# CS 4770: Cryptography

# CS 6750: Cryptography and Communication Security

Alina Oprea
Associate Professor, CCIS
Northeastern University

March 19 2018

# Announcements

- Office hours this week
  - Wed 2:30-4:30pm
- Distinguished speaker on Thu 03/22
  - Location 97 Cargill, 3-4:30pm
  - Prof Mike Reiter, UNC Chapel Hill
  - Title: “Side channels in multi-tenant environments”
  - Extra credit for next homework: submit a paragraph about his talk
- If anyone is interested in meeting him 4:30-5pm (ISEC 632), please email me

# Outline

- Generating large primes
  - Miller-Rabin primality testing
- How to distribute cryptographic keys
- Key distribution centers
  - Needham-Shroeder
- Public-key cryptography
  - Diffie-Hellman key exchange

# How to generate large primes?

- Input: length n; parameter t
- Output: a uniform n-bit prime p
- For i = 1 to t:
  - $p' \leftarrow \{0,1\}^{n-1}$
  - $p = 1||p'$
  - If p is prime, return p

Primality test

- Return fail

The fraction of prime n-bit numbers is > 1/3n

Set t to get a negligible prob of fail (e.g., for t=3n$^2$, probability of failure < e$^{-n}$)

# Miller-Rabin primality test

- Input: Integer N; parameter t
- Output: A decision whether N is prime/composite
- If N even, return "composite"
- If N perfect power, return "composite"
- Decompose $N - 1 = 2^r u$, u odd
- For j = 1 to t:
  - $a \leftarrow$ {1,…,N-1} // choose at random
  - If $a^u \neq \pm 1\, mod\, N$ and $a^{2^i u} \neq -1\, mod\, N, \forall i \in \{1, \ldots, r-1\}$, return "composite"
- Return "prime"

If N composite, prob ½ to find strong witness in each iteration
If N composite, the probability that it outputs prime is $1/2^t$

# Test perfect powers

- Input: Integer N of n bits
- Output: Is N perfect power (exists m,e st N=$m^e$)
- For all e < n
  - Set a = 1, b = N
  - While $a \leq b$
    - $m = \left\lfloor \frac{a+b}{2} \right\rfloor$
    - If $m^e = N$, return "perfect power"
    - If $m^e > N, set\ b = m - 1$
    - If $m^e < N, set\ a = m + 1$
  - Return "not perfect power"

# How to distribute the cryptographic keys?

- If the users can meet in person beforehand – **it's simple.**

- But what to do if they **cannot meet**?

  (a typical example: on-line shopping)

Private-key cryptography relies on secure distribution of secret keys

# Key Distribution Centers

Some *server* (a **Key Distribution Center, KDC**) “gives the keys” to the users

- **feasible** if the users are working in one company
- Users share keys with **KDC** only
- **KDC** generates new fresh keys (called *session keys*) when users initiate communication

Disadvantages

- **infeasible** on the internet
- relies on the honesty of **KDC**
- Who can implement a trusted **KDC**?
- **KDC** needs to be permanently available
- **KDC** is single point of failure

