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

Public Key Cryptography– Part I

Z. Jerry Shi

From Textbook Slides by Prof. Amir Herzberg, revised by Prof. G Almashaqbeh

Outline

- Intro to public key cryptography
- Key exchange
- Hardness assumptions: DL, CDH, DDH

Public Key Cryptology

- Kerckhoff: cryptosystem (algorithm) is public
- What we learned until now:
  - Only the key is secret (unknown to attacker)
  - Send a message ? The recipient needs the key!
  - Same key for encryption, decryption

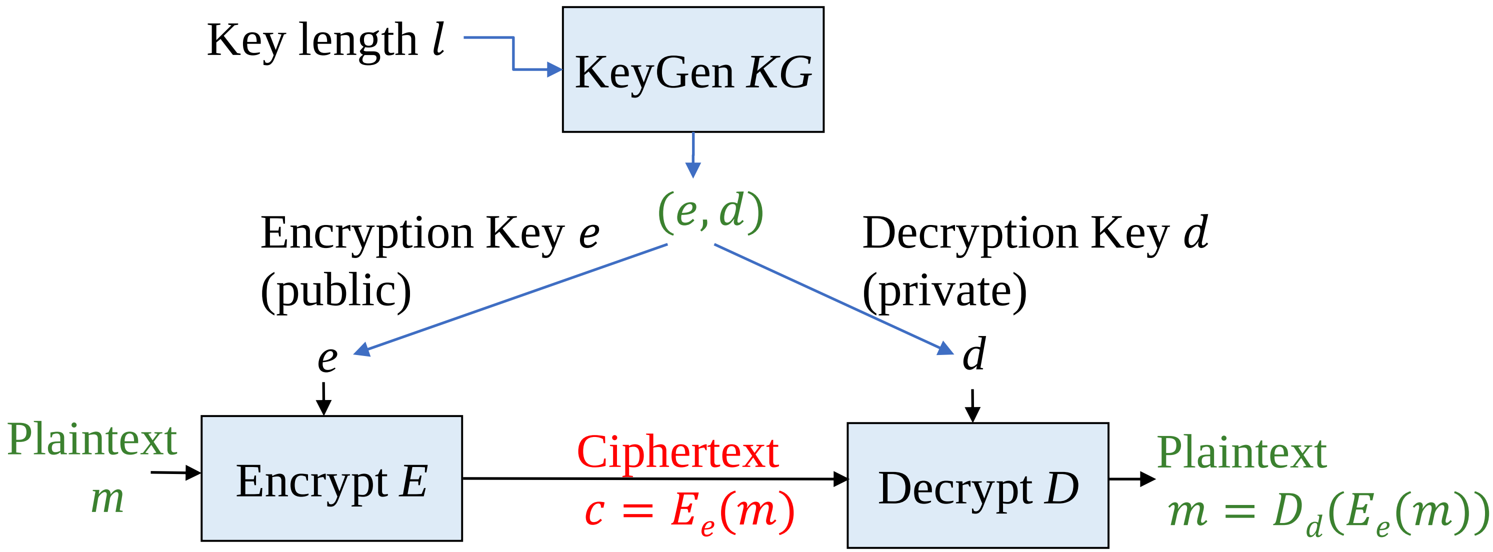
    → if you can encrypt, you can also decrypt!

How do two parties establish a secret key ?

Do they have to establish a key?

Public Key Cryptosystem (PKC)

- A pair of keys: a public key and private key
  - Everyone can use the public key to encrypt
  - Need the matching private key to decrypt
    - It is hard to find out private key from the public key
  - Everybody can send me mail, only I can read it.



Is it Only About Encryption?

- Also: Digital signatures
  - Recall MAC relies on shared keys

Key generation algorithm generates a pair of keys (s, v)

