# ECE 443/518 – Computer Cyber Security Lecture 08 Euclidean Algorithm, Fermat's Little Theorem

Professor Jia Wang Department of Electrical and Computer Engineering Illinois Institute of Technology

September 16, 2024

ECE 443/518 – Computer Cyber Security, Dept. of ECE, IIT

1/20

### Outline

Euclidean Algorithm

Fermat's Little Theorem

- ► This lecture: UC 6.3
- Next lecture: UC 6, 7, except 7.6

### Outline

Euclidean Algorithm

Fermat's Little Theorem

### Euclidean Algorithm

Input: two integers  $a \ge b > 0$ 1  $r_0 = a, r_1 = b, i = 1$ 2 **Do**: 3 i = i + 14  $r_i = r_{i-2} \mod r_{i-1}$ 5 **While**  $r_i \ne 0$ Output:  $gcd(a, b) = r_{i-1}$ 

• Example:  $r_0 = 27, r_1 = 21, r_2 = 6, r_3 = 3, r_4 = 0$ 

In practice, there is no need to keep each r<sub>k</sub> – we use them just for ease of presentation.

For a proof of correctness

Is this algorithm better than the simple one?

ECE 443/518 - Computer Cyber Security, Dept. of ECE, IIT

5/20

#### Time Complexity of Euclidean Algorithm

• Let 
$$q_{k-1} = \lfloor \frac{r_{k-2}}{r_{k-1}} \rfloor$$
. Since  $r_{k-2} \ge r_{k-1}$ ,  $q_{k-1} \ge 1$ . So,

 $r_{k-2} = q_{k-1}r_{k-1} + r_k \ge r_{k-1} + r_k \ge 2r_k, \forall k = 2, 3, \dots, i.$ 

For i being odd, we have,

$$a = r_0 \ge 2r_2 \ge 2^2 r_4 \ge \cdots \ge 2^{\frac{i-1}{2}} r_{i-1} \ge 2^{\frac{i-1}{2}}.$$

Similar for i being even.

The loop iterates O(log a) = O(N) rounds.

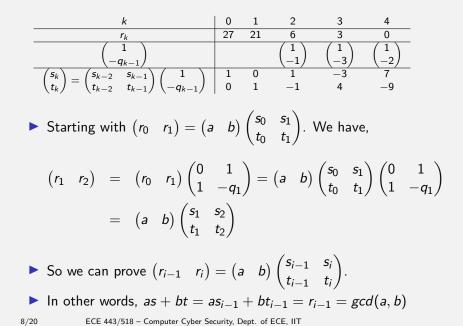
• Overall the time complexity is  $O(N^3)$ .

- GCD can be computed efficiently in polynomial time.
  - What is the complexity to obtain any divisor of a that is not 1 or a? Or to prove that a is a prime number?

## Extended Euclidean Algorithm (EEA)

- Same time complexity as Euclidean Algorithm:  $O(N^3)$ 
  - Same rounds of iterations. Additional calculations do not increase complexity.
- Anything special?

#### Extended Euclidean Algorithm (EEA Cont.)



### Solve Modular Algebra Equations

 $\blacktriangleright ax \equiv b \pmod{m}$ 

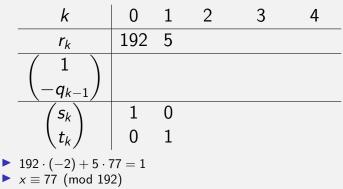
9/20

- Assume gcd(a, m) = 1.
- Apply EEA to find s and t such that as + mt = 1.
- Solution:  $x \equiv bs \pmod{m}$
- Check:  $ax \equiv abs \equiv b(1 mt) \equiv b bmt \equiv b \pmod{m}$ .

• Time complexity is  $O(N^3)$ , dominated by EEA.

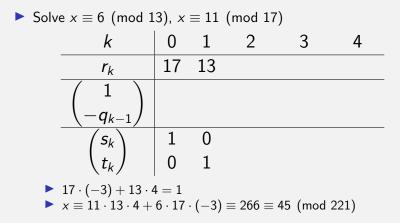
#### Examples 1

Solve  $5x \equiv 1 \pmod{192}$ .



## Solve System of Modular Algebra Equations

Examples 2



### Outline

**Euclidean Algorithm** 

Fermat's Little Theorem

What about modular n-th root?