# How to establish a key with a trusted server?

key shared by Alice and the server: **$K_{AS}$**

**server S**

key shared by Bob and the server: **$K_{BS}$**

**A**

want to establish a **fresh session key**

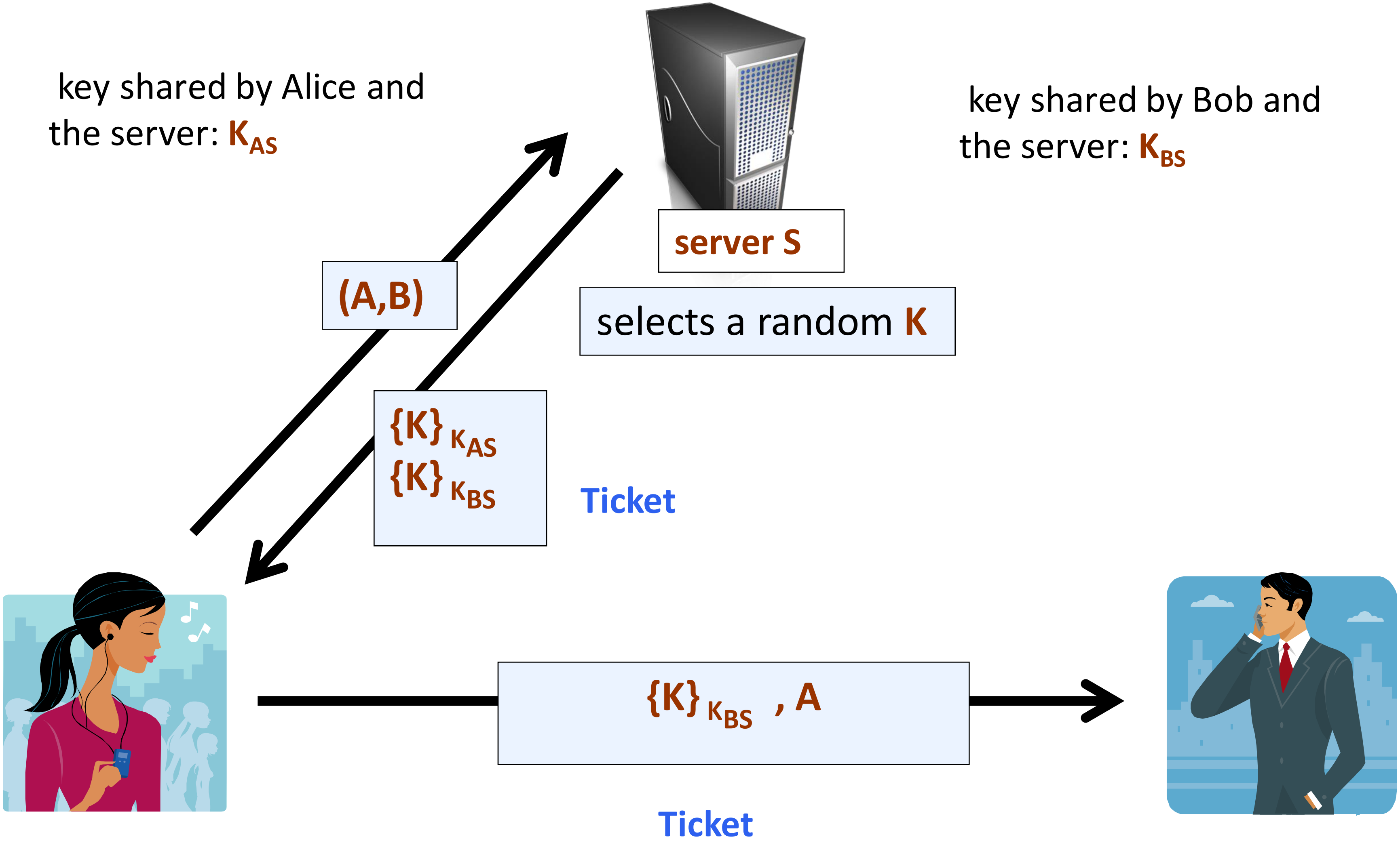
**B**

# Notation

**$\{M\}_K$** – a message **M** encrypted and authenticated with **K**

- Any authenticated encryption scheme can be used
- $K = (K_0, K_1)$: one key for encryption, one for authentication
- Encrypt-then-MAC the preferred method

# An idea (1)



key shared by Alice and the server: $K_{AS}$

key shared by Bob and the server: $K_{BS}$

server S

selects a random K

(A,B)

$\{K\}_{K_{AS}}$

$\{K\}_{K_{BS}}$

Ticket

$\{K\}_{K_{BS}}$ , A

Ticket

# Generating keys: a toy protocol

Goal: Alice wants a shared key with Bob

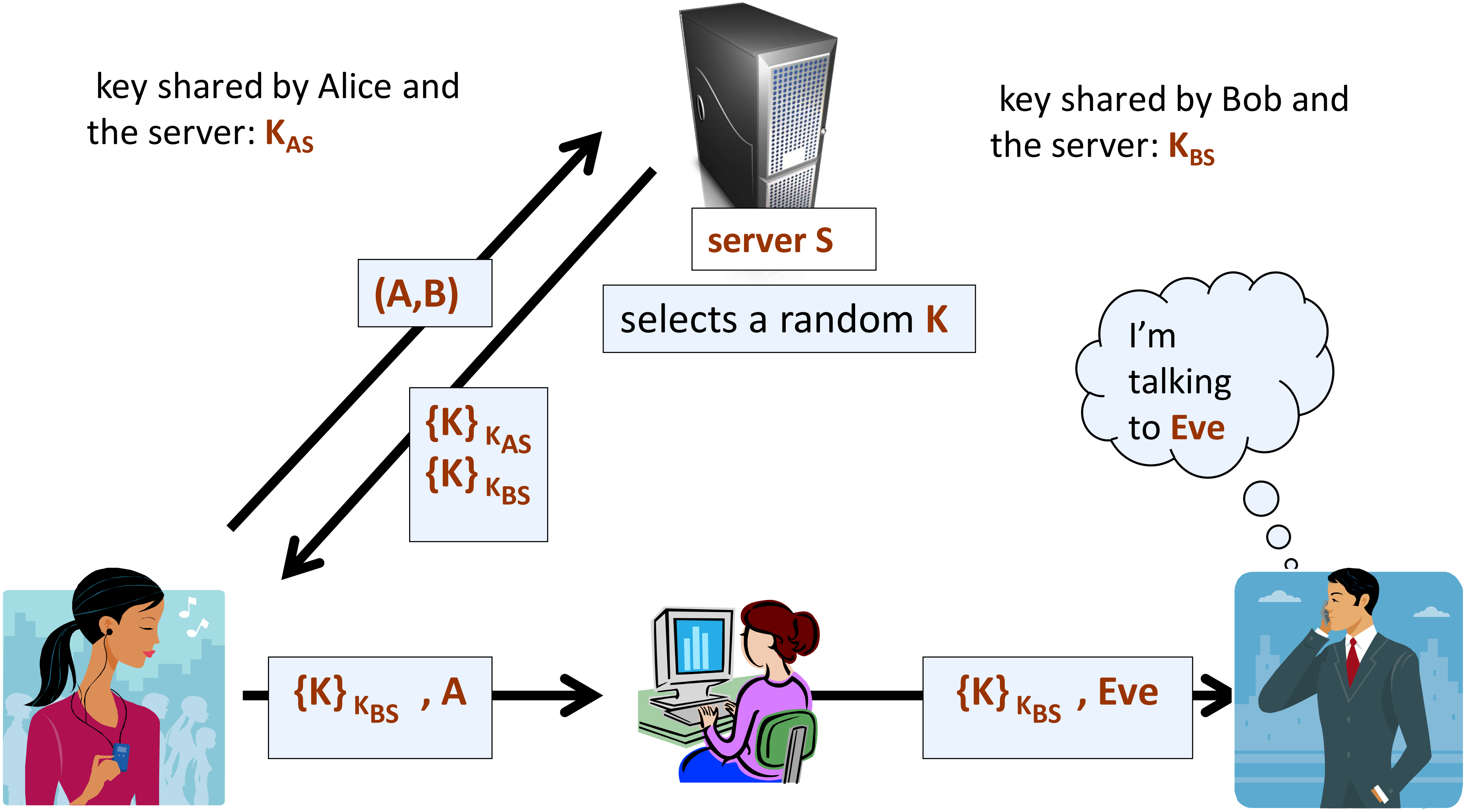
Adversarial model: Eavesdropping security only