s : secret signing key

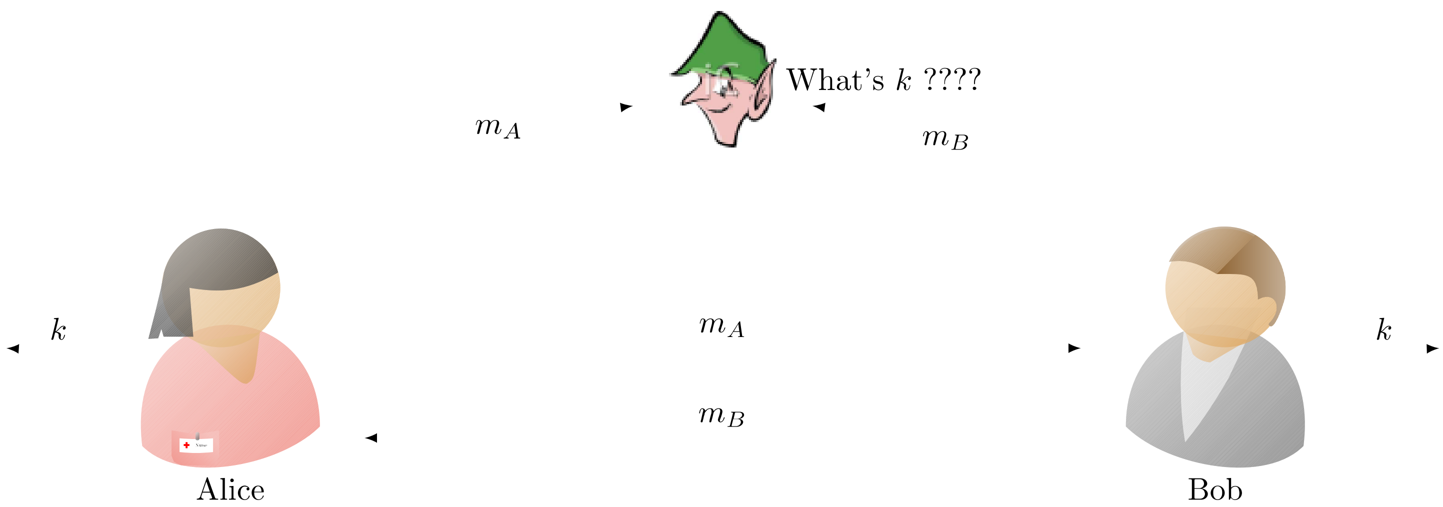
v: public verification key

Sign with s. Generate a signature σ.

Verify σ with the public key v, which can be done by anyone

More: Key-Exchange Protocol

- Key Exchange Protocols
  - Establish shared key between Alice and Bob without assuming an existing shared ('master') key !!
  - Use public information 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 (Diffie Hellman) 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 Summary

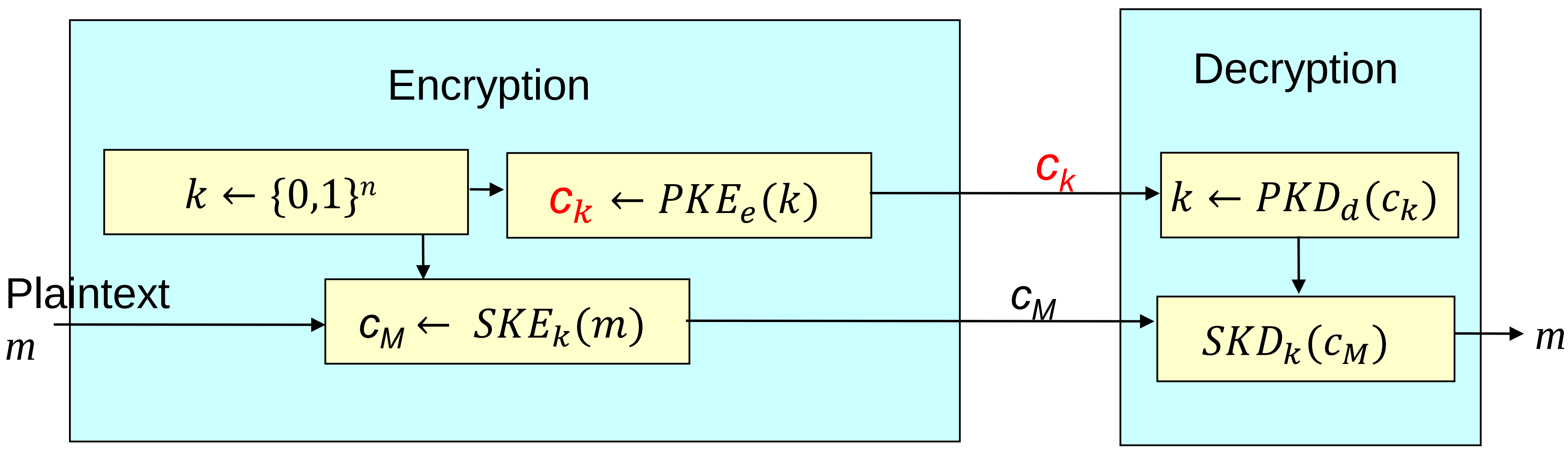
- Minimize the use of PKC
- In particular: apply PKC only to short inputs
- How ??

