MONASH University
Information Technology

FIT2093 INTRODUCTION TO CYBER SECURITY



# Week 3 Lecture



# Cryptography II:

# Public Key Encryption: Part 1 – Concept and Maths

## Principles for CONFIDENTIALITY

# Outline

**Last lecture (week 2) -- Symmetric Key Encryption**

**This week (week 3):**

- **Public Key Encryption -- PKE** (aka Asymmetric Key Encryption)
  - **Why** do we need public key cryptography?
  - The **concept** of public key cryptography
- **Maths of PKE**
  - The **math** behind public key cryptography: **number theory**



**Next Week (week 4): How PKE Algorithms work**

- Diffie-Hellman key exchange & ElGamal encryption
- RSA encryption

MONASH University

# Public Key Encryption (PKE)

# Symmetric Key Encryption (SKE): limitation

Public Key Crypto

- both sides share same secret key
- Q: how to share a key privately?
  - key distribution problem



# Symmetric Key Encryption (SKE): limitation

Public Key Crypto

- key distribution problem:
  - how to share?
    - far apart / never met before / channel insecure
  - Q: why need to share key privately?



# Public Key Encryption (PKE)

Public Key Crypto

- Idea of PKE (Diffie-Hellman 1976): Make the encryption key public
- Encrypt with one key (public key), decrypt with another key (private key)
- no need to share privately (no private key distribution problem)
- public key need **not** be kept secret



# PKE vs SKE

Public Key Crypto

| | PKE | SKE |
|---|---|---|
| **Keys** | Each user has a public key pk & private key sk; pk & sk are mathematically related | Each user privately shares a secret key k with another |
| Do, Undo | Do with pk, undo with sk | Do and undo with k |
| Problem | No Key Distribution Problem | Key Distribution Problem |
| Speed | Slower. **Q:** Why? | Faster |

# PKE Requirements

Public Key Crypto

- Functionality: easy for good guys
  - generate key pair pk & sk
  - encrypt, decrypt
- Security: computationally infeasible for bad guys
  - get sk from pk
  - get message P from C and pk



MONASH University

# Maths of PKE

# Maths of PKE

Public Key Crypto

- Why does PKE need maths?
  - two keys need to be related
    - do with one, undo with another
- Why SKE no need maths?
  - no such requirement
- What type of maths?
  - number theory
  - why number theory?

# Maths of PKE

Public Key Crypto

- Why PKE needs number theory?
  - two keys need to be **related**
    - s.t. **do** with one, **undo** with another
      - use pk to get C
      - use sk to "cancel out" pk from $C_{pk}$ to get back P
    - yet knowing pk, **cannot** compute the sk
      - should have one-way function from sk to pk
- *we recap on high school maths, and a little bit more*

# Maths of PKE

Public Key Crypto

- Commonly used PKE algorithms (next week's lecture):
  - based on **discrete logarithm problem** (DLP)
    - e.g. ElGamal Encryption [ElGamal 1994]



  - based on Integer Factorisation Problem (IFP)
    - e.g. RSA Encryption [Rivest, Shamir, Adleman 1977]

Both based on number theory problems!

So, let's learn a little about those…

# The Maths behind PKE

## Number Theory

**What type of maths is useful for PKE?**

- Number Theory: playing with integers (0,1,2,3,…)
- Brief History:
  - Ancient times to 1970s: numerous beautiful discoveries
    - But no known practical uses (for mathematicians only)!
  - 1970s: Unexpected discovery of use in cryptography
    - How to build a PKE!
  - 1980s – now: numerous new ”impossible sounding” applications in cryptography
    - E.g. **homomorphic encryption**: how to compute on encrypted data without decrypting it (will not cover here, useful for, e.g. private outsourcing of computation to the cloud)

# Integers *... revision*

Public Key Crypto

- A whole number including its negative and 0,
  - Examples: 1, 5, −2, 100, etc.
- Has basic mathematical **properties** e.g.
  - Adding/Multiplying 2 integers gives an integer too
  - Adding/Multiplying: order does not matter
  - Add 0 / Multiply 1: nothing changes
  - Inverse for Add: −x
  - Inverse for Mul: 1/x (but not integer)
  - …

