CSE 3400/CSE 5850 - Introduction to Computer & Network Security
/ Introduction to Cybersecurity

Lecture 10
Public Key Cryptography– Part I

Ghada Almashaqbeh
UConn

*Adapted from the textbook slides

Outline

- Introduction to public key cryptography and motivation.
- Number theory review.
- The discrete log assumption.
- The Diffie-Hellman key exchange protocol.

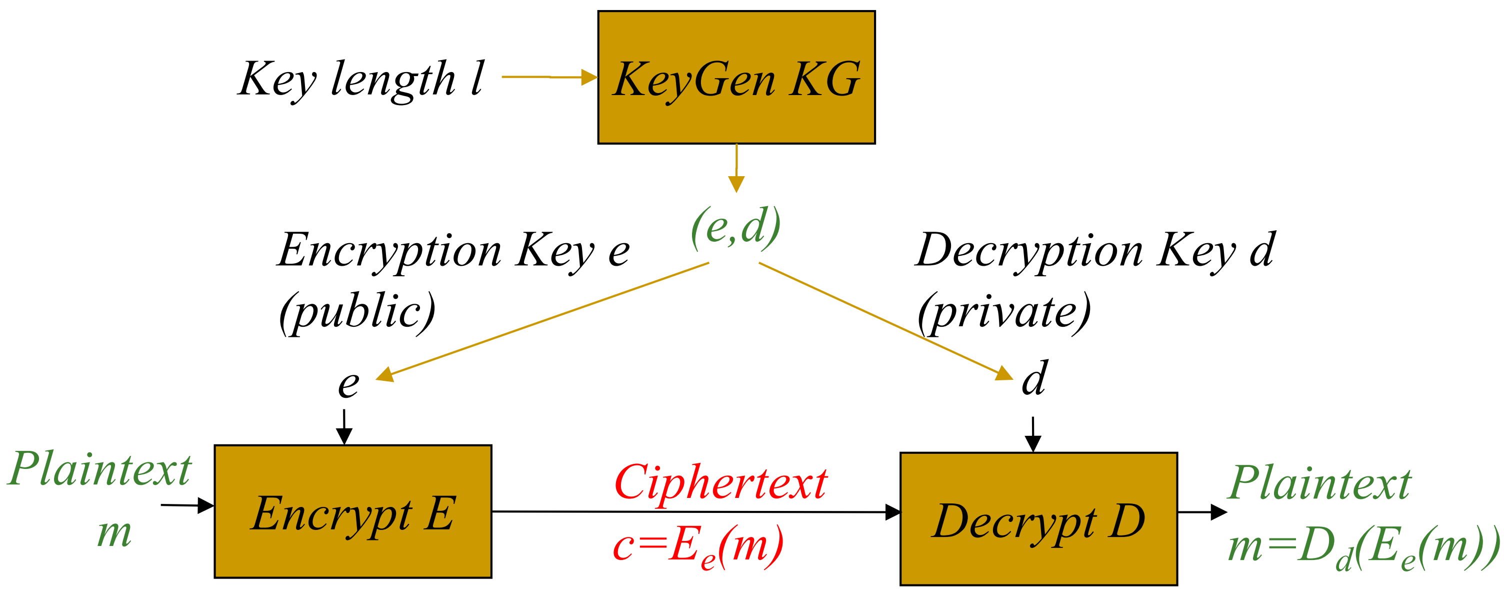
Intro to Public Key Cryptography

Public Key Cryptography

- Kerckhoff's principle: the cryptosystem (algorithm) is public
- What we learned until now: ***symmetric or shared key*** setting
  - Only the key is secret (unknown to attacker)
  - Same key for encryption and decryption ➔ if you can encrypt, you can also decrypt!
  - Shared keys for MACs and PRFs, etc.
- But can we give ***asymmetric*** cryptographic capability, e.g., encryption capability without a decryption capability?
  - Yes, using public key cryptography!