- For signatures:
  - Hash-then-sign
    - Sign the hash

- For public-key encryption:
  - Hybrid encryption
    - Protect the key used in symmetric-key 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 the k, encrypt it under Bob's public key and send it to Bob, Bob can then recover this key).



Hard Modular Math Problems

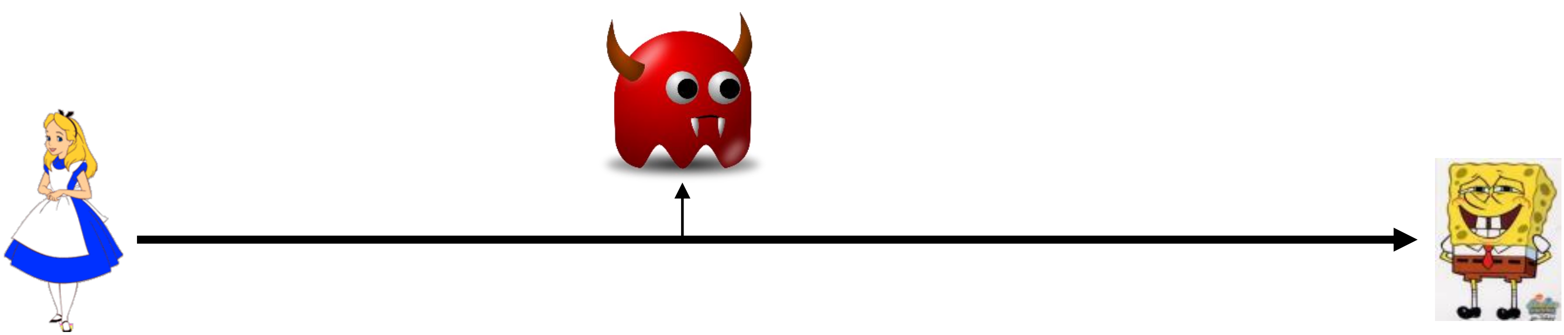
- No efficient solution, in spite of extensive efforts
  - But: **verification** of solutions is easy (`one-way' hardness)
  - Discrete log: exponentiation
- **Problem 1: Factoring**
  - Choose large primes p and q randomly
  - Given $n = p \cdot q$, it is infeasible to find p and q
  - Verification? Easy, just multiply p and q
  - Basis for the RSA cryptosystem and many other tools
- **Problem 2: Discrete logarithm in cyclic group 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

