one.

Deterministic Algorithms

A deterministic prim ality testing algorithm accepts an integer and always outputs a prime or a composite. all deterministic a lgorithms were so inefficient at finding l a rger primes th a t they were considered infeasible. As we will show shortly, a newer algorithms looks more promising.

Divisibility Algorithm

The most elementary deterministic test for primality is the divisibility test. We use as divisors all numbers small that $\sqrt{n}$. If any of these numbers divides n,then n is composite.

Algorithm shows the divisibility test in primitive,very inefficient form.

This algorithm can be improved y testing only odd numbers. It can be further improved by using a table of primes between 2 and $\sqrt{n}$. The number of arithmetic operations on algorithm10.1. is $\sqrt{n}$. if we assume that each arithmetic operation uses only one bit operation (unrealistic) then the bit-operation complexity of algorithm 10.1 $f(n_b) = \sqrt{2^{n_b}} = 2^{n_b/2}$. Where n_b is the number of bits in – n. in big O notation ,the complexity can be shown as $O(2^{n_b})$;exponential,in other words, the divisibility algorithm is feasible(intractable) if n_b is large.

The bit-operation complexity of the divisibility test is exponential.

Algorithm 9.1 *Pseudocode for the divisibility test*

```
Divisibility_Test (n)                    // n is the number to test for primality
{
  r ← 2
  while (r < √n )
  {
   if (r | n) return "a composite"
   r ← r + 1
  }
   return "a prime"
}
```

Example 10.7

Assume *n* has 200 bits. What is the number of bit operations needed to run the divisibility-test algorithm?

Solution

The bit-operation complexity of this algorithm is $2^{n_b/2}$. This means that the algorithm needs 2^{100} bit operations. On a computer capable of doing 2^{30} bit operations per second, the algorithm needs 2^{70} seconds to do the testing (forever).

AKS Algorithm

In 2002 ,Agrawal, kayal, and Saxena announced that they had found an algorithm for primality testing with polynomial bit-operation time complexity of $O((\log_2 n_b)^{12})$. The algorithm uses the fact that $(x-a)^p \equiv (x^p - a)$ *mod p.* it is not surprising to see some future refinements makes this algorithm the standard primality test in mathematics and computer science.

Example 10.8

Assume *n* has 200 bits. What is the number of bit operations needed to run the AKS algorithm?

Solution

This algorithm needs only $(\log_2 200)^{12}$ = 39,547,615,483 bit operations. On a computer capable of doing 1 billion bit operations per second, the algorithm needs only 40 seconds.

Probabilistic Algorithms

This methods may be used for a while until the AKS is formally accepted as the standard. A probabilistic algorithm does not guarantee the correctness of the result. Algorithm in this category returns either a prime or composite based on the following rules:

1. If the integer to be tested is actually prime,the algorithm definitely returns a prime.
2. If the integer to be tested is actually a composite,it returns a composite with probability $1-\varepsilon$, but it may return a prime with the probability ε.

The probability of mistake can be improved if we run the algorithm more than once with different parameters or using different methods. If we run the algorithm m times. The probability of error may reduce to ε^m.

Fermats Test

The first probabilistic method we discussis this Fermat primality test.Result the Fermat little theorem.

$$\textbf{If } n \textbf{ is a prime, then } a^{n-1} \equiv 1 \bmod n.$$

1. If n is a prime, the congruence holds. it does not mean that if the congruence holds ,n is a prime . the integer can be a prime or composite. We define the as the following as the Fermat test.
2. If n is a prime , $a^{n-1} \equiv 1 \bmod n$
3. If n is a composite, it is possible that $a^{n-1} \equiv 1 \bmod n$.

Example 10.9

Does the number 561 pass the Fermat test?

Solution

Use base 2

$$2^{561-1} = 1 \bmod 561$$

The number passes the Fermat test, but it is not a prime, because 561 = 33 × 17.