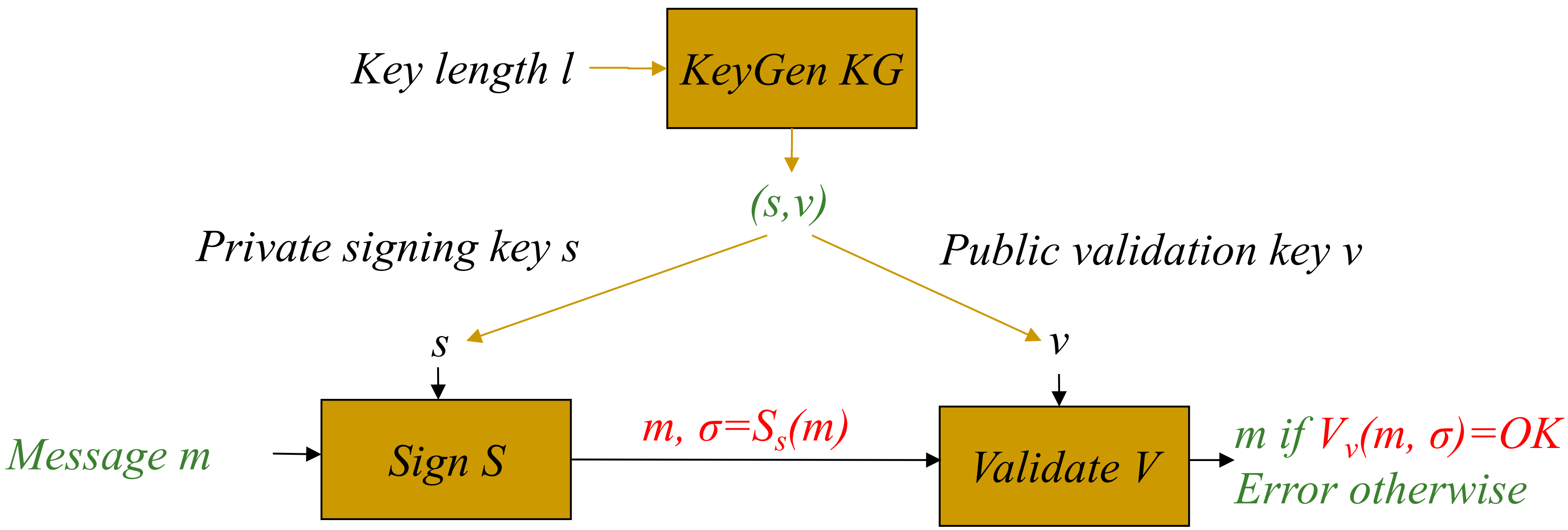
Public Key Cryptosystem (PKC)

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



Is it Only About Encryption?

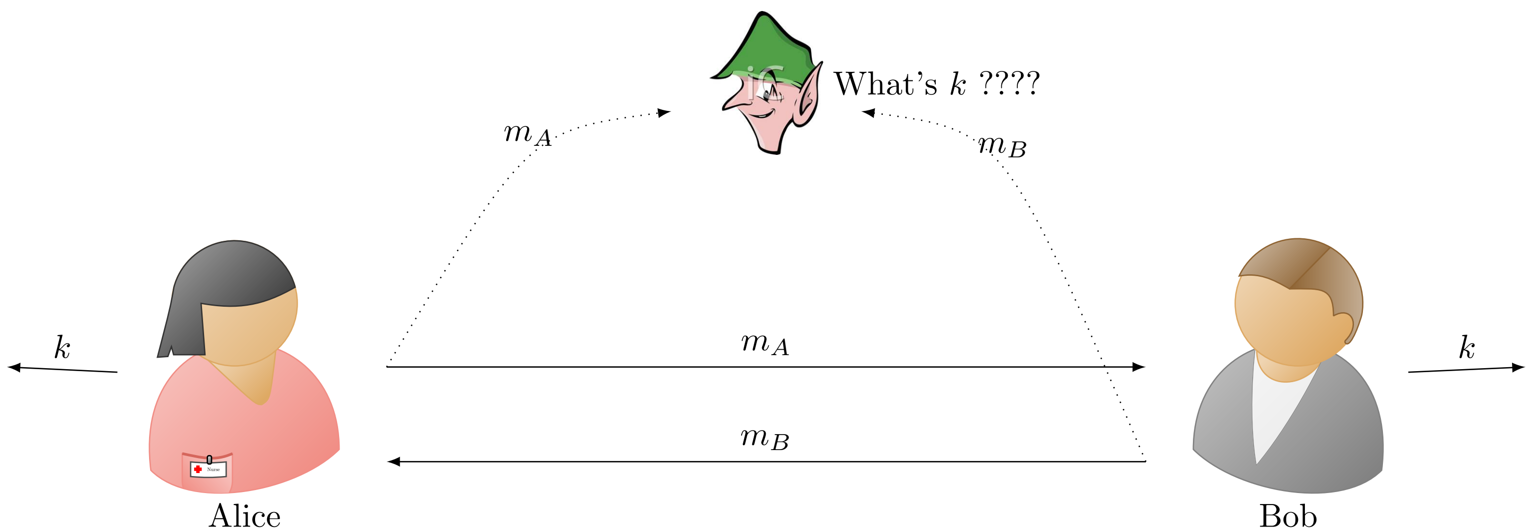
- Also: Digital signatures for integrity and non-repudiation.
  - Sign with private key s, verify with public key v
  - (Recall MACs; a shared key cryptosystem for message authentication).