Eavesdropper sees $\{K\}_{K_{AS}}$; ticket = $\{K\}_{K_{BS}}$

Encryption is CPA-secure ⇒
Eavesdropper learns nothing about k

How about active attacks?

# An attack



key shared by Alice and the server: $K_{AS}$

key shared by Bob and the server: $K_{BS}$

server S

selects a random K

(A,B)

$\{K\}_{K_{AS}}$
$\{K\}_{K_{BS}}$

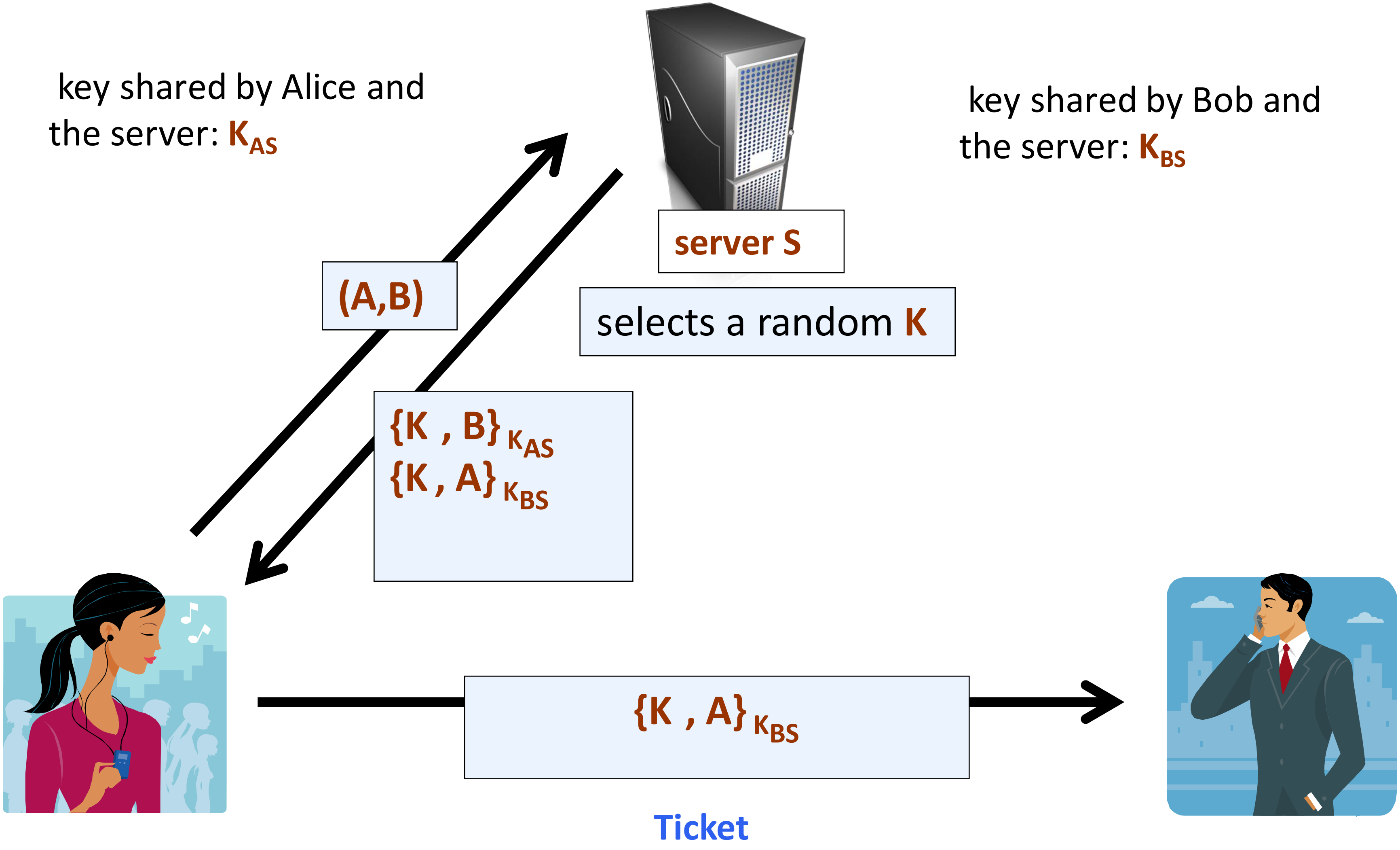
$\{K\}_{K_{BS}}$ , A

$\{K\}_{K_{BS}}$ , Eve

I'm talking to Eve

Man-in-the-middle