Key Exchange

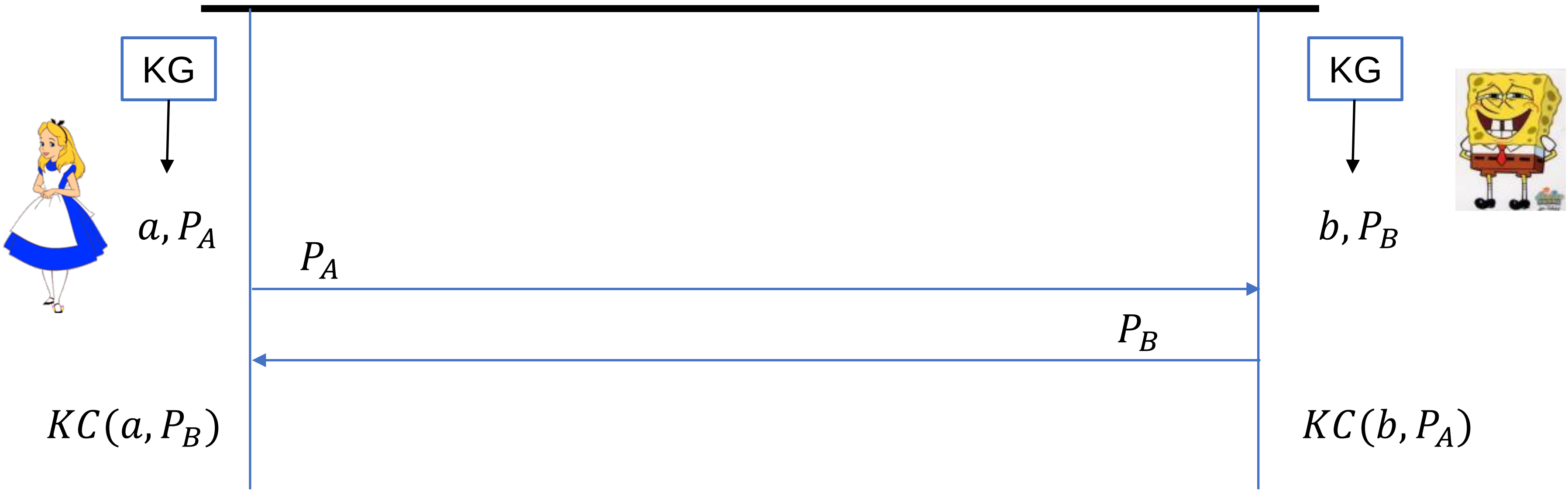
The Key Exchange Problem

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 and key indistinguishability

Correctness: both parties compute the same shared key

$$KC(a, P_B) = KC(b, P_A)$$

Key indistinguishability: the established key is indistinguishable from random

Discrete Log Assumption

$\boldsymbol{p = 2q + 1}$
for prime $\boldsymbol{q}$

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

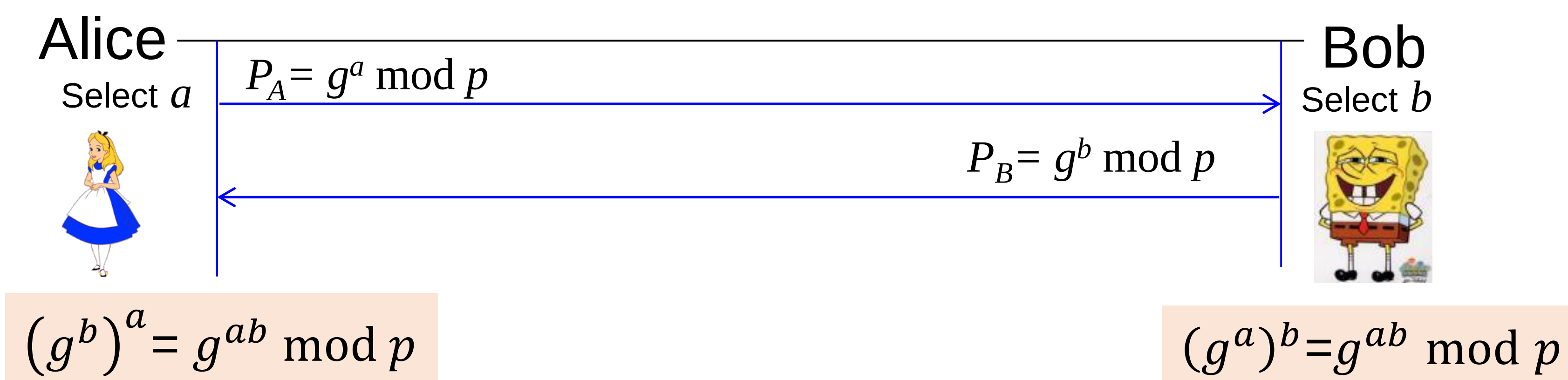
$$\Pr\left[\begin{array}{c} g \leftarrow Generator\left(Z_p^*\right); \\ x \overset{\$}{\leftarrow} Z_p^* \\ a = A(x) \quad s.t. \;\; x = g^a \bmod p \end{array}\right] \approx negl(n)$$