 $x^n \equiv a \pmod{m}$ .

Obviously you can solve it via brute-force in O(2<sup>N</sup>) time for a N-bit m. However, this is not what we are interested into.

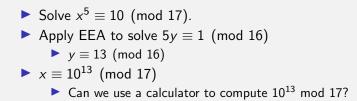
• Consider the case when m = p is a prime number first.

#### Fermat's Little Theorem

Consider an integer x that is not a multiple of p. What does the sequence kx mod p look like for  $k = 1, 2, \ldots, p - 1?$ A permutation of  $1, 2, \ldots, p-1$  since • These p-1 remainders are all within  $1, 2, \ldots, p-1$ . They are all different since p is prime. So  $x \cdot (2x) \cdots ((p-1)x) \equiv 1 \cdot 2 \cdots (p-1) \pmod{p}$ . ln other words,  $(p-1)!x^{p-1} \equiv (p-1)! \pmod{p}$ . • So  $x^{p-1} \equiv 1 \pmod{p}$  since gcd((p-1)!, p) = 1. Fermat's Little Theorem:  $x^p \equiv x \pmod{p}$ Also include the case  $x \equiv 0 \pmod{p}$ . • Example:  $2^{13} \equiv 2 \pmod{13}$ ,  $3^{13} \equiv 3 \pmod{13}$ .

#### Solve Modular *n*-th Root for Prime *p*

Solve  $x^5 \equiv 2 \pmod{13}$ . ▶  $x^{10} \equiv 4 \pmod{13}$ ,  $x^{15} \equiv 8 \pmod{13}$ ,  $x^{25} \equiv 6 \pmod{13}$ . Fermat's Little Theorem:  $x^{13} \equiv x \pmod{13}$ • So  $x^{25} \equiv x^{13}x^{12} \equiv xx^{12} \equiv x \pmod{13}$ . Solution:  $x \equiv 6 \pmod{13}$ • How about  $x^n \equiv a \pmod{p}$ ? Assume gcd(n, p-1) = 1. No, you can't use this method if n = 2. Solve  $ny \equiv 1 \pmod{p-1}$  for y (via EEA). Solution:  $x \equiv a^y \pmod{p}$ , or practically  $x = a^y \mod p$ . • Check:  $x^n \equiv a^{ny} \equiv a^{(ny) \mod (p-1)} \equiv a \pmod{p}$ . Time complexity EEA takes  $O(N^3)$  time. a<sup>y</sup> mod p can be completed in O(N<sup>3</sup>) time. (How?) • Overall  $O(N^3)$  time again!



# Square-and-Multiply

Compute 10<sup>13</sup> mod 17

$$\blacktriangleright \ 10^{13} \equiv 10^8 \cdot 10^4 \cdot 10^1 \pmod{17}$$

▶ Since 13 = (1101)<sub>2</sub>

• Use square to calculate  $10^2 \mod 7$ ,  $10^4 \mod 7$ , etc.

• 
$$10^2 \equiv 100 \equiv 15 \pmod{17}$$

▶ 
$$10^4 \equiv 225 \equiv 4 \pmod{17}$$

▶ 
$$10^8 \equiv 16 \pmod{17}$$

• So  $x \equiv 10^{13} \equiv 16 \cdot 4 \cdot 10 \equiv 11 \pmod{17}$ 

lndeed, this algorithm computes  $a^{\gamma}$  mod p in  $O(N^3)$  time.

O(N) modular multiplications.

$$10^{13} \equiv 10^{12} \cdot 10$$
  
$$\equiv 100^{6} \cdot 10 \equiv 15^{6} \cdot 10$$
  
$$\equiv 225^{3} \cdot 10 \equiv 4^{3} \cdot 10$$
  
$$\equiv 4^{2} \cdot 40 \equiv 4^{2} \cdot 6$$
  
$$\equiv 16 \cdot 6 \equiv 96 \equiv 11 \pmod{17}$$

Be creative with your calculators!

- EEA is essential for solving modular algebra equations.
  - In particular, if gcd(a, b) = 1, we can apply EEA to find integers s and t such that as + bt = 1.
- EEA is efficient with a time complexity of O(N<sup>3</sup>) for N-bit inputs.
- With Fermat's Little Theorem, we are able to solve modular *n*-th root for prime numbers in many cases for O(N<sup>3</sup>) time as well.