Anyone can verify the signature!

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 Alice and Bob to setup shared secret key k.
  - Eavesdroppers cannot learn the key k.



Public keys solve more problems …

- Signatures provide **evidence**
  - Everyone can validate, only ‘owner’ can sign
- Establish shared secret keys
  - Use authenticated public keys
    - Signed by trusted certificate authority (CA)
  - Or: use DH (Diffie Hellman) key exchange
- Stronger resiliency to key exposure
  - Perfect forward secrecy and recover security
  - Threshold security
    - Resilient to key exposure of t out of n parties

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 (e.g., ciphertext or signature)

Public key crypto is harder…

- Requires related public, private keys
  - Usually we say a keypair (pk, sk)
  - 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, e.g., complex (slow) key generation

Commercial-grade security from [LV02]

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

For the table:

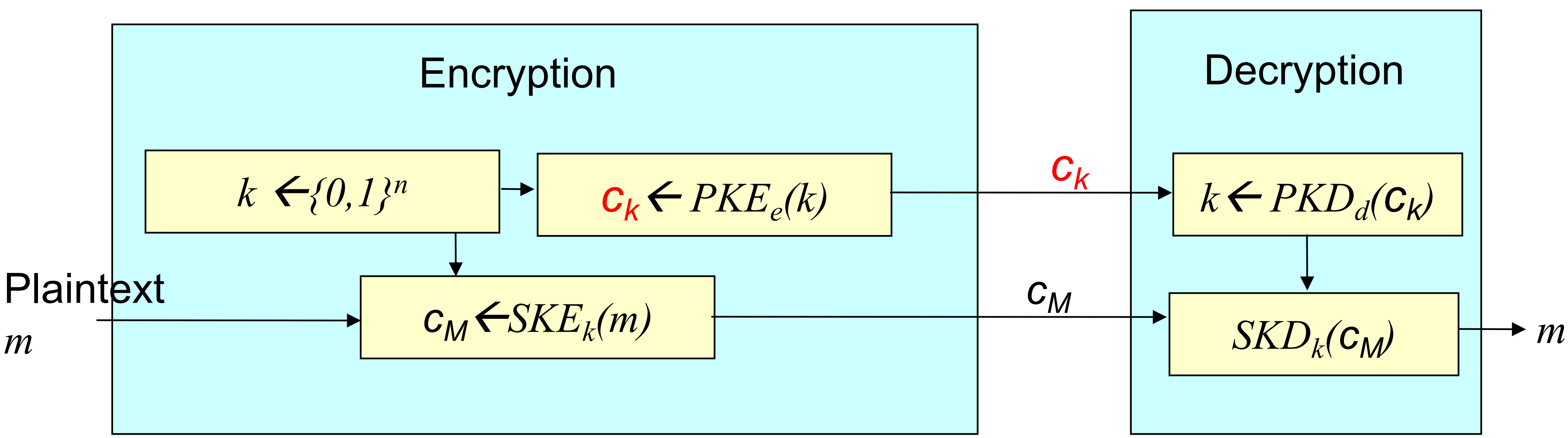
- The year indicates until when confidentiality to be preserved.
- AES: A symmetric encryption scheme
- RSA and DH: encryption schemes based on factoring and discrete log hardness problems
- ECIES: Elliptic Curve Integrated Encryption Scheme

In Sum

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

Hybrid Encryption

- 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 *k*, encrypts it under Bob's public key and sends the ciphertext c_k to Bob.
    - Bob can decrypt and recover *k,* and then use *k* to decrypt c_M.