Square Root Test

In modular arithmetic ,if n is a prime, the square root of 1 is either +1 or -1.if n is composite, the square root is +1 or -1,but there may be other roots. This is knows as the square root primality test.

Note that in modular arithmetic, -1 means n-1

If n is a prime, $\sqrt{1} \bmod n = \pm 1$.
If n is a composite, $\sqrt{1} \bmod n = \pm 1$ and possibly other values.

Example 10.10

What are the square roots of 1 mod *n* if *n* is 7 (a prime)?

Solution

The only square roots are 1 and −1. We can see that

$1^2 = 1 \bmod 7$	$(-1)^2 = 1 \bmod 7$
$2^2 = 4 \bmod 7$	$(-2)^2 = 4 \bmod 7$
$3^2 = 2 \bmod 7$	$(-3)^2 = 2 \bmod 7$

Note that we don't have to test 4,5 and 6 because4=-3 mod 7,5= -2 mod 7 and 6= -1 mod 7.

Example 10.11

What are the square roots of 1 mod *n* if *n* is 8 (a composite)?

Solution

There are four solutions: 1, 3, 5, and 7 (which is −1). We can see that

$$1^2 = 1 \bmod 8 \qquad (-1)^2 = 1 \bmod 8$$
$$3^2 = 1 \bmod 8 \qquad 5^2 = 1 \bmod 8$$

Example 10.12

What are the square roots of 1 mod *n* if *n* is 17 (a prime)?

Solution

There are only two solutions: 1 and −1

$$\begin{array}{ll}
1^2 = 1 \bmod 17 & (-1)^2 = 1 \bmod 17 \\
2^2 = 4 \bmod 17 & (-2)^2 = 4 \bmod 17 \\
3^2 = 9 \bmod 17 & (-3)^2 = 9 \bmod 17 \\
4^2 = 16 \bmod 17 & (-4)^2 = 16 \bmod 17 \\
5^2 = 8 \bmod 17 & (-5)^2 = 8 \bmod 17 \\
6^2 = 2 \bmod 17 & (-6)^2 = 2 \bmod 17 \\
(7)^2 = 15 \bmod 17 & (-7)^2 = 15 \bmod 17 \\
(8)^2 = 13 \bmod 17 & (-8)^2 = 13 \bmod 17
\end{array}$$

Example 10.13

What are the square roots of 1 mod *n* if *n* is 22 (a composite)?

Solution

Surprisingly, there are only two solutions, +1 and −1, although 22 is a composite.

$$1^2 = 1 \bmod 22$$
$$(-1)^2 = 1 \bmod 22$$

Miller-Rabin Test

The Miller-Rabin Primality test combines the Fermat test and square root test in a very elegant way to find a strong pseudoprime (a prime with a very high probability). In this test, we write *n-1* as the product of an odd number *m* and *a* power of 2.

$$n - 1 = m \times 2^k$$

The Fermat test in base a can be written as shown in figure 10.2

Figure 10.2 *Idea behind Fermat primality test*

$$a^{n-1} = a^{m \times 2^k} = \left[a^m\right]^{2^k} = \left[a^m\right]^{\underbrace{2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}}_{k \text{ times}}}$$

Algorithm 10.2 shows the pseudocode for the Miller-Rabin test

Algorithm 9.2 *Pseudocode for Miller-Rabin test*

```
Miller_Rabin_Test (n, a)                         // n is the number; a is the base.
{
    Find m and k such that n − 1 = m × 2^k
    T ← a^m mod n
    if (T = ± 1)   return "a prime"
    for (i ← 1 to k − 1)                         // k − 1 is the maximum number of steps.
    {
        T ← T^2 mod n
        if (T = +1) return "a composite"
        if (T = −1) return "a prime"
    }
    return "a composite"
}
```

There exists a proof that each time a number passes a Miller-Rabin test ,the probability that it is not a prime is ¼. If the number passesm tests(with m different bases),the probability that its not a prime is (1/4) the power of m. $(1/4)^m$.