# An idea (2)



key shared by Alice and the server: $K_{AS}$

key shared by Bob and the server: $K_{BS}$

server S

selects a random K

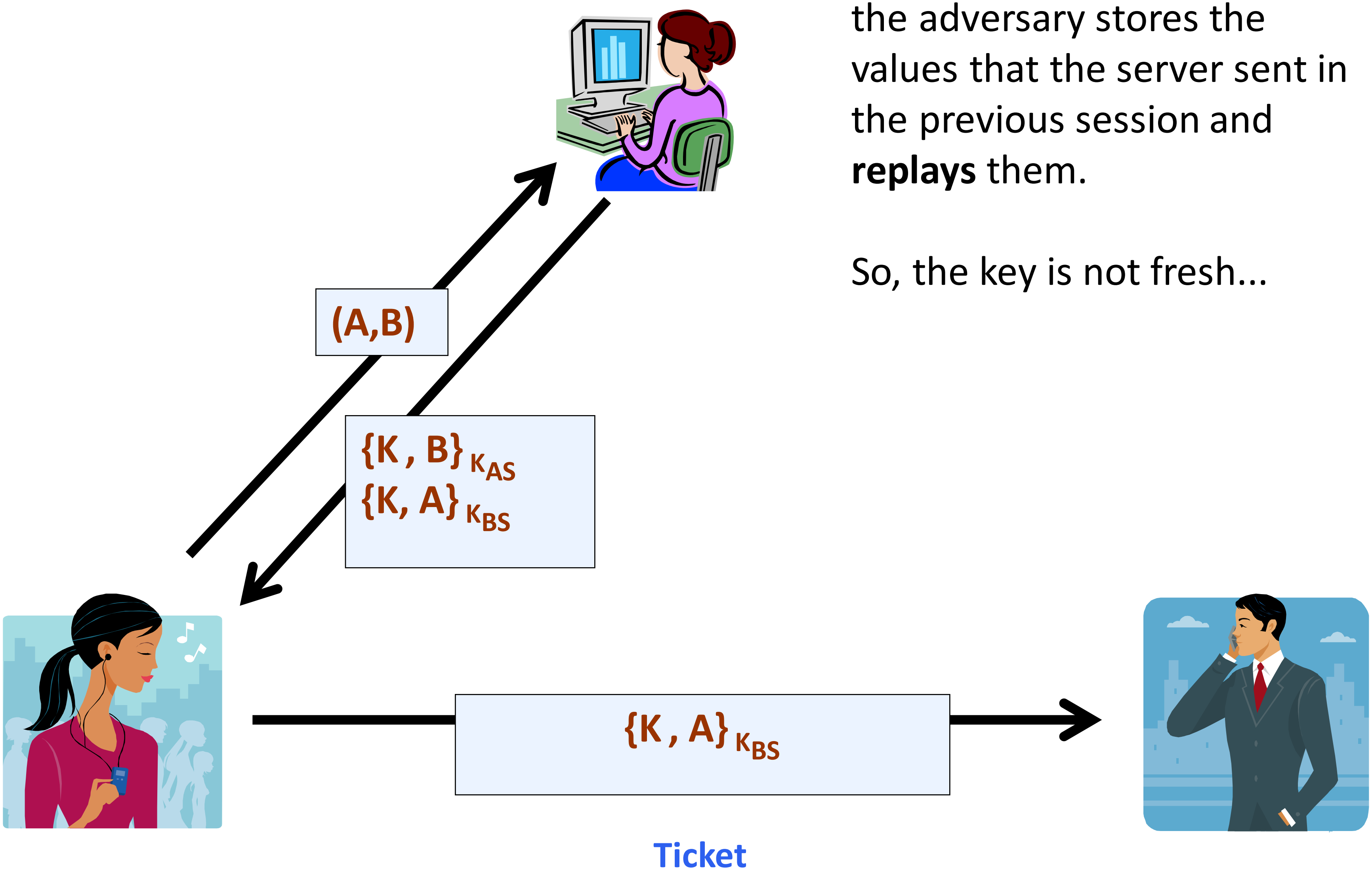
(A,B)

$\{K , B\}_{K_{AS}}$
$\{K , A\}_{K_{BS}}$

$\{K , A\}_{K_{BS}}$

Ticket

# A replay attack



(A,B)

$\{K , B\}_{K_{AS}}$
$\{K, A\}_{K_{BS}}$

$\{K , A\}_{K_{BS}}$

Ticket

the adversary stores the values that the server sent in the previous session and **replays** them.

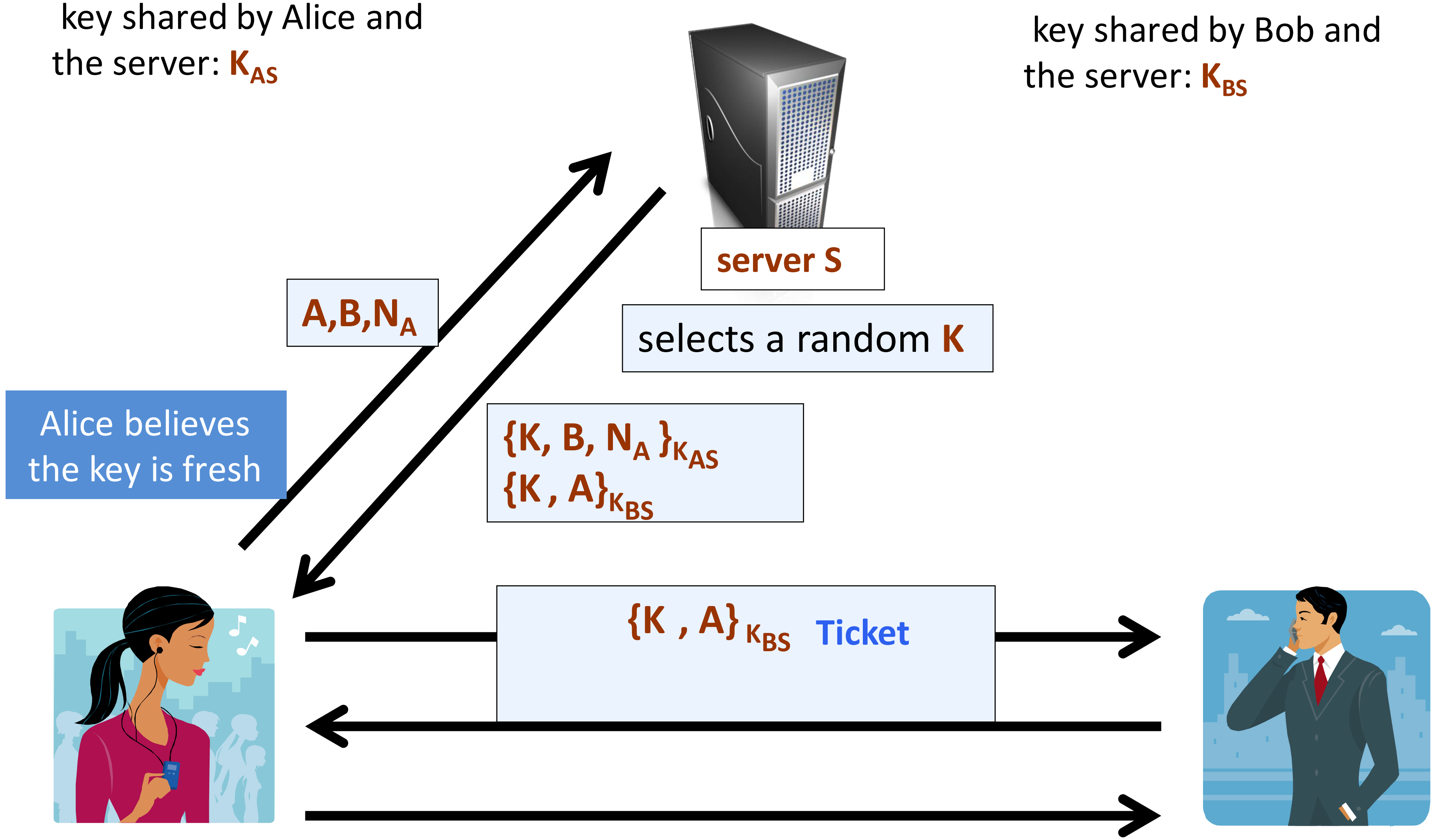
So, the key is not fresh...

# How to protect against the replay attacks?

**Nonce** – “number used once”.

**Nonce** is a random number generated by one party and returned to that party to show that a message is newly generated.

# An idea (3): Needham Schreoder 1972



key shared by Alice and the server: $K_{AS}$

key shared by Bob and the server: $K_{BS}$

server S

selects a random K

$A, B, N_A$

Alice believes the key is fresh

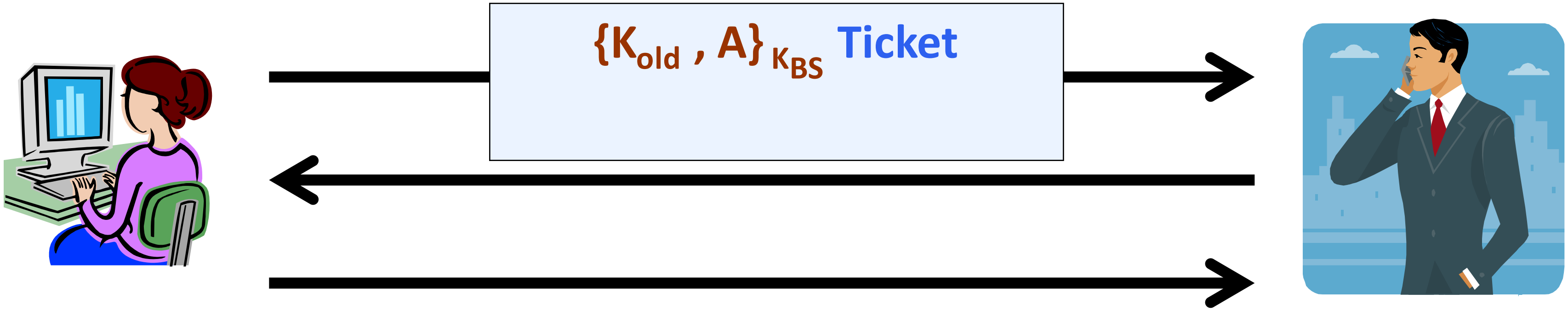
$\{K, B, N_A\}_{K_{AS}}$

$\{K, A\}_{K_{BS}}$

$\{K, A\}_{K_{BS}}$ Ticket

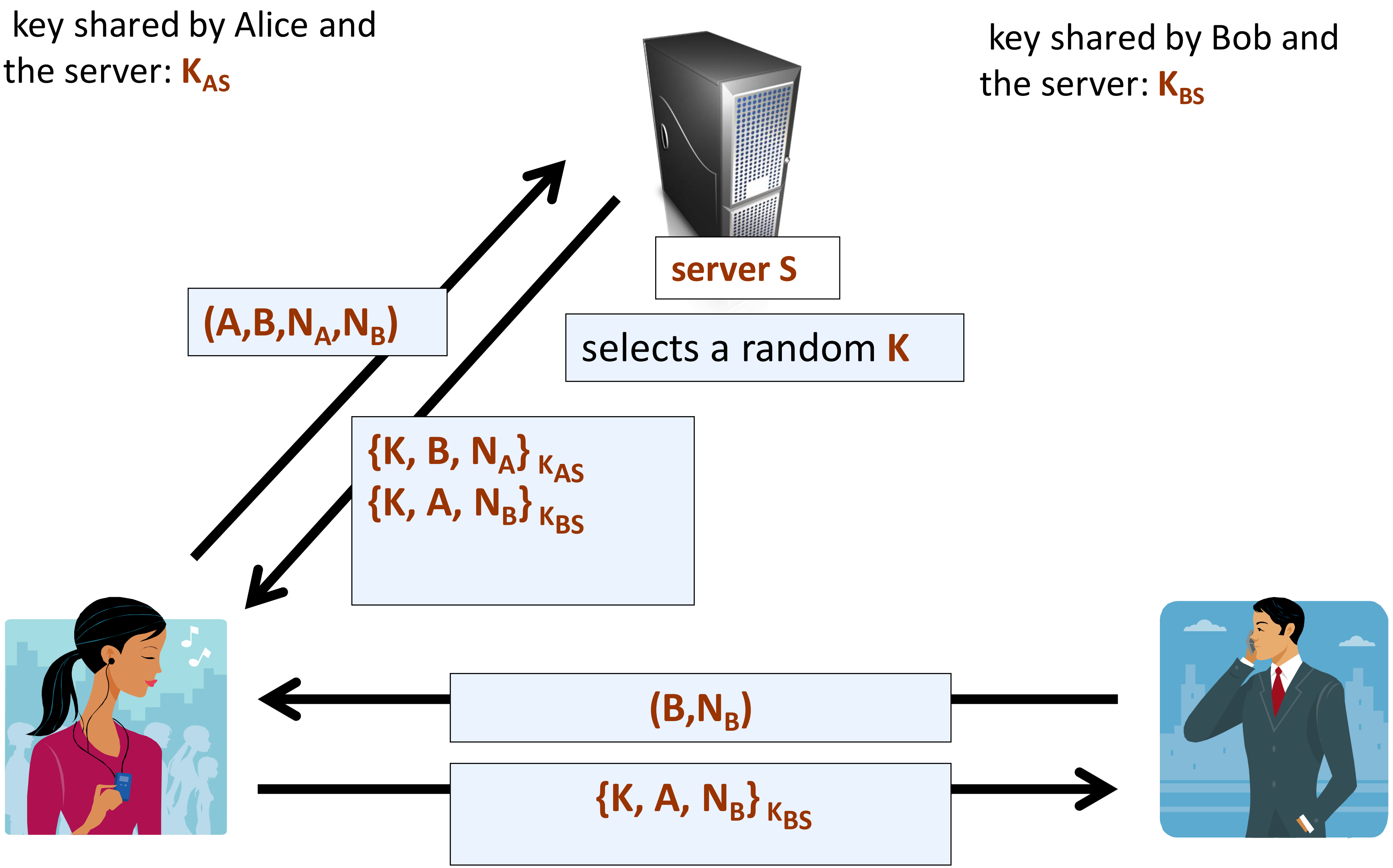
# An attack on Needham Schroeder

- Assume that an old session key $K_{old}$ is compromised by the adversary
- **B** can not tell if the key is fresh



$\{K_{old}, A\}_{K_{BS}}$ Ticket

# Solution



key shared by Alice and the server: $K_{AS}$

key shared by Bob and the server: $K_{BS}$

server S

selects a random K

$(A,B,N_A,N_B)$

$\{K, B, N_A\}_{K_{AS}}$
$\{K, A, N_B\}_{K_{BS}}$

$(B,N_B)$

$\{K, A, N_B\}_{K_{BS}}$

Kerberos uses timestamps to guarantee key freshness

# Key Distribution Centers

Some *server* (a **Key Distribution Center, KDC**) “gives the keys” to the users

- **feasible** if the users are e.g. working in one company
- Users share keys with **KDC** only
- **KDC** generates new fresh keys (called *session keys*) when users initiate communication

Disadvantages

- **infeasible** on the internet
- relies on the honesty of **KDC**
- Who can implement a trusted **KDC**?
- **KDC** needs to be permanently available
- **KDC** is single point of failure

# Key question

Can we generate shared keys without an **online** trusted 3rd party?

Answer: yes!

Starting point of public-key cryptography:

- Merkle (1974), Diffie-Hellman (1976), RSA (1977)
- More recently
  - Identity-based encryption [BF 2001)
  - Functional encryption [BSW 2011]

# The solution **without** KDC

## **Public-Key Cryptography**

Ralph Merkle (1974)

Whitfield Diffie and Martin Hellman (1976)

# A little bit of history

- **Diffie and Hellman** were the first to publish a paper containing the idea of the public-key cryptography:

  W.Diffie and M.E.Hellman,
  **New directions in cryptography**
  IEEE Trans. Inform. Theory, IT-22, 6, **1976**, pp.644-654**.**

- A similar idea was described by **Ralph Merkle**:
  - in **1974** he described it in a project proposal for a Computer Security course at UC Berkeley
    (it was rejected)
  - in **1975** he submitted it to the CACM journal (it was rejected)

  (see http://www.merkle.com/1974/ )

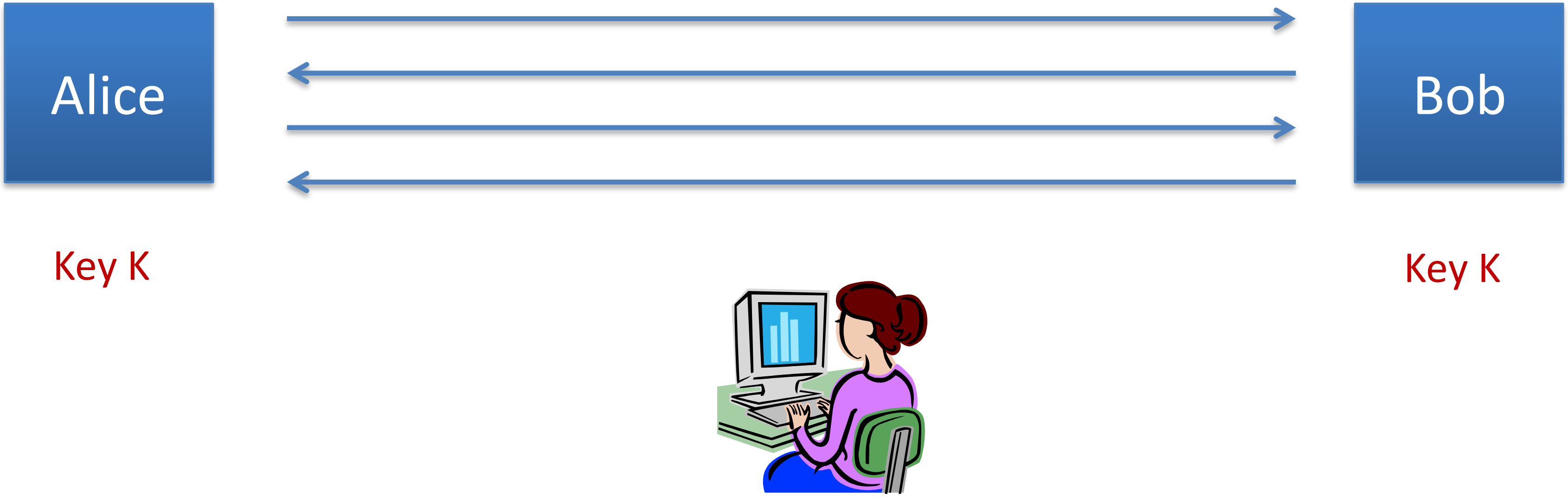
- 1977: R. Rivest, A. Shamir and L. Adelman published the first construction of public-key encryption (RSA)

- It 1997 the GCHQ (the British equivalent of the NSA) revealed that they knew it already in **1973**.

# Key exchange without an online TTP?

Goal: Alice and Bob want shared secret, unknown to eavesdropper

- For now: security against eavesdropping only (no tampering)



Alice

Bob

Key K

Key K

# The Diffie-Hellman protocol

Fix a large prime p (e.g. 600 digits)

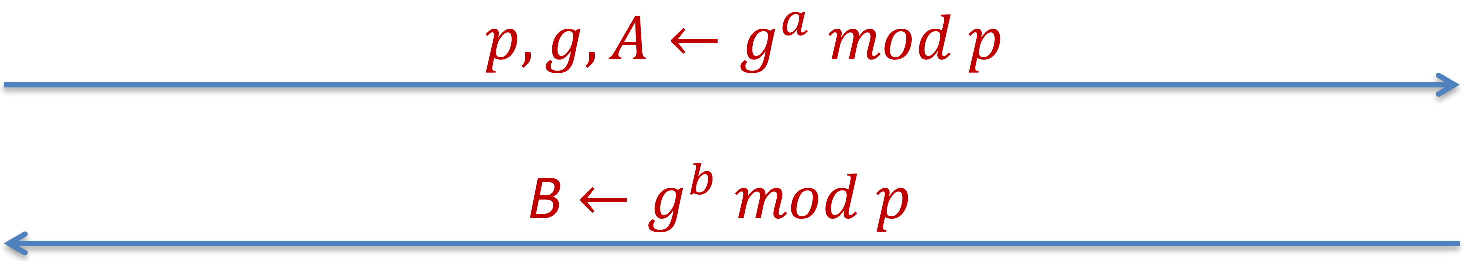
Fix an integer g in {1, …, p}

**Alice**

**Bob**

choose random **a** in {1,…,p-1}

choose random **b** in {1,…,p-1}



$p, g, A \leftarrow g^a \; mod \; p$

$B \leftarrow g^b \; mod \; p$

$\mathbf{B^a}$ (mod p) = $(g^b)^a$ = $\mathbf{k_{AB} = g^{ab}}$ (mod p) = $(g^a)^b$ = $\mathbf{A^b}$ (mod p)

# Security (informally)

Eavesdropper sees: p, g, $A=g^a \pmod p$, and $B=g^b \pmod p$

Can she compute $g^{ab} \pmod p$ ??

More generally: define $DH_g(g^a, g^b) = g^{ab} \pmod p$

How hard is the DH function mod p?

# Intractable problems with primes

Fix a prime p>2 and g in $(Z_p)^*$ of order q.

Consider the function: **$x \longmapsto g^x$ in $Z_p$**

Now, consider the inverse function:

**$\text{Dlog}_g\,(g^x) = x$ where $x$ in $\{0, \dots, q\text{-}2\}$**

Example:

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| in $\mathbb{Z}_{11}$ : | 1, | 2, | 3, | 4, | 5, | 6, | 7, | 8, | 9, | 10 |
| $\text{Dlog}_2(\cdot)$ : | 0, | 1, | 8, | 2, | 4, | 9, | 7, | 3, | 6, | 5 |

# DLOG: more generally

Let **G** be a finite cyclic group and **g** a generator of G

$G = \{ 1 , g , g^2 , g^3 , \ldots , g^{q-1} \}$ ( q is called the order of G )

**<u>Def</u>**: We say that **DLOG is hard in G** if for all efficient alg. A:

$\Pr_{g \leftarrow G,\, x \leftarrow Z_q} [ A( G, q, g, g^x ) = x ] < \text{negligible}$

Example candidates:

(1) $(Z_p)^*$ for large p, (2) Elliptic curve groups mod p

# How hard is the DH function mod p?

Suppose prime p is n bits long.

Best known algorithm (GNFS): run time exp( $\tilde{O}(\sqrt[3]{n})$ )

| Level of security | modulus size | Elliptic Curve size |
|---|---|---|
| 80 bits | 1024 bits | 160 bits |
| 128 bits | 3072 bits | 256 bits |
| 256 bits (AES) | **15360** bits | 512 bits |

As a result: slow transition away from (mod p) to elliptic curves

# Decisional Diffie-Hellman

Let **G** be a finite cyclic group and **g** generator of G

$G = \{ 1 , g , g^2 , g^3 , \ \ldots \ , \ g^{q-1} \}$

q is called the order of G

**Definition**: We say that **DDH is hard in G** if for all PPT adversaries A:

$|\Pr[\ A( G, q,\ g, g^x ,g^y ,g^{xy} ) = 1 ] - \Pr[\ A( G, q,\ g, g^x ,g^y ,g^z ) = 1 ] \ | \ < \ \text{negligible}$