Note: the figure above only focuses on confidentiality, additional modules are needed to ensure integrity.

Going Forward

- First, introduce the mathematical concepts (mainly number theory) that we need for a particular primitive/protocol.
  - This would involve hardness problems/assumptions.
- Then, study the primitive/protocol itself.
- Lastly, and as before, show correctness and reason about security.
  - In general, security will be based on mathematical hardness problems.

Number Theory Review
--Modular Arithmetic--

Notation

- $\mathbb{Z}$: The set of all integers {…, -3, -2, -1, 0, 1, 2, 3, …}.
- $\mathbb{Z}_n$: The set of integers modulo *n*, i.e., {0, 1, …, *n* - 1}
- $\mathbb{N}$: The set of natural numbers {1, 2, 3, …}.
- **Prime number:** p is prime if its only factors are 1 and p.
- **Composite number:** not prime.
- **Co-prime numbers:** m and n are co-prime if their greatest common divisor is 1.
- $\mathbb{Z}_p^*$: For a prime *p*, it is the set of integers modulo *p* *excluding zero*, i.e., {1, …, *p* - 1}
- $\mathbb{Z}_n^*$: For a composite *n*, it is the set of positive integers that are less than *n* (excluding zero) and co-prime to *n*.

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 $a \mod m$.*

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

$$(a+b) \mod m = [(a \mod m) + (b \mod m)] \mod m \qquad (1.2)$$

$$(a-b) \mod m = [(a \mod m) - (b \mod m)] \mod m \qquad (1.3)$$

$$a \cdot b \mod m = [(a \mod m) \cdot (b \mod m)] \mod m \qquad (1.4)$$

$$a^b \mod m = (a \mod m)^b \mod m \qquad (1.5)$$

Examples

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

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

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

Multiplicative Inverse

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

Multiplicative Inverse

Fact A.2. *Let $a \in \mathbb{Z}$ be an integer. We say that integer b is the* multiplicative inverse modulo m *of a, if $a \cdot b \equiv 1 \pmod{m}$; if it exists, we denote the multiplicative inverse by $b = a^{-1} \mod m$ (or, when m is clear from context, simply a^{-1}).*

An integer a has multiplicative inverse modulo integer $m > 0$, if and only if a and m are coprime, *namely, they do not have a common divisor (except 1).*

- Examples:
  - 3/5 mod 4 = 3 . 5^{-1} mod 4 = ?
  - 3/5 mod 6 = 3 . 5^{-1} mod 6 = ?

- The algorithm used to compute the inverse 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 n and co-prime to n.
  - gcd below is the greatest common devisor.

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

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 $\phi(p^l) = p^l - p^{l-1}$.*

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

Lemma 1.4. *Let $n = \Pi_{i=i}^{n}\left(p_i^{l_i}\right)$, 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:*

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

Euler's Theorem

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

$$m^l \mod n = m^{l \mod \phi(n)} \mod n \tag{1.19}$$

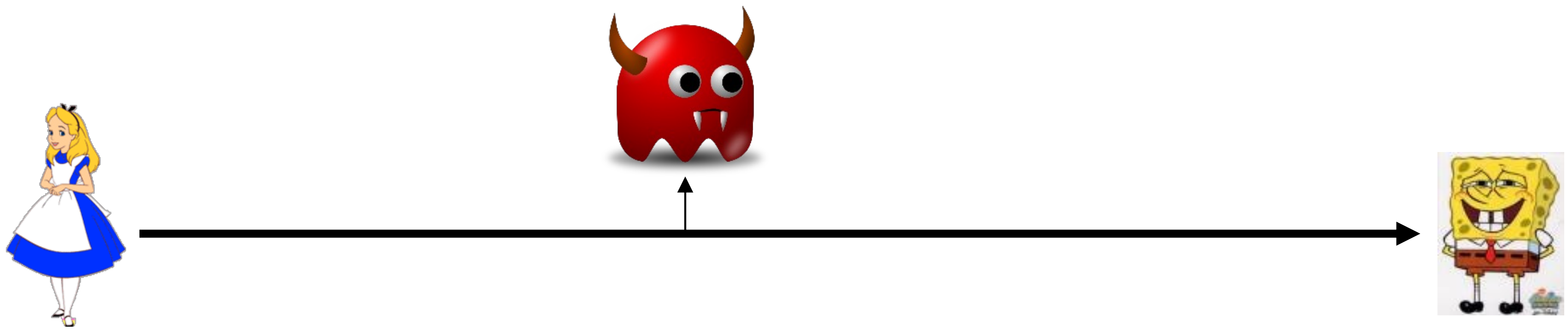
- Examples:
  - $13^{31} \bmod 31 = ?$
  - $27^{26} \bmod 10 = ?$