# Prime vs Composite numbers

Public Key Crypto

- **Prime number**
  - has exactly two (distinct) divisors/factors, which are 1 and the (prime) number itself
  - Example: 2, 3, 5, 7, 11, 13, 17, 19, ...
- Composite number (the opposite)
  - has at least one positive divisor other than 1 and itself
  - Example: 4 has 1, 2 and 4 as possible divisors



- Number 0 and 1 are special numbers. They are not considered as prime or composite numbers

# Prime Numbers

Public Key Crypto

- **List of prime numbers p < 200**

```
2  3  5  7  11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83
89 97 101 103 107 109 113 127 131 137 139 149 151 157 163 167 173
179 181 191 193 197 199
```

# **Factor** (divisor)

Public Key Crypto

- x is factor of n if n can divide x without remainder (R = 0)
- **Examples:**
  - **2** is a factor of 4 because
    - 4 ÷ **2** = 2 (quotient) remainder 0
  - **2** is not a factor of 9 because
    - 9 ÷ **2** = 4 (quotient) and remainder = 1
- **Trivial factor** (not interesting, it's obvious)
  - A trivial factor of an integer *n* is the number 1 and the number *n* itself

# # Interesting Factors

Public Key Crypto

- Definition: **non-trivial factor** = factor that's not trivial
  - more interesting (not 1 and itself)



- **Example:** for number 6,
  - 1 and 6 are the trivial factors,
  - **2 and 3 are non-trivial factors**





  - *since 2 & 3 are prime, we call them **prime factors**:*
    - *They don't decompose into smaller factors*

# Integer Factorisation

Public Key Crypto

- **break** down a composite (it cannot be prime) **number** into smaller non-trivial **factors**
  - i.e. find the factors
- **Examples:**
  - 6 = 2 x 3
  - 12 = 2 x 2 x 3
  - 18 = 2 x 3 x 3

# # Integer Factorisation Problem

Public Key Crypto

- **Prime factorisation**
  - find the prime factors of n, may appear multiple times
  - factoring n as a product of prime powers
    - $15 = 3 \times 5, \quad 24 = 2^3 \times 3$
    - $3600 = 2^4 \times 3^2 \times 5^2$
- **Theorem (Prime Factorisation):** Every integer can be factored into a product of prime powers in a **unique** way



- **Integer factorisation Problem:** Given an integer n, compute the prime factorisation of n

# # Integer Factorisation Problem

Public Key Crypto

- **Integer Factorisation Problem (IFP):** Given an integer n, compute the prime factorisation of n





- Gist: it definitely exists (due to the proven theorem), just **find** it (the prime factors)



Q: What is the prime factorisation of 63?

# Integer Factorisation Problem: **Computational Difficulty**

Public Key Crypto

**Integer factorisation Problem (IFP):** Given an integer n, compute the prime factorisation of n



- Do one direction (opposite to IFP) is easy: multiply numbers
  - Given prime factors of n, **multiply** factors to get n
  - primary school maths





- Do other direction (IFP) is computationally difficult: factor
  - Factor large number n is harder than multiplying
  - some algorithms exist but not efficient

Q: How do we measure efficiency of algorithms?

# Is an Algorithm Efficient?

Public Key Crypto

- Time T (or number of steps) to run algorithm, depends on
  - **Length len(n)** of input i.e. how many digits/bits that n has
- **Examples**
- **Decimal (base 10):**
  - $len_{10}(3) = 1$
  - $len_{10}(34) = 2$
- **Binary (base 2):**
  - $len_2(3) = len_2(11_2) = 2$
  - $len_2(34) = len_2(100010_2) = 6$

# Is Algorithm Efficient?

Public Key Crypto

- Efficient algorithm:
  - T = polynomial function of len(n)
  - e.g. multiply two 3-digit numbers n=456 and m=123



456 so len(n) = len(m) = 3

x 123



---

zzz



zzz

zzz

T = 9 steps = $3^2$ = len(n)$^2$

# Is Algorithm Efficient?

Public Key Crypto

- Inefficient algorithm:
  - T = super-polynomial **i.e. > polynomial** (e.g. exponential) function of len(n)
  - e.g. integer factorisation
    - $T = O(2^{len(n)})$ (exponential),
    - $T = O(2^{\sqrt{len(n)}})$ (sub-exponential)