Example 10.14

Does the number 561 pass the Miller-Rabin test?

Solution

Using base 2, let $561 - 1 = 35 \times 2^4$, which means $m = 35$, $k = 4$, and $a = 2$.

Initialization:	$T = 2^{35} \bmod 561 = 263 \bmod 561$	
$k = 1$:	$T = 263^2 \bmod 561 = 166 \bmod 561$	
$k = 2$:	$T = 166^2 \bmod 561 = 67 \bmod 561$	
$k = 3$:	$T = 67^2 \bmod 561 = +1 \bmod 561$	$\rightarrow$ **a composite**

Example 10.15

We already know that 27 is not a prime. Let us apply the Miller-Rabin test.

Solution

With base 2, let $27 - 1 = 13 \times 2^1$, which means that $m = 13$, $k = 1$, and $a = 2$. In this case, because $k - 1 = 0$, we should do only the initialization step: $T = 2^{13} \bmod 27 = 11 \bmod 27$. However, because the algorithm never enters the loop, it returns a composite.

Example 10.16

We know that 61 is a prime, let us see if it passes the Miller-Rabin test.

Solution

We use base 2.

$$61 - 1 = 15 \times 2^2 \rightarrow \quad m = 15 \quad k = 2 \quad a = 2$$
$$\textit{Initialization:} \quad T = 2^{15} \bmod 61 = 11 \bmod 61$$
$$k = 1 \quad T = 11^2 \bmod 61 = -1 \bmod 61 \quad \rightarrow \quad \textbf{a prime}$$

Recommended Primality Test

Today, one of the most popular primality test is a combination of the divisibility test and the Miller-Rabin test.

1. choose an odd integer , because all even integers re definitely composites.
2. do some trivial divisiblitytests on some known prime such as 3,5,7,11,13… and so on to be sure that you are not dealing with an obvious composite. If the number passes all of these tests,move to the next step. Ifthe number fails any of the test s,go back to step 1 and choose another odd number.
3. choose a set of bases for testing. A large set of bases is preferable.

4. Do Miller-Rabin tests on each of the bases. If any of them fails ,go back to step 1 and choose another odd number. If the test passes for all bases,declare the number a strong pdeudoprime.

Example 10.17

The number 4033 is a composite (37 × 109). Does it pass the recommended primality test?

Solution

1. Perform the divisibility tests first. The numbers 2, 3, 5, 7, 11, 17, and 23 are not divisors of 4033.

2. Perform the Miller-Rabin test with a base of 2, 4033 − 1 = 63 × 26, which means *m* is 63 and *k* is 6

Initialization: $T \equiv 2^{63} \pmod{4033} \equiv 3521 \pmod{4033}$

$k = 1$ $\quad T \equiv T^2 \equiv 3521^2 \pmod{4033} \equiv -1 \pmod{4033}$ $\rightarrow$ **Passes**

Example 10.18

3. But we are not satisfied. We continue with another base, 3.

Initialization: $T \equiv 3^{63} \pmod{4033} \equiv 3551 \pmod{4033}$

$k = 1$ $\quad T \equiv T^2 \equiv 3551^2$ (mod 4033 $\equiv 2443 \pmod{4033}$)

$k = 2$ $\quad T \equiv T^2 \equiv 2443^2$ (mod 4033 $\equiv 3442 \pmod{4033}$)

$k = 3$ $\quad T \equiv T^2 \equiv 3442^2$ (mod 4033 $\equiv 2443 \pmod{4033}$)

$k = 4$ $\quad T \equiv T^2 \equiv 2443^2$ (mod 4033 $\equiv 3442 \pmod{4033}$)

$k = 5$ $\quad T \equiv T^2 \equiv 3442^2$ (mod 4033 $\equiv 2443 \pmod{4033}$) $\rightarrow$**Failed (composite)**