Key Exchange

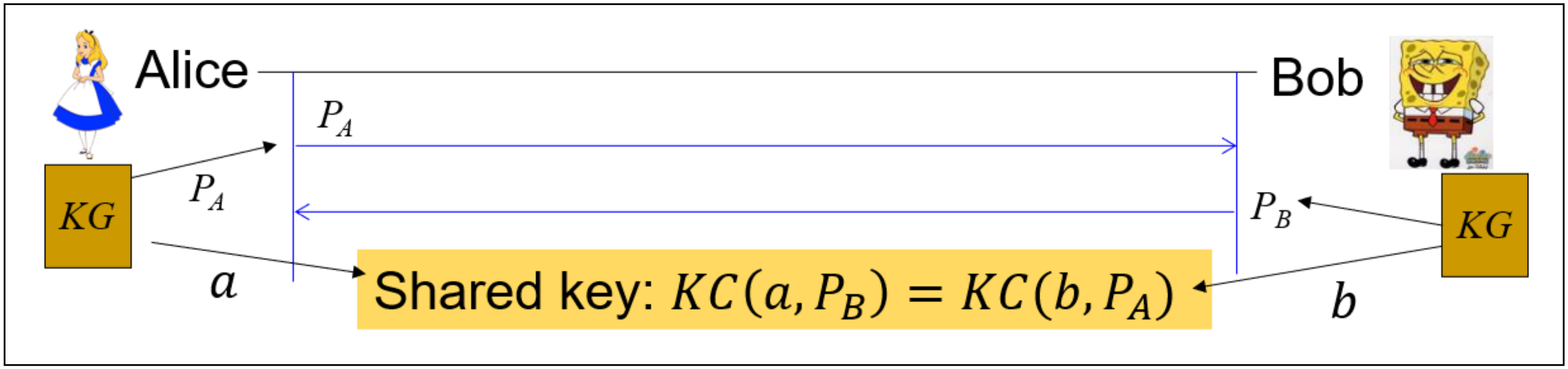
The Key Exchange Problem

Aka key agreement

- Alice and Bob want to agree on secret (key)
  - Secure against **eavesdropper** adversary
  - Assume no prior shared secrets (key)



Defining a Key Exchange Protocol



*KG: Key Generate, KC: Key Compute, a and b are secret, while P_A and P_B are public

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

--Group Theory Review--

- A group is a pair of (G, op) is composed of a set of elements G and an operation op such that G is closed under the operation op, i.e., for any two elements $a, b \in G$ we have $a \; op \; b = c \in G$, and it satisfies the following requirements:

Associativity: *for every* $a, b, c \in G$ *holds* $(a \cdot b) \cdot c = a \cdot (b \cdot c)$.

Identity element: *there exists a (unique) element in* G, *which we call the identity element and usually denote by* $1 \in G$, *such that for every element* $a \in G$ *holds:* $a = a \cdot 1 = 1 \cdot a$.

Inverse: *For each* $a \in G$, *there is an element* $a^{-1} \in G$ *such that* $a \cdot a^{-1} = a^{-1} \cdot a = 1$, *where* 1 *is the identity element. For each* a, *there is only one such element, which we call the* inverse *of* a *and denote* a^{-1}. *(From the identity element property, it follows that the identity element is always its own inverse.)*

A commutative group *is a group that also satisfies:*

Commutativity: *for every* $a, b \in G$ *holds* $a \cdot b = b \cdot a$.

Although the properties are for multiplication operations, same applies for addition. The only different is that the identity element is 0.

--Group Theory Review--

- We focus on finite commutative groups.
- We will consider Finite Additive Groups:
  - Example: $(\mathbb{Z}_n, +)$ where $\mathbb{Z}_n = \{0, 1, 2, \ldots, (n-1)\}$ and the operation is addition modulo n
  - Exercise: show the group above satisfies all properties listed in the previous slide.
- We will consider Finite Multiplicative Groups, mostly, modulo a prime p:
  - Example: $(\mathbb{Z}_p^*, .)$ where $\mathbb{Z}_p^* = \{1, 2, \ldots, (p-1)\}$ and the operation is multiplication modulo p
  - Exercise: show the group above satisfies all properties listed in the previous slide.
- We use the exponentiation notation to denote the repeated application of the group operation.
  - That is, $a^1 = a$ and $a^i = a^{i-1}\ op\ a$ and so on.

--Cyclic Groups--

Definition A.4 (Cyclic group, generator and order). *A group G is* cyclic, *if there is an element $g \in G$ such that for every element $a \in G$, there is an integer i such that $a = g^i$. Such an element g is called a* generator *of G. The* order *of G is the integer $q > 0$ such that $g^q = 1$, where g is a generator of G and 1 is the unit element of G.*

Note that $G = \{g^1, \ldots, g^q\} = \{1, g, g^2, \ldots, g^{q-1}\}$, hence, the order q of a cyclic group G, is also the number of element in G. We also define the order *of an* element *$a \in G$; this is the* smallest *possible integer $q > 0$ such that $a^q = 1$. In particular, the order of a is the same as the order of G if, and only if, a is a generator of g.*

Examples:

- For prime p , the additive group $\mathbb{Z}_p = \{0, 1, \ldots, p-1\}$ is a cyclic group of order p and every element in this group (except 0) is a generator (because the order of this group is prime). *Exercise: verify that!*
- For prime p , the group $\mathbb{Z}_p^*$ = {1,…p-1} is a cyclic multiplicative group. E.g., $\mathbb{Z}_7^* = \{1, 2, \ldots, 6\}$ is a cyclic group of order 6, a generator for this group is 3 (2, for example, is not a generator. *Exercise: verify that!).*

The Discrete Log Problem

- A computational hard problem is one that is:
  - Hard to solve
  - But easy to verify
- **Discrete log problem:** given a generator g and an element $a \in G$, find i such that $a = g^i$
  - Verification: exponentiation (efficient algorithm)
- Computing logarithm is quite efficient over the reals. But is discrete-log hard?
  - Some ‘weak’ groups, i.e., where discrete log is **not** hard:
    - $\mathbb{Z}_p^*$ for prime p, where $(p-1)$ has only ‘small’ prime factors
      - Using the Pohlig-Hellman algorithm
    - Mistakes/trapdoors found, e.g., in OpenSSL’16, so always check!
  - Other groups studied, considered Ok (‘hard’)
  - **Safe-prime** groups: $\mathbb{Z}_p^*$ for safe prime: $p = 2q + 1$ for prime q

Discrete Log Assumption

q -1

Definition 6.2 (The discrete logarithm problem). *Let Gen be a PPT algorithm that, on input 1^n, outputs (g, q) such that $\{1, g, \dots, g^{q-1}\}$ is a cyclic group (using a given group operation). We say that the* discrete logarithm problem *is* hard *for groups generated by Gen, if for every PPT algorithm $\mathcal{A}$ holds:*

$$\Pr\left[(g,q) \leftarrow Gen\,(1^n)\ ;\ y \stackrel{\$}{\leftarrow} \{1,\dots,q\} : y = \mathcal{A}(g^y)\right] \in NEGL(1^n) \qquad (6.6)$$

And remember, discrete-log is hard with respect to a particular group!

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

Diffie-Hellman [DH] Key Exchange

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

- Setup: Agree on a random safe prime *p* and generator *g* for the cyclic multiplicative group $\mathbb{Z}_p^*$
- Alice: pick at random secret integer a from $\mathbb{Z}_p^*$, then compute $P_A = g^a \mod p$, and send P_A to Bob.
- Bob: pick at random secret integer b from $\mathbb{Z}_p^*$, then compute $P_B = g^b \mod p$, and send P_B to Alice.
- Both parties: compute the shared key $k = g^{ab} \mod p$, do you see how?

Alice

Select a

$P_A = g^a \mod p$ →

Bob

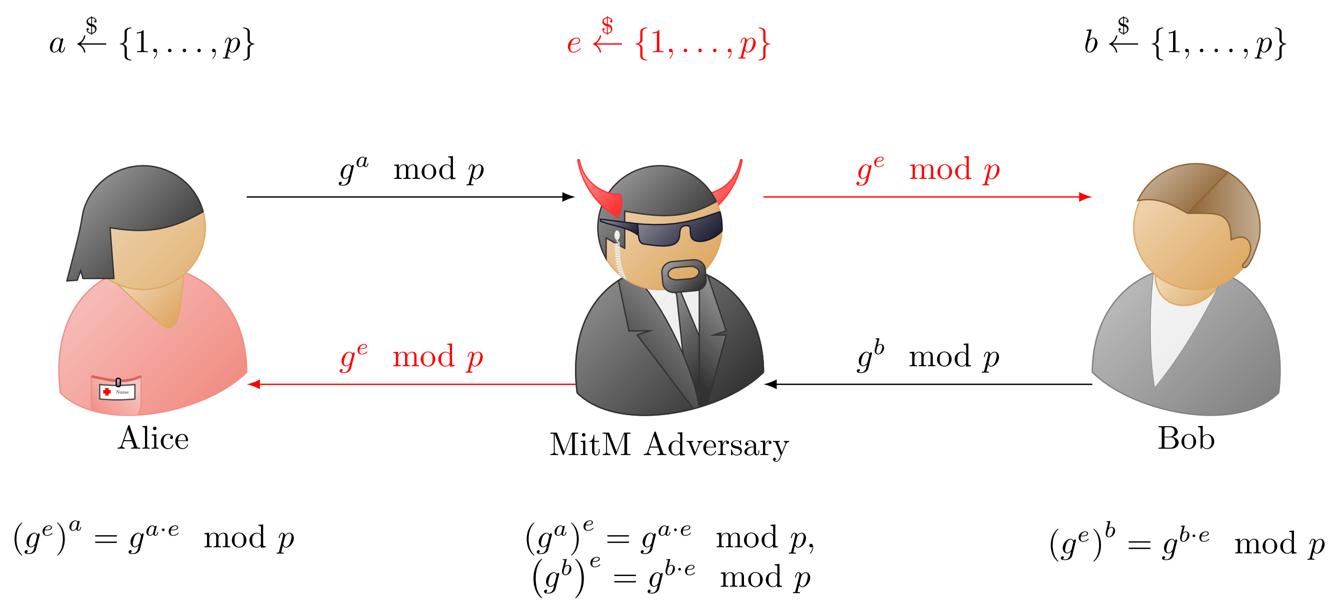
Select b

← $P_B = g^b \mod p$

$$(g^b)^a = g^{ab} = (g^a)^b \mod p$$

Caution: Authenticate Public Keys!

- Diffie-Hellman key exchange is only secure against eavesdroppers but not MitM attackers.
- So the public messages being sent must be authenticated, e.g., using digital signatures.
  - Still each party must have a certificate for her public (verification) key.



Security of [DH] Key Exchange

- Assume authenticated communication
- DH key exchange requires stronger assumption than Discrete Log:
  - Maybe from $g^b \ mod\ p$ and $g^a \ mod\ p$, adversary can compute $g^{ab} \ mod\ p$ (without knowing/learning a,b or ab)?
- The Computational Diffie-Hellman (CDH) Assumption is what we need.
  - In simple terms, it states that given $g^b \ mod\ p$ and $g^a \ mod\ p$, an efficient adversary cannot compute $g^{ab} \ mod\ p$ with non-negligible probability.
- So DH key exchange protocol is secure for groups in which the CDH assumption holds.
- Assume CDH holds. Can we use g^{ab} as key?
  - Not necessarily; maybe finding some bits of g^{ab} is easy?

Using DH securely?

- Can g^a , g^b expose *something* about g^{ab} *mod p* ?
  - Bad news: Finding (at least) one bit about g^{ab} *mod p* is easy! (details in textbook if interested)

- So, how to use DH 'securely'? Two options:
  - Option 1: Use DH but with a `stronger' group (other than $\mathbb{Z}_p^*$) for which the stronger DDH assumption holds.
    - The **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 $\mathbb{Z}_p^*$ and safe prime *p… (where only CDH holds)* but use a **key derivation function (KDF)** to derive a secure shared key.
    - Example, use an unkeyed hash function to obtain $k = h(g^{ab} mod\ p)$, where h is randomness-extracting hash function.

Covered Material From the Textbook

- Appendix A.2
- Chapter 6:
  - Sections 6.1 (except 6.1.8.3),
  - Section 6.2 (except 6.2.5, also 6.2.1 and 6.2.3 are optional reading),

Thank You!