# Integer Factorisation is Computationally Difficult

Public Key Crypto

- Computationally difficult?
  - known methods are
    - not efficient (cannot do in polynomial time)
    - inefficient (need super-polynomial time)



- Methods
  - Trial division: time T ~ exponential in len(n)
  - Fermat's method: T ~ exponential in len(n)
  - Pollard's rho method: T ~ exponential in len(n)
  - Quadratic Sieve: time T ~ $2^{O(\sqrt{len(n)})}$ sub-exponential
  - Number Field Sieve: T ~ $2^{O(\sqrt[3]{len(n)})}$ sub−exponential

# Integer Factorisation is Computationally Difficult

Public Key Crypto

- Strategy: how to use infeasible integer factorisation problem for PKE?
  - Do encryption so that attacking requires factorisation
  - used to get the security for **RSA encryption (next week)**

# Greatest Common Divisor (GCD)

Public Key Crypto

- largest positive integer that can divide two integers without leaving a remainder
- **Examples:**
  - GCD of numbers 6 and 9: GCD(6,9) = 3
    - 6 = 2 x **3**
    - 9 = 3 x **3**
    - The greatest common factor = 3 = GCD(6, 9)
  - GCD of integers 48 and 64: GCD(48,64) = 16
    - 48 = **2 x 2 x 2 x 2** x 3 = **16** x 3 = 8 x 6 = 4 x 12 = 2 x 24
    - 64 = **2 x 2 x 2 x 2** x 2 x 2 = 8 x 8 = **16** x 4 = 32 x 2
    - GCD (48, 64) = 16



**Surprising Fact:** There's an efficient algorithm for computing GCD.

- Called Euclid's algorithm (we won't study)-- no need to factor integers!

# Relatively Prime / Co-Prime

Public Key Crypto

- Two numbers x, y are **relatively prime**
  - if **GCD(x,y) = 1**
- **Examples:**
  - 5 and 6 are relatively prime since GCD(5,6) = 1
  - 6 and 9 are not relatively prime because GCD (6,9) = 3
    - Both 6 and 9 can be divided by 3 without leaving any remainder



**Q:** What is the GCD of 28 and 25? Are they relatively prime?
How about 66 and 44?

# Modular Arithmetic

Public Key Crypto

- Modular arithmetic = clock arithmetic
- Clock has 12 numbers, for whatever the number, restart after 12 steps i.e. mod 12,



e.g. 14.00 hours ≡ 2 PM

14 mod 12 ≡ 2

14 ≡ 2 (mod 12)




- 14 and 2 occupy the same position in the 12-hrs clock
- 14 and 2 are **congruent** modulo 12