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

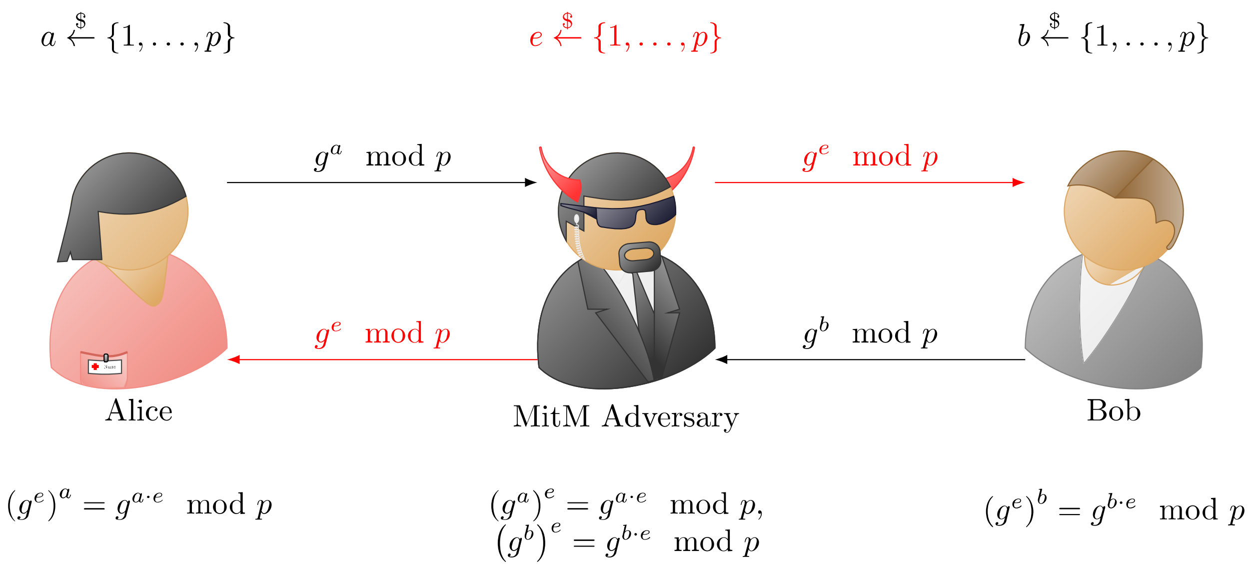
Diffie-Hellman [DH] Key Exchange

- Simplified Discrete Exponentiation Key Exchange
- Agree on a random safe prime p and a generator g for the cyclic group Z_p^*
- Alice and Bob set up a shared key as follows
  - Alice selects a and keeps it secret (does not send it to Bob)
  - Bob selects b and keeps it secret (does not send it to Alice)



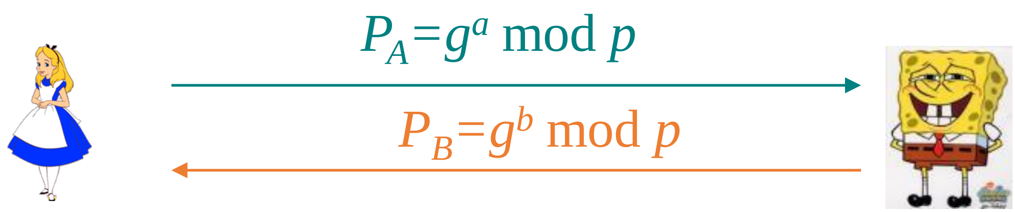
Caution: Authenticate Public Keys!

- Diffie-Hellman key exchange is only secure when 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



Security of [DH] Key Exchange

- Assume authenticated communication
- Based on Computational Discrete Log Assumption
  - Not really DLP
- Can adversary can compute $g^{ab} \bmod p$, given $g^{a} \bmod p$ and $g^{b} \bmod p$?
  - They do not have to know a, b or ab



Computational DH (CDH) Assumption

DH requires CDH, stronger than Discrete Log

Given PPT adversary A:

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

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 Z_p^*(multiplicative group for (safe) prime p)
- Can g^a g^b expose *something* about $g^{ab} \bmod p$?
- Bad news:
  - Finding (at least) one bit about $g^{ab} \bmod p$ is easy!
  - (details in textbook if interested)
- So…how to use DH 'securely'?

Using DH securely?

- Two options!
  - Option 1: Use DH but with a `stronger' group, where DDH holds
    - 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)

- <u>Randomized - KDF:</u> $k = KDF_s(g^{ab} \bmod p)$ where KDF is a key derivation function and s is public random ('salt')

- <u>Deterministic - crypto-hash:</u> $k = h(g^{ab} \bmod p)$ where h is <u>randomness-extracting crypto-hash</u>
  - No need in salt, but **not** provably-secure

