

CSE 3400 - Introduction to Computer & Network Security  
(aka: Introduction to Cybersecurity)

Lecture 10  
Public Key Cryptography – Part I

Ghada Almashaqbeh  
UConn

From Textbook Slides by Prof. Amir Herzberg  
UConn

---

# Outline

- ❑ Number theory review.
- ❑ Intro to public key cryptography.
- ❑ Key exchange.
- ❑ Hardness assumptions: DL, CDH, DDH.



# Number Theory Review

---

# Our Focus

- A brief overview of mainly modular arithmetic.
- The minimalist set we need in topics covered in this course.

# The Modulo Operation

**Definition 1.2** (The modulo operation). *Let  $a, m \in \mathbb{Z}$  be integers such that  $m > 0$ . We say that an integer  $r$  is a residue of  $a$  modulo  $m$  if  $0 \leq r < m$  and  $(\exists i \in \mathbb{Z})(a = r + i \cdot m)$ . For any given  $a, m \in \mathbb{Z}$ , there is exactly one such residue of  $a$  modulo  $m$ ; we denote it by  $a \bmod m$ .*

Properties (make it easier to compute complex modular arithmetic expressions):

$$(a + b) \bmod m = [(a \bmod m) + (b \bmod m)] \bmod m \quad (1.2)$$

$$(a - b) \bmod m = [(a \bmod m) - (b \bmod m)] \bmod m \quad (1.3)$$

$$a \cdot b \bmod m = [(a \bmod m) \cdot (b \bmod m)] \bmod m \quad (1.4)$$

$$a^b \bmod m = (a \bmod m)^b \bmod m \quad (1.5)$$

# The Modulo Operation

Properties (extends also to polynomials):

Similar properties hold for any polynomial  $p(x)$  with integer coefficients and input ( $x \in \mathbb{Z}$ ), as well as for a polynomial  $p(x_1, x_2, \dots)$  with integer coefficients and multiple integer parameters ( $x_1, x_2, \dots \in \mathbb{Z}$ ):

$$[p(x)] \bmod m = [p(x \bmod m)] \bmod m \quad (1.6)$$

$$[p(x_1, x_2, \dots)] \bmod m = p(x_1 \bmod m, x_2, \dots) \bmod m = \quad (1.7)$$

$$= p(x_1 \bmod m, \dots) \bmod m \quad (1.8)$$

# Examples

- $7 \bmod 9 = ?$
- $13 \bmod 8 = ?$
- $0 \bmod 11 = ?$
- $4 \bmod 4 = ?$
- $(30 + 66) \bmod 11 = ?$
- How about:  $445 \cdot (81 \cdot 34^{13} + 83 \cdot 33^{345}) \bmod 4$  ?

Denote  $445 \cdot (81 \cdot 34^{13} + 83 \cdot 33^{345}) \bmod 4$  by  $x$ . Then we find  $x$  as follows:

$$\begin{aligned}x &= 445 \cdot (81 \cdot 34^{13} + 83 \cdot 33^{345}) \bmod 4 \\&= (445 \bmod 4) \cdot ((81 \bmod 4) \cdot (34 \bmod 4)^{13} + \\&\quad + (83 \bmod 4) \cdot (33 \bmod 4)^{345}) \bmod 4 \\&= 1 \cdot (1 \cdot 2^{13} + 3 \cdot 1^{345}) \bmod 4 \\&= (2 \cdot 4^6 + 3) \bmod 4 \\&= 3 \bmod 4 = 3\end{aligned}$$



# Multiplicative Inverse

- Needed to support division in modular arithmetic.
  - Division not always produce integers.
  - Modular arithmetic requires integers to work with!!
- To compute  $a/b \text{ mod } m$ , multiply a by the multiplicative inverse of  $b$ .
  - That is compute  $ab \text{ mod } m = ab^{-1} \text{ mod } m$ .
  - Where  $b^{-1}$  is the multiplicative inverse such that  $bb^{-1} \text{ mod } m = 1$
- Not all integers have multiplicative inverses with respect to a specific modulus m.

# Multiplicative Inverse

If and only if  $x$  and  $m$  are a coprime.

**Definition 1.3.** Let  $x, m \in \mathbb{Z}$  be integers such that  $m > 0$  and  $x \bmod m \neq 0$ . Then there is a unique integer  $x^{-1}$  such that  $x \cdot x^{-1} \bmod m = 1$  and  $m > x^{-1} > 0$ . We say that  $x^{-1}$  is the multiplicative inverse of  $x$  modulo  $m$ .

## □ Examples:

- $3/5 \bmod 4 = 3 \cdot 5^{-1} \bmod 4 = ?$
- $3/5 \bmod 6 = 3 \cdot 5^{-1} \bmod 6 = ?$

**Fact 1.2.** Let  $x, m \in \mathbb{Z}$  be integers such that  $m > 0$  and  $x \bmod m \neq 0$ . Then there is an efficient algorithm that finds  $x^{-1} \bmod m$ .

- Such an algorithm is called the Extended Euclidean algorithm (out of scope for this course).



# Modular Exponentiation

- Will be encountered a lot; discrete log based scheme, RSA, etc.
- We have seen a property to reduce the base, but how about the exponent?
  - Its reduction will be with respect to a different modulus than the one in the original operation.
- Fermat's Little Theorem:

**Theorem 1.1.** *For any integers  $a, b, p \in \mathbb{Z}$ , if  $p$  is a prime and  $p > 0$ , then*

$$\begin{aligned} a^b \mod p &= a^{b \mod (p-1)} \mod p \\ &= (a \mod p)^{b \mod (p-1)} \mod p \end{aligned} \tag{1.9}$$

# Modular Exponentiation

- Examples; Use Fermat's Little theorem (if applicable) to solve the following:
  - $13^{32} \bmod 31 = ?$
  - $19^{930} \bmod 4 = ?$
  - $19^{60} \bmod 7 = ?$
- Can we reduce the exponent for non prime (composite) modulus?
  - We can use Euler's Theorem.



# Euler's Function

- Called also Euler's Totient function. For every integer  $n \geq 1$ , this function computes the number of positive integers that are less than or equal to  $n$  and co-prime to  $n$ .

$$\phi(n) = |\{i \in \mathbb{N} : i \leq n \wedge \gcd(i, n) = 1\}|$$

Examples:

$n$	1	2	3	4	5	6	7	8	9	10
$\phi(n)$	1	1	2	2	4	2	6	4	6	4
factors?	none	none	none	$2 \cdot 2$	none	$2 \cdot 3$	none	$2^3$	$3 \cdot 3$	$2 \cdot 5$

# Euler's Function Properties

**Lemma 1.1.** *For any prime  $p > 1$  holds  $\phi(p) = p - 1$ . For prime  $q > 1$  s.t.  $q \neq p$  holds  $\phi(p \cdot q) = (p - 1)(q - 1)$ .*

**Lemma 1.2** (Euler function multiplicative property). *If  $a$  and  $b$  are co-prime positive integers, then  $\phi(a \cdot b) = \phi(a) \cdot \phi(b)$ .*

**Lemma 1.3.** *For any prime  $p$  and integer  $l > 0$  holds  $\phi(p^l) = p^l - p^{l-1}$ .*

**Lemma 1.4.** *Let  $n = \prod_{i=1}^n (p_i^{l_i})$ , where  $\{p_i\}$  is a set of distinct primes (all different), and  $l_i$  is a set of positive integers (exponents of the different primes). Then:*

$$\phi(n) = \phi\left(\prod_{i=1}^n (p_i^{l_i})\right) = \prod_{i=1}^n \left(p_i^{l_i} - p_i^{l_i-1}\right) \quad (1.12)$$

# Euler's Theorem

**Theorem 1.2** (Euler's theorem). *For any co-prime integers  $m, n$  holds  $m^{\phi(n)} = 1 \pmod{n}$ . Furthermore, for any integer  $l$  holds:*

$$m^l \pmod{n} = m^{l \pmod{\phi(n)}} \pmod{n} \quad (1.19)$$

## □ Examples:

- $13^{31} \pmod{31} = ?$
- $27^{26} \pmod{10} = ?$

# Last Stop

- **Congruence:**  $a \equiv b \pmod{m}$ 
  - Used when two expressions have the same residue with respect to some modulus.
  - It is an equivalence relation, so it satisfies:
    - Reflexivity:**  $a \equiv a \pmod{m}$ .
    - Symmetry:**  $a \equiv b \pmod{m}$  if  $b \equiv a \pmod{m}$ .
    - Transitivity:** if  $a \equiv b \pmod{m}$  and  $b \equiv c \pmod{m}$  then  $a \equiv c \pmod{m}$ .
- Lastly, we have the fundamental theorem of arithmetic.

**Theorem 1.3** (The fundamental theorem of arithmetic). *Every number  $n > 1$  has a unique representation as a product of powers of distinct primes.*

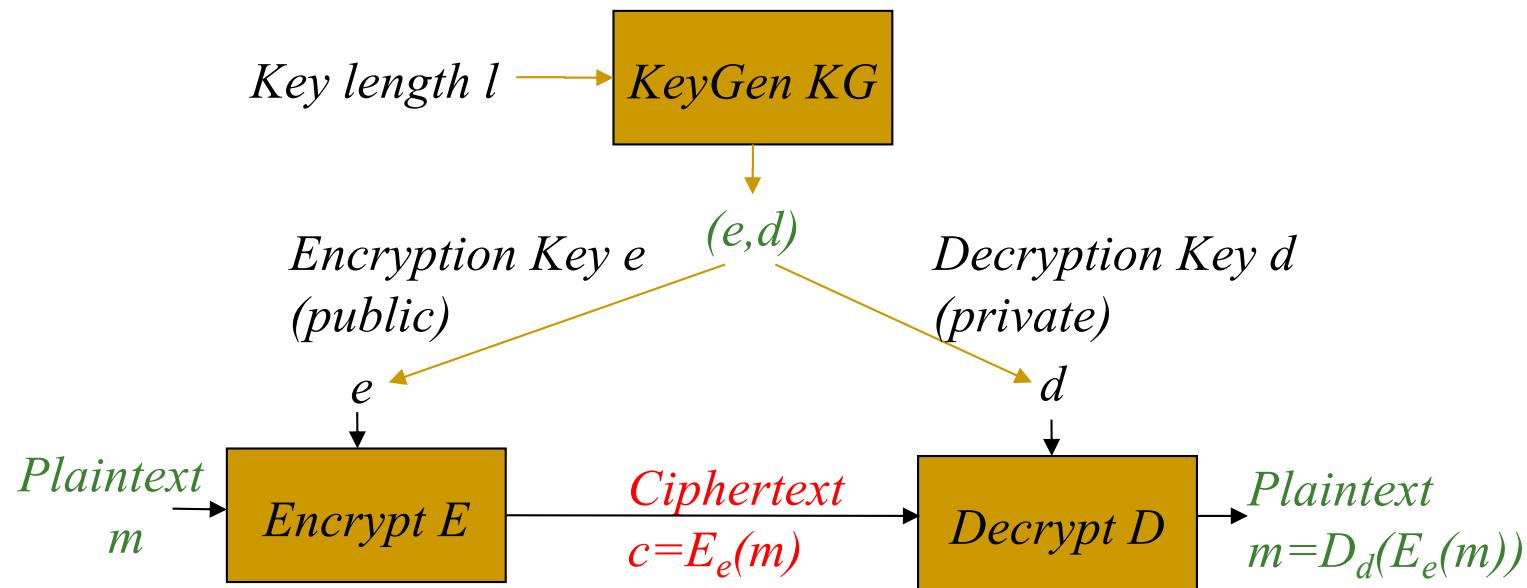
# Intro to Public Key Cryptography

# Public Key Cryptology

- Kerckhoff: cryptosystem (algorithm) is public
- What we learned until now:
  - Only the key is secret (unknown to attacker)
  - Same key for encryption, decryption  
→ if you can encrypt, you can also decrypt!
- But can we give encryption capability without a decryption capability?
  - Yes, using public key cryptography!

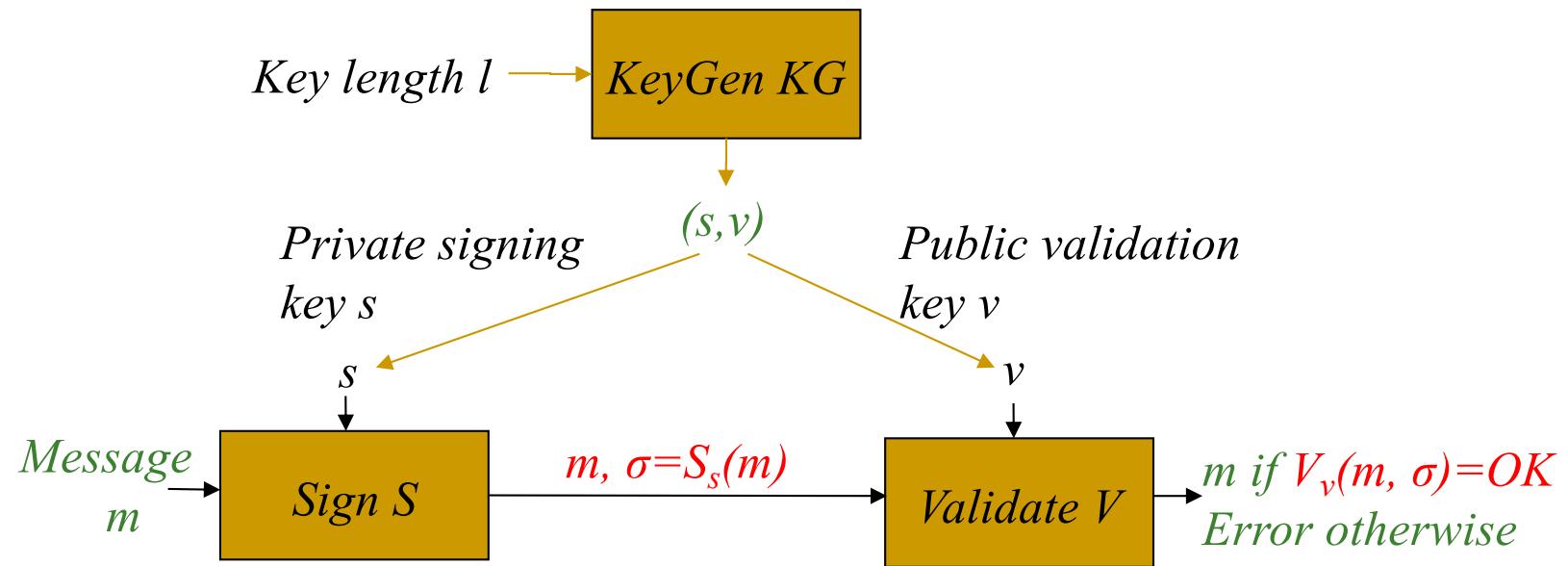
# Public Key Cryptosystem (PKC)

- Kerckhoff: cryptosystem (algorithm) is public
- [DH76]: can encryption key be public, too??
  - Decryption key will be different (and private)
  - Everybody can send me mail, only I can read it.



# Is it Only About Encryption?

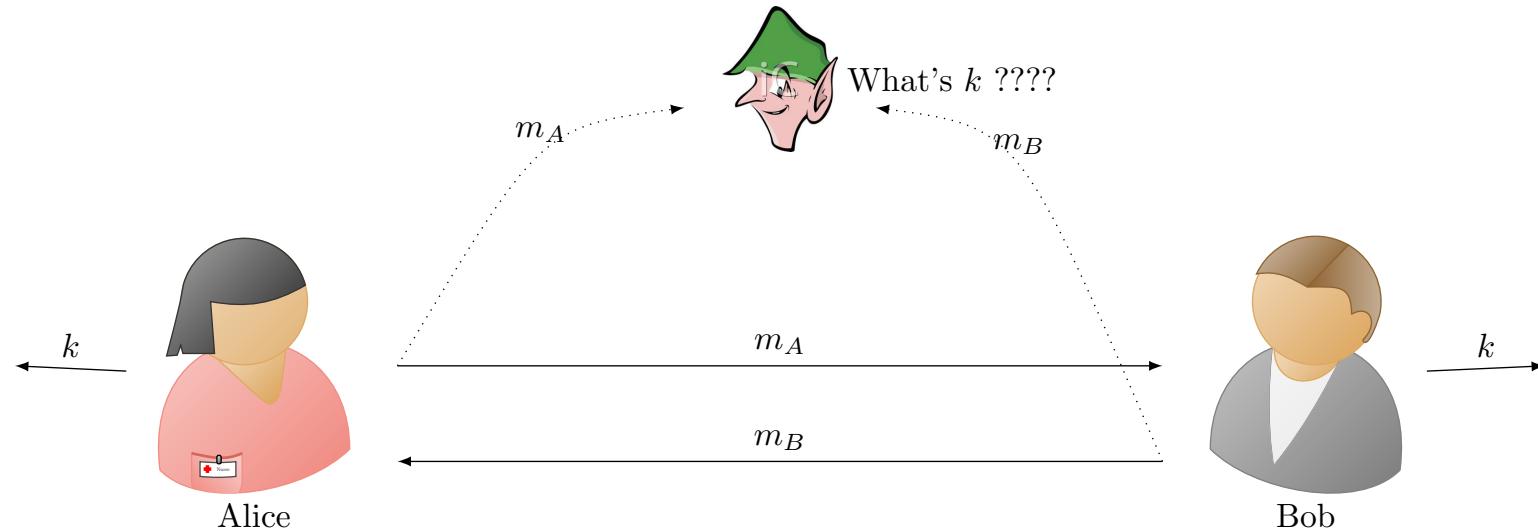
- Also: Digital signatures (RSA, DSA,...)
  - Sign with private key  $s$ , verify with public key  $v$
  - (Recall MACs; a shared key cryptosystem for message authentication).



# More: Key-Exchange Protocol

## ■ Key Exchange Protocols

- ❑ Establish shared key between Alice and Bob **without** assuming an existing shared ('master') key !!
- ❑ Use public information from A and B to setup shared secret key  $k$ .
- ❑ Eavesdropper cannot learn the key  $k$ .



# Public keys solve more problems...

- Signatures provide **evidences**
  - Everyone can validate, only ‘owner’ can sign
- Establish shared secret keys
  - Use authenticated public keys
    - Signed by trusted certificate authority (CA)
  - Or: use DH key exchange
- Stronger resiliency to key exposure
  - Perfect forward secrecy and recover security
    - Protect confidentiality from possible key exposures
  - Threshold (and proactive) security
    - Resilient to exposure of  $k$  out of  $n$  parties (every period)

# Public keys are easier...

- To distribute:
  - From directory or from incoming message (still need to be authenticated)
  - Less keys to distribute (same public key to all)
- To maintain:
  - Can keep in non-secure storage as long as being validated (e.g. using MAC) before using
  - Less keys:  $O(|parties|)$ , not  $O(|parties|^2)$
- So: why not **always** use public key crypto?

# The Price of PKC

## ■ Assumptions

- Applied PKC algorithms are based on a small number of specific computational assumptions
  - Mainly: hardness of factoring and discrete-log
- Both may fail against quantum computers

## ■ Overhead

- Computational
- Key length
- Output length (ciphertext/signature)

# Public key crypto is harder...

- Requires related public, private keys
  - Private key `reverses` public key
  - Public key does not expose private key
- Substantial overhead
  - Successful cryptanalytic shortcuts → need long keys
  - Elliptic Curves (EC) may allow shorter key (almost no shortcuts found)
  - Complex computations
  - RSA: very complex (slow) key generation
- Most: based on hard modular math problems

[LV02]	Required key size		
Year	AES	RSA, DH	EC
2010	78	1369	160
2020	86	1881	161
2030	93	2493	176
2040	101	3214	191

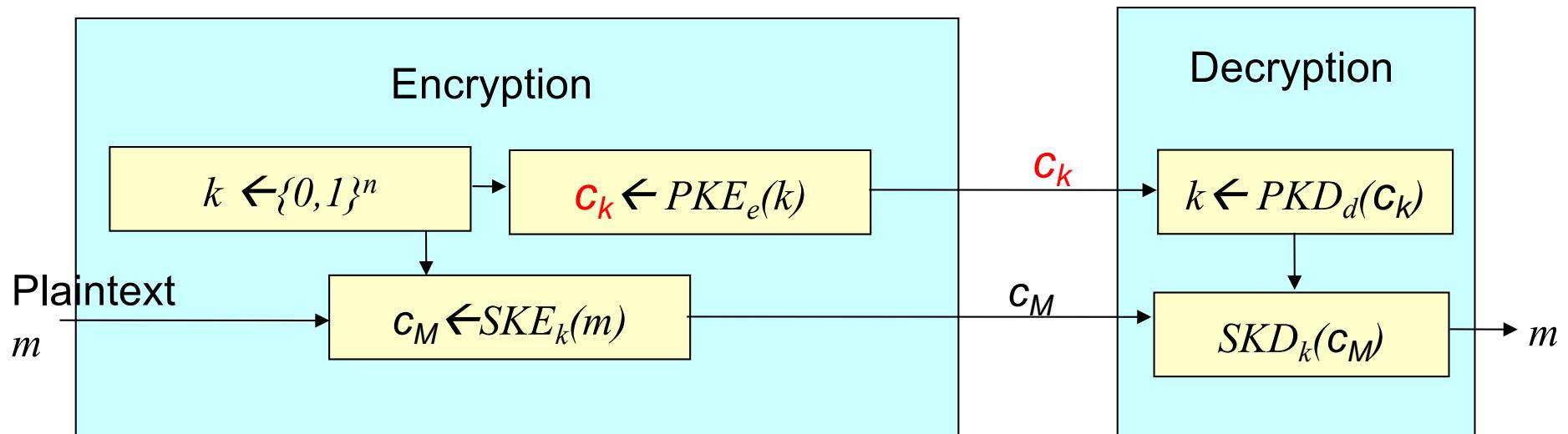
Commercial-grade security  
Lenstra & Verheul [LV02]

# In Sum

- Minimize the use of PKC
- In particular: apply PKC only to **short inputs**
- How ??
  - For signatures:
    - **Hash-then-sign**
  - For public-key encryption:
    - **Hybrid encryption**

# Hybrid Encryption ('enveloping')

- Challenge: public key cryptosystems are slow
- Hybrid encryption:
  - Use a shared key encryption scheme to encrypt all messages.
  - But use a public key encryption system to exchange the shared key (Alice generates the  $k$ , encrypt it under Bob's public key and send it to Bob, Bob can then recover this key).



# Hard Modular Math Problems

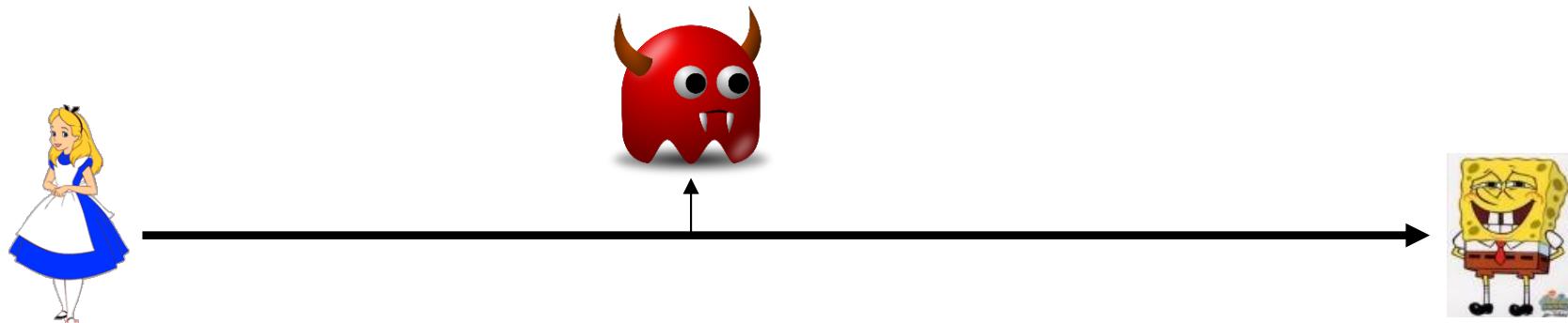
- No efficient solution, in spite of extensive efforts
  - But: **verification** of solutions is easy ('one-way' hardness)
    - Discrete log: exponentiation
- **Problem 1: Factoring**
  - Choose randomly  $p, q \in_R \text{LargePrimes}$
  - Given  $n = pq$ , it is infeasible to find  $p, q$
  - Verification? Easy, just multiply factors
  - Basis for the RSA cryptosystem and many other tools
- **Problem 2: Discrete logarithm in cyclic group  $\mathbb{Z}_p^*$** 
  - Where  $p$  is a safe prime [details in textbook]
  - Given random number, find its (discrete) logarithm
  - Verification is efficient by exponentiation:  $O((\lg n)^3)$
  - Basis for the Diffie-Hellman Key Exchange and many other tools
  - We first discuss key-Exchange problem, then [DH] and disc-log

# Key Exchange

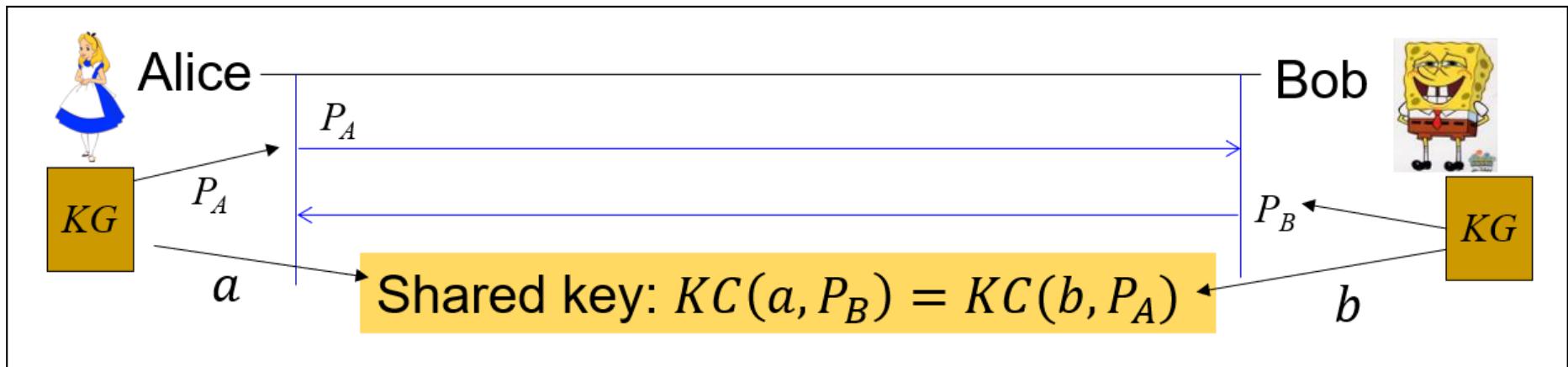
# The Key Exchange Problem

Aka key agreement

- Alice and Bob want to agree on secret (key)
  - Secure against **eavesdropper** adversary
  - Assume no prior shared secrets (key)
    - Otherwise seems trivial
    - Actually, we'll later show it's also useful in this case...



# Defining a Key Exchange Protocol



Must satisfy correctness; both parties compute the same shared key, and key indistinguishability (the key that the two parties establish is indistinguishable from random).

Discrete Log (DL) Assumption  
and  
The Computational/Decisional Diffie-  
Hellman Assumptions (CDH/DDH)  
and  
The DH Key Exchange Protocol

# The Discrete Log Problem

- Computing logarithm is quite efficient over the reals
- Consider a cyclic multiplicative group  $G$ 
  - Cyclic group: exists generator  $g$  s.t.  $(\forall a \in G)(\exists i)(a = g^i)$
  - Discrete log problem: given generator  $g$  and  $a \in G$ , find  $i$  s.t.  $a = g^i$
  - Verification: exponentiation (efficient algorithm)
  - For prime  $p$ , the group  $\mathbb{Z}_p^* = \{1, \dots, p-1\}$  is cyclic [multiplications mod  $p$ ]
- Is discrete-log hard?
  - Some ‘weak’ groups, i.e., where disc-log is **not** hard:
    - $\mathbb{Z}_p^*$  for prime  $p$ , where  $(p - 1)$  has only ‘small’ prime factors
      - Using the Pohlig-Hellman algorithm
    - Check!! Mistakes/trapdoors found, e.g., in OpenSSL’16
  - Other groups studied, considered Ok (‘hard’)
  - **Safe-prime** groups:  $\mathbb{Z}_p^*$  for **safe prime**:  $p = 2q + 1$  for prime  $q$

# Discrete Log Assumption

[for safe prime group:  $p = 2q + 1$  for prime  $q$ ]

Given PPT adversary  $A$ , and  $n$ -bit safe prime  $p$ :

$$\Pr \left[ \begin{array}{l} g \leftarrow \text{Generator}(Z_p^*); \\ x \stackrel{\$}{\leftarrow} Z_p^* \\ A(x) = a \mid x = g^a \text{ mod } p \end{array} \right] \approx \text{negl}(n)$$

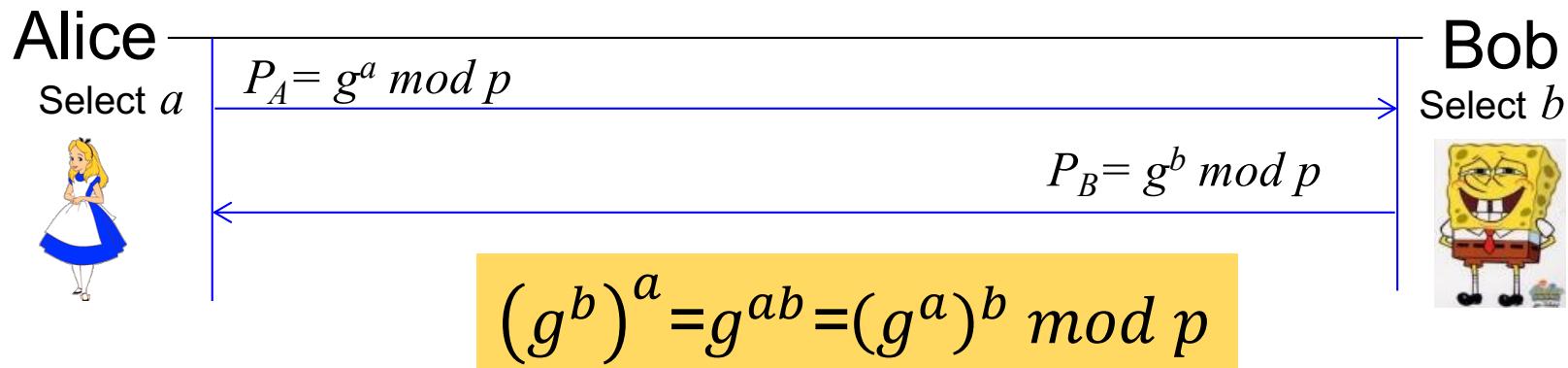
Comments:

1. Similar assumptions for (some) other groups
2. Knowing  $q$ , it is easy to find a generator  $g$
3. Any generator (primitive element) will do

# Diffie-Hellman [DH] Key Exchange

Using cyclic group  $\mathbb{Z}_p^*$

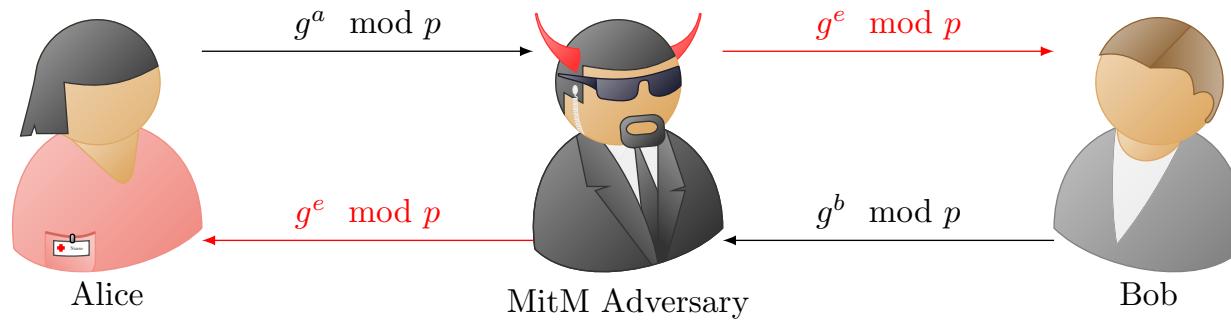
- Simplified Discrete Exponentiation Key Exchange
- Agree on a random safe prime  $p$ 
  - And generator  $g$  for the cyclic group  $\mathbb{Z}_p^*$
- Alice: secret key  $a$ , public key  $P_A = g^a \text{ mod } p$
- Bob: secret key  $b$ , public key  $P_B = g^b \text{ mod } p$
- To set up a shared key :



# Caution: Authenticate Public Keys!

- Diffie-Hellman key exchange is only secure using the authentic public keys
  - Or (equivalently): against eavesdropper
- If Bob simply receives Alice's public key, [DH] is vulnerable to 'Man in the Middle' attack

$$a \xleftarrow{\$} \{1, \dots, p\} \quad e \xleftarrow{\$} \{1, \dots, p\} \quad b \xleftarrow{\$} \{1, \dots, p\}$$



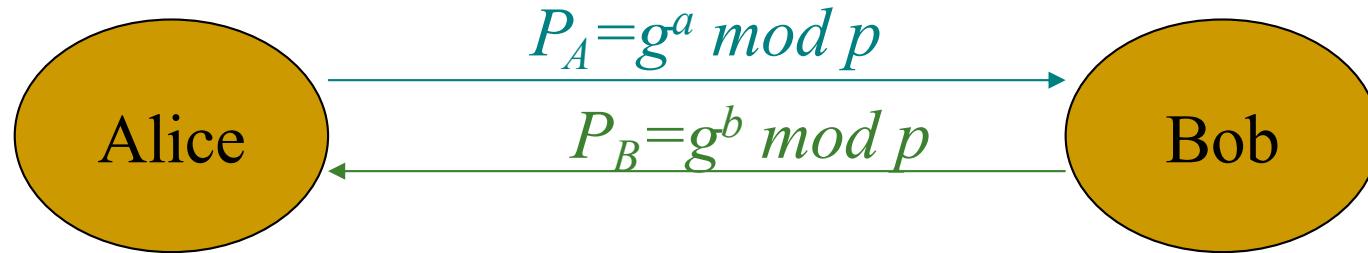
$$(g^e)^a = g^{a \cdot e} \bmod p$$

$$(g^a)^e = g^{a \cdot e} \bmod p,$$
  
$$(g^b)^e = g^{b \cdot e} \bmod p$$

$$(g^e)^b = g^{b \cdot e} \bmod p$$

# Security of [DH] Key Exchange

- Assume authenticated communication
- Based on Computational Discrete Log Assumption
- But DH requires stronger assumption than Disc-Log:
  - Maybe from  $g^b \bmod p$  and  $g^a \bmod p$ , adversary can compute  $g^{ab} \bmod p$  (without knowing/learning  $a, b$  or  $ab$ )?



# Computational DH (CDH) Assumption [for safe prime group]

Given PPT adversary A:

$$\Pr \left[ \begin{array}{l} (p, q) \leftarrow \text{primes s.t. } p = 2q + 1; \\ g \leftarrow \text{Generator}(\mathbb{Z}_p^*); \\ a, b \leftarrow \{1 \dots p - 1\}; \\ A(g^a, g^b \bmod p) = g^{ab} \bmod p \end{array} \right] \approx \text{negl}(n)$$

Assume CDH holds. Can we use  $g^{ab}$  as key?

Not necessarily; maybe finding some bits of  $g^{ab}$  is easy?

# Using DH securely?

- Consider  $\mathbb{Z}_p^*$  (multiplicative group for (safe) prime  $p$ )
- Can  $g^a, g^b$  expose *something* about  $g^{ba} \bmod p$  ?
- Bad news:
  - Finding (at least) **one bit** about  $g^{ba} \bmod p$  is easy!
  - (details in textbook if interested)
- Good news:
  - Many of the bits were shown to be as secure as the whole
  - Also, there are other stronger groups (e.g., Schnorr's) in which security holds
- So...how to use DH ‘securely’?

# Using DH securely?

- Adversary may compute *some* bits over  $g^{ba} \bmod p$
- So...how to use DH ‘securely’? Two options!
- Option 1: Use DH but with a ‘stronger’ group, e.g., Schnorr’s - **not**  $\mathbb{Z}_p^*$  (mod safe-prime  $p$ )
  - The (stronger) **Decisional DH (DDH) Assumption**: adversary can’t **distinguish** between  $[g^a, g^b, g^{ab}]$  and  $[g^a, g^b, g^c]$ , for random  $a, b, c$ .
- Option 2: use DH with safe prime  $p$ ... (*where only CDH holds*) but use a **key derivation function (KDF)** to derive a secure shared key
- Applied crypto mostly uses KDF... and we too ☺

# Using DH ‘securely’: CDH+KDF

- **Key Derivation Function (KDF)**
  - Two variants: random-keyed and unkeyed (deterministic)
- Randomized - KDF:  $k = KDF_s(g^{ab} \bmod p)$  where  $KDF$  is a key derivation function and  $s$  is public random ('salt')
- Deterministic - crypto-hash:  $k = h(g^{ab} \bmod p)$  where  $h$  is randomness-extracting crypto-hash
  - No need in salt, but **not** provably-secure

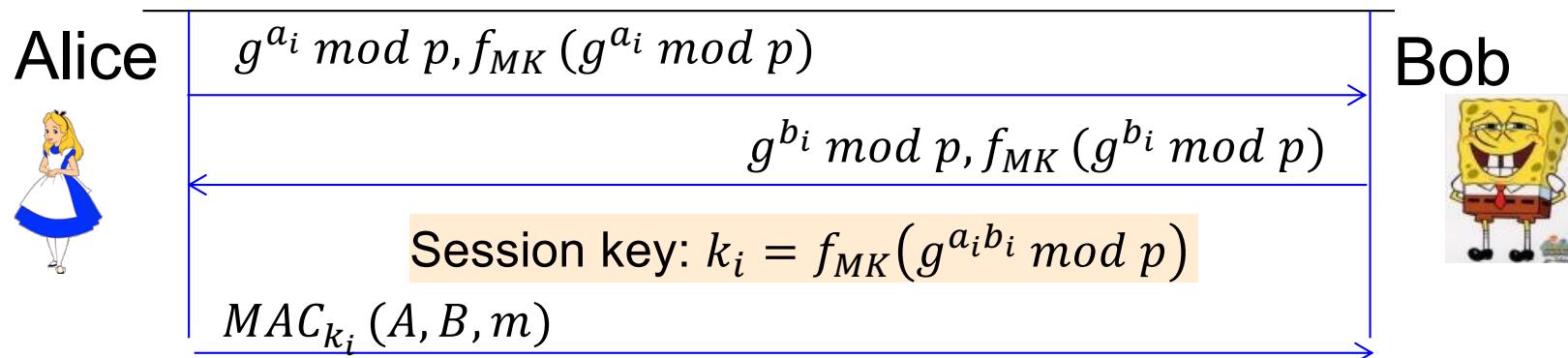
# Resilience to Key Exposure

# Authenticated DH

- Recall: DH is not secure against MitM attacker
- Use DH for **resiliency to key exposure**
  - Do authenticated DH periodically
  - Use derived key for confidentiality, authentication
    - Some protocols use key to authenticate next exchange
  - → Perfect Forward Secrecy (PFS):
    - Confidentiality of session  $i$  is resilient to exposure of all keys, except  $i$ -th session key, after session  $i$  ended

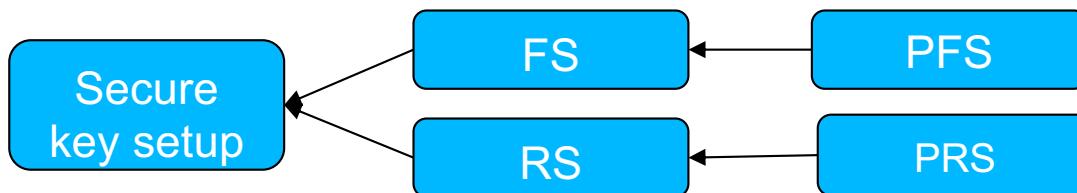
# Authenticated DH: using KDF/PRF<sub>[TLS]</sub>

- Assume  $f$  which is both a PRF and a KDF
- $MK$  is secret +  $f$  is PRF (& MAC) → authentication
  - And, assuming  $MK$  is secret, session keys are secure – even if disclosure would be easy (quantum computers or math breakthrough)
- Assuming CDH and that  $f$  is **KDF**: secure if MK exposed
  - Since most bits of  $g^{a_i b_i}$  are secret
  - Against eavesdropping adv. or if MK-exposed only after session ends.
  - Perfect forward secrecy (PFS) !



# Resiliency Notions: Shared + Public Key

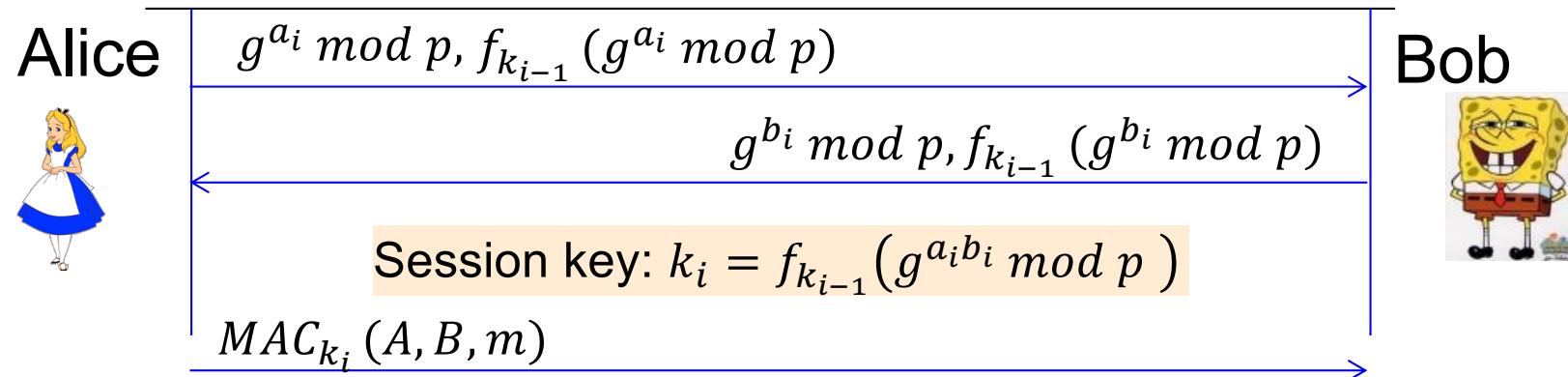
Notion	Session $i$ is secure if keys not exposed, and...	Crypto
Secure key-setup	... attacker is given <u>session key <math>k_j</math> (<math>j \neq i</math>)</u> . <b>Master key never exposed !</b>	✓ ey
Forward secrecy	... attacker is given <b>all</b> keys of sessions $> i$	✓ ey
Perfect Forward Secrecy (PFS)	... is also given all keys of sessions $< i$ , <u>but only after session <math>i</math> ended</u>	✓ ey
Recover Security	... if no attack during session $i$ , or if previous session, $i - 1$ , is secure	No!
Perfect Recover Security (PRS)	... if no <b>MitM</b> attack during session $i$ , or if previous session, $i - 1$ , is secure	Why?



Exposing master key makes all future session vulnerable to MitM

# Perfect Recover Secrecy: Ratchet DH

- Extend Auth-DH to perfect recover secrecy (PRS)
  - Idea: avoid fixed master key; use 'ratchet' of keys



- PRS: session  $i$  secure, if previous session  $(i - 1)$  secure, or if no MitM during session  $i$
- Previous key  $k_{i-1}$  was secret  $\rightarrow$  session  $i$  was authenticated  $\rightarrow$   $k_i$  is secret
  - Or: no MitM  $\rightarrow$  security due to DH

# Ratchet-DH's Exposure-Resiliency

Notion	Session $i$ is secure if keys not exposed, and...	Ratchet-DH
Secure key-setup	... attacker is given <u>session key</u> $k_j$ ( $j \neq i$ ). <b>Master key never exposed !</b>	
Forward secrecy	... attacker is given <b>all</b> keys of sessions $> i$	
Perfect Forward Secrecy (PFS)	... is also given all keys of sessions $< i$ , <u>but only after session <math>i</math> ended</u>	
Recover Secrecy	... if no attack during session $i$ , or if previous session, $i - 1$ , is secure	
Perfect Recover Secrecy (PRS)	... if no <b>MitM</b> attack during session $i$ , or if previous session, $i - 1$ , is secure	

# Covered Material From the Textbook

- Appendix A.2
- Chapter 6: sections 6.1, 6.2, and 6.3

# Thank You!