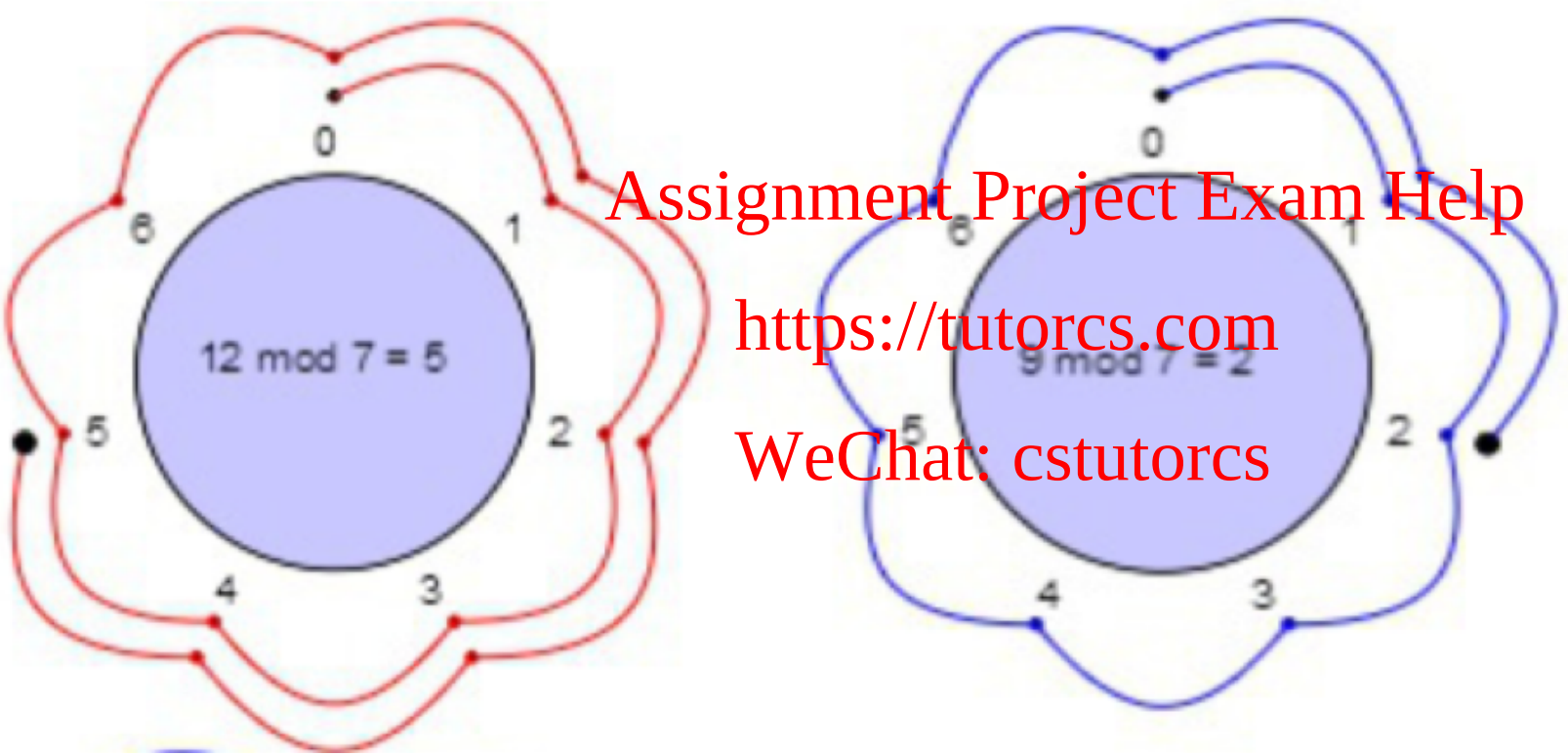
# Modular Arithmetic

Public Key Crypto

- **Examples**
  - 25 mod 12 = 1, 25 ≡ 1 (mod 12)
  - 9 mod 3 = 0, 9 ≡ 0 (mod 3)
  - 10 mod 3 = 1, 10 ≡ 1 (mod 3)
  - 12 mod 7 = 5, 12 ≡ 5 (mod 7)
  - 9 mod 7 = 2 9 ≡ 2 (mod 7)

# Modular Arithmetic

## Public Key Crypto



# Modular Arithmetic Trick

Public Key Crypto

- **( c + d ) mod n ≡ ( c mod n + d mod n ) mod n**
  - add then mod
  - Is equivalent to,
  - mod each, then add and mod

Q: What is 42 + 45 mod 15? Use the mod addition trick.



- **Example:**
  - (2 + 14) mod 12 ≡ ?
  - It can be calculated as:
    - 16 mod 12 ≡ 4

      **OR**
    - (2 mod 12 + 14 mod 12) mod 12

      ≡ ( 2 + 2 ) mod 12 ≡ 4

# Modular Arithmetic Trick

Public Key Crypto

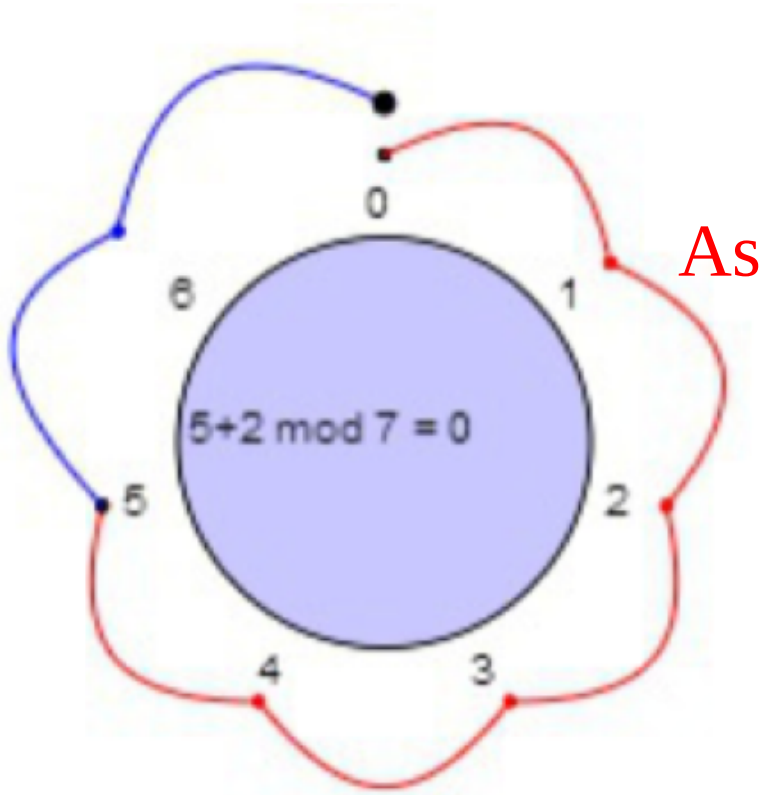
- **( c + d ) mod n ≡ ( c mod n + d mod n ) mod n**
  - add then mod
  - Is equivalent to,
  - mod each, then add and mod
- **Another Example:**
  - (12 + 9) mod 7 ≡ ?
  - It can be calculated as:
    - 21 mod 7 ≡ 0

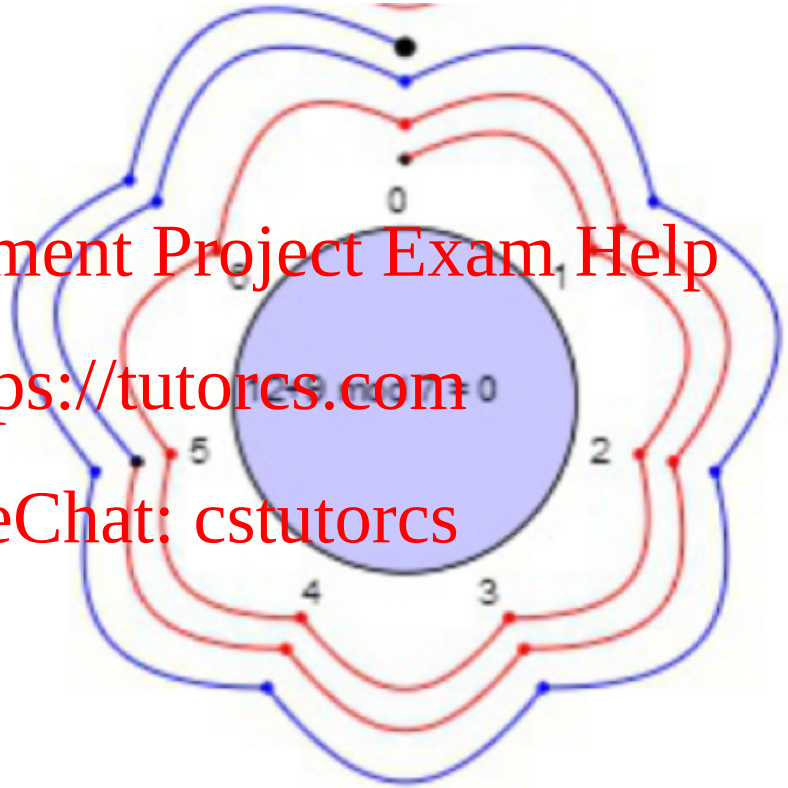
      **OR**
    - (12 mod 7 + 9 mod 7) mod 7

      ≡ ( 5 + 2 ) mod 7 ≡ 0

# Modular Arithmetic

## Public Key Crypto



# Modular Subtraction

Public Key Crypto

- (c – d) mod n ≡ ( c mod n + (–d) mod n ) mod n
  - minus then mod,
  - Is equivalent to
  - mod each then minus and mod
- **Example:**
  - (2 – 18) mod 12 ≡ ?
- Can be calculated as
  - –16 mod 12 ≡ – 4

**OR**

  - (2 mod 12 + (–18) mod 12) mod 12

    ≡ (2 + (–6)) mod 12 ≡ –4

Q: What is 42 - 88 mod 15? Use the mod subtraction trick.

# Modular Multiplication

Public Key Crypto

- **Example:**

  (5 * 14) mod 12 ≡ ?

  Can be calculated as:

  70 mod 12 ≡ 10

  OR

  (5 mod 12 * 14 mod 12) mod 12

  ≡ (5 * 2) mod 12 ≡ 10



**Q:** What is 42 x 77 mod 15?
Use the mod multiply trick.

# Multiplicative Inverse mod n

Public Key Crypto

- Multiplicative inverse of integer a -- regular (not modular) arithmetic:
  - A number b, denoted by $a^{-1}$ (or 1/a) such that a x b = 1
  - Example: a = 5, b = $5^{-1}$ = 1/5 = 0.2
  - $a^{-1}$ is a fraction (not an integer)



- Multiplicative inverse of integer a -- modular arithmetic mod n:
  - Similar, but $a^{-1}$ mod n is an integer b < n
  - Multiplicative inverse of integer a mod n
    - Number b, denoted by $a^{-1}$ mod n, such that a x b mod n = 1
    - Examples:
    - a = 5, n = 7: b = $5^{-1}$ mod 7 = 3
      - check: 5 x 3 mod 7 = 15 mod 7 = 1
    - a = 2, n = 9: b = $2^{-1}$ mod 9 = 5
      - check: 2 x 5 mod 9 = 10 mod 9 = 1

**Q:** What is $3^{-1}$ mod 11?

# Multiplicative Inverse mod n

Public Key Crypto

- How to divide mod n with modular inverse
  - In normal (not modular) arithmetic, $a \div b = a \times b^{-1}$
    - Examples
      - $2 \div 5 = 2 \times 5^{-1} = 2 \times 0.2 = 0.4$
      - $8 \div 2 = 8 \times 2^{-1} = 8 \times 0.5 = 4$
  - In modular arithmetic mod n, $a \div b \bmod n = a \times b^{-1} \bmod n$
    - Examples
      - $2 \div 5 \bmod 7 = 2 \times 5^{-1} \bmod 7 = 2 \times 3 \bmod 7 = 6$
      - (check: $6 \times 5 \bmod 7 = 30 \bmod 7 = 2$)
      - $8 \div 2 \bmod 9 = 8 \times 2^{-1} \bmod 9 = 8 \times 5 \bmod 9 = 40 \bmod 9 = 4$
      - (check: $4 \times 2 \bmod 9 = 8 \bmod 9 = 8$).

# When does no multiplicative inverse exist?

Public Key Crypto

In normal (not mod arithmetic),

- Inverse of 0 (1/0 = $0^{-1}$) does not exist
  - cannot divide by 0!

→ There is no number b such that b x 0 = 1



In modular arithmetic mod n,

- $a^{-1}$ mod n does not exist if a has common factor > 1 with modulus n (i.e if a not relatively prime to n)
  - Example:
    - n = 12, a = 4: GCD(4,12) = 4 > 1.
    - So $4^{-1}$ mod 12 does not exist!

# Modular Division vs Multiplicative Inverse

Public Key Crypto

- **Theorem**: Existence of Multiplicative Inverse of d mod n

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

  when d and n are **relatively prime**,

  d **has** a multiplicative inverse mod n, usually denoted by $d^{-1}$



**Fact:** If d and n are relatively prime, there's an efficient algorithm – Extended Euclid's algorithm (we won't study it) -- to compute $d^{-1} \bmod n$

# Integer Exponentiation

## Public Key Crypto



# Modular Exponentiation: Square & Multiply Algorithm

Public Key Crypto

**Square & Multiply Algorithm: $a^e$ mod n, e.g. $7^5$ mod 11 (a = 7, e = 5, n = 11)**

- Decompose exponent in binary representation: $e = 5_{10} = 101_2$
- Start from Most Significant (leftmost) bit of e (always 1), set z = a.
- For each subsequent bit $b_i$ of e:
  - If bit $b_i = 0$, just square z, i.e. set $z = z^2 \bmod n$
  - If bit $b_i = 1$, square z and multiply it by a, i.e. set $z = z^2 \times a \bmod n$
- $7^5$ mod 11 example:
  - Start with MS bit $b_2$=1 of $e = 101_2$ , set z = a = 7.
  - I = 2: bit $b_1 = 0$ → square: $z = z^2 \bmod n = 7^2 \bmod 11 = 5$
  - i = 1: bit $b_0 = 1$ → square & multiply: $z = z^2 \times a \bmod 11 = 5^2 \times 7 \bmod 11 = 3 \times 7 \bmod 11 = 10$

# Modular Exponentiation: Square & Multiply Algorithm

Public Key Crypto

**More Examples**

- $4^{12} \bmod 13$:
  - Start with MS bit $b_3$=1 of $e = 12_{10} = 1100_2$ , set $z = a = 4$.
  - i = 2: bit $b_2 = 1$ → square & multiply: $z = z^2 \times a \bmod n = 4^2 \times 4 \bmod 13 = 12$
  - i = 1: bit $b_1 = 0$ → square: $z = z^2 \bmod n = 12^2 \bmod 13 = 1$
  - i = 0: bit $b_0 = 0$ → square: $z = z^2 \bmod n = 1^2 \bmod 13 = 1$. → So $4^{12} \bmod 13 = 1$
- $5^{11} \bmod 7$:
  - Start with MS bit $b_3$=1 of $e = 11_{10} = 1011_2$ , set $z = a = 5$.
  - i = 2: bit $b_2 = 0$ → square: $z = z^2 \bmod n = 5^2 \bmod 7 = 4$
  - i = 1: bit $b_1 = 1$ → square & multiply: $z = z^2 \times a \bmod n = 4^2 \times 5 \bmod 7 = 3$
  - i = 0: bit $b_0 = 1$ → square & multiply: $z = z^2 \times a \bmod 7 = 3^2 \times 5 \bmod 7 = 3$. → So $5^{11} \bmod 7 = 3$

# Discrete Log Problem (DLP)

Public Key Crypto

- Discrete Logarithm: the reverse operation of modular exponentiation
- Infeasible to solve with known algorthims



- **DLP**: Given **$b = a^e \bmod n$, $a$ & $n$**, compute exponent **$e$**



- Known algorithms: inefficient, infeasible



  - brute force: time T ~ exponential in len(n)
  - DLP Number Field Sieve: time is sub-exponential in len(n)
- **Design Strategy** for Public-Key Encryption, crypto
  - do encryption/etc so that attacking requires solving DLP
  - used for security guarantees of Diffie-Hellman, ElGamal, ...

# Euler's Totient function(n) … for RSA

Public Key Crypto

- **Example: n = 10**
  - Residues = numbers < n: {0,1,2,3,4,5,6,7,8,9}
  - Residues relatively prime to n: {1,3,7,9}
- Euler's Totient function **ø(n) = number of** residues relatively prime (i.e. GCD =1) to *n*
  - **ø(10) = 4**

# Euler's Totient function ø(n) for prime *n*

Public Key Crypto

- If ***n*** is a prime number, then $ø(n) = n - 1$



  - Example:



    - $n=3$, $ø(3) = 3-1 = 2$, i.e. the numbers {1,2}
    - $n=5$, $ø(5) = 5-1 = 4$, i.e. the numbers {1,2,3,4}

# ø(n) for **product** of **two primes** p & q

Public Key Crypto

- If *n* is product of prime numbers ***p,q*** then
  - ø(*n*) = (p–1)(q–1)

**Q:** What is ø(*15*)?



- **Example:**
  - ø(*n*) = ø(*p***q*) = (3–1)*(7–1) = 12 i.e. the numbers {1,2,4,5,8,10,11,13,16,17,19,20}

# Euler's Theorem & Fermat's Little Theorem

Public Key Crypto

- **Euler's Theorem:** If **M** and ***n*** are **relatively prime**, then
  - $M^{\phi(n)} \bmod n = 1$
  - so $M^{\phi(n)+1} \bmod n = M$ *(used for RSA)*



- ***Fermat's Little Theorem:*** If *p* is a prime number and **M** is **relatively prime to *p***, then
  - $M^{\phi(n)} \bmod p$

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

# Euler's Theorem: Examples

Public Key Crypto

## Examples:

**Euler's Theorem:** $M^{ø(n)} \bmod n = 1$



- n = 21, $M_1$ = 5, $M_2$ = 8



$ø(21) = (3-1) \times (7-1) = 2 \times 6 = 12$ (because $21 = 3 \times 7$);



So by Euler's Theorem:

- $M_1^{ø(n)} \bmod n = 5^{ø(21)} \bmod 21 = 5^{12} \bmod 21 = 1.$
- $M_1^{ø(n)} \bmod n = 8^{ø(21)} \bmod 21 = 8^{12} \bmod 21 = 1.$

# Fermat's Little Theorem: Examples

Public key Crypto

**Examples:**

**Fermat's Little Theorem:** $M^{p-1} \bmod p = 1$



p = 7 , $M_1 = 2$, $M_2 = 3$:



- $M_1^{\phi(n)} \bmod p = 2^6 \bmod 7 = 64 \bmod 7 = 1$
- $M_2^{\phi(n)} \bmod p = 3^6 \bmod 7 = 27 \times 27 \bmod 7 = 6 \times 6 \bmod 7 = 1$

# Euler's Theorem: *for RSA decryption*

Public Key Crypto

- Recall again Euler's Theorem: If M and *n* are **relatively prime**, then
  - $M^{\phi(n)} \bmod n = 1$



**Implication of Euler's Theorem:**

- If $y \bmod \phi(n) = 1$ **then** $M^y \bmod n = M$

- **Reason:**
  - Since $y = \phi(n) \times k + 1$ for some integer $k$ then
  - $M^y \bmod n = M^{\phi(n) \times k + 1} \bmod n$

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

# Implication of Euler's Theorem: Examples

Public key Crypto

**Examples:**

**Implication of Euler's Theorem**: $M^y \bmod n = M$ if $y \bmod \phi(n) = 1$



$n = 21, M_1 = 5, M_2 = 8, y = 13$:



$y \bmod \phi(n) = 13 \bmod 12 = 1$ so:



- $M_1^y \bmod n = 5^{13} \bmod 21 = 5$.
- $M_2^y \bmod n = 8^{13} \bmod 21 = 8$.

# How to Find Large Prime Numbers?

Public Key Crypto

- For PKE, need to **efficiently** generate large random prime numbers

- **Q1: how to check if p is prime, efficiently?**
  - brute-force primality check: divide *p* by all integers up p

    T ~ exponential – inefficient



  - **Fact (Miller-Rabin Primality Testing)**: There is an **efficient** algorithm for primality testing (no trial division)

    - Idea of Miller-Rabin (we won't study): use Fermat's Little Theorem for p
      - choose many random a, check if $a^{p-1} \bmod p = 1$

# How to Find Large Prime Numbers?

Public Key Crypto

- For PKE, need to **efficiently** generate large random prime numbers

- **Q2: If we choose a random L-bit integer p, what's the chance that p is prime?**
  - If chance small, need to repeat primality test on too many random integers until find a prime integer
  - Luckily, chance is not too small:



  - **Theorem (Prime Number Theorem):** For large L, the fraction of integers of bit length up to L that are prime, is close to 1 / (0.693*L)

    - Example: On **average**, we should find a prime after testing about 693 random 1000-bit integers

# How to Find Large Prime Numbers?

Public Key Crypto

- **Conclusion (Q1+Q2):**
  - It's possible to **efficiently generate** large random prime integers
    - Just keep picking random integers and test them for primality until a prime is found

# Summary: Feasible and Infeasible Problems

Public Key Crypto

| | Feasible Problems (efficient algs.)<br>(Used in PKE by honest parties) | Infeasible Problems (inefficient algs.)<br>(for PKE security against attacks) |
|---|---|---|
| | Integer Multiplication | Integer Factorisation Problem (IFP) |
| | Modular Exponentiation | Discrete Logarithm Problem (DLP) |
| | Greatest Common Divisor (GCD) | |
| | Multiplicative inverse mod n | |
| | Finding large random primes | |

# Further Reading

## Public Key Crypto

- Section 2.3 of the textbook: *Computer Security: Principles and Practice" by William Stallings & Lawrie Brown,* Prentice Hall, 2015 [Pub key Encryption concept]
- Optional deeper reading for interested students:
  - Sections 8.1.1, 8.1.2 and 8.2.1 of the textbook: *Introduction to Modern Cryptography*, by Jonathan Katz and Yehuda Lindell, Chapman & Hall/CRC, 2014.