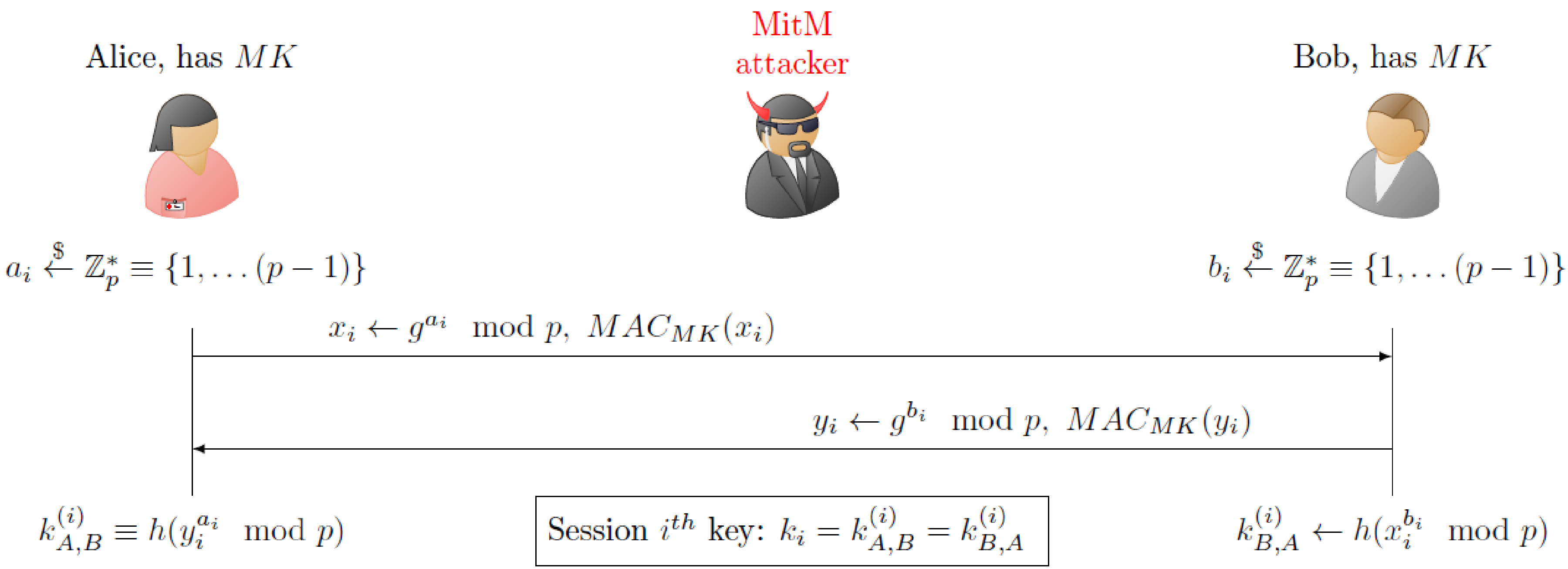
Authenticated DH

- 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

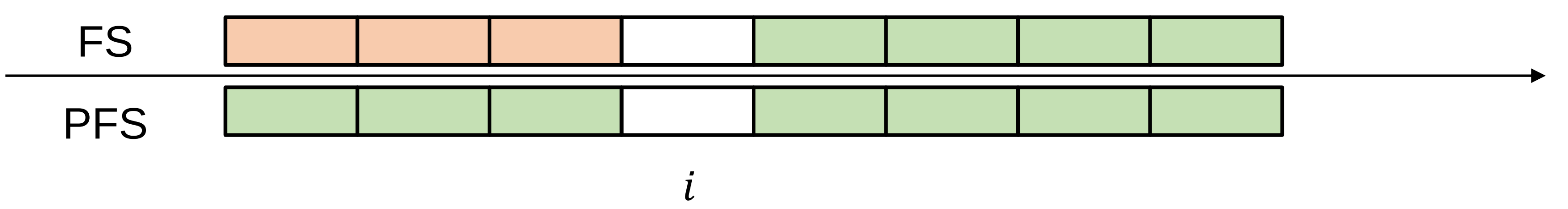
Auth-h-DH Protocol

- Assumptions
  - MK is secret, MAC is secure, and h is a keyless randomness extractor hash function
- Perfect forward secrecy (PFS) !
  - The session key is secure if MK is exposed after session ends



Forward Secrecy (FS) vs Perfect Forward Secrecy (PFS)

- Forward Secrecy (FS):
  - Confidentiality of session i is resilient to exposure of all keys in later sessions
    - It is fine if keys in session $i + 1$ are exposed
    - Insecure if keys in session $i - 1$ are exposed
- Perfect Forward Secrecy (PFS):
  - Confidentiality of session i is resilient to exposure of all keys in other (earlier or later) sessions, after session i ended
    - Remain secure if keys in sessions $i - 1$, $i + 1$ are exposed



Resilience to Key Exposure: Recover Security

- The previous DH protocol does not achieve recover security, why?
  - Exposing MK makes all future session vulnerable to MitM (this adversary can authenticate any public key he wants to the other party)
- There is another version, called Ratchet DH, that achieves perfect recover security.
  - Will not be covered in this class

Covered Material From the Textbook

- Chapter 6: sections 6.1, 6.2, and 6.3 (except 6.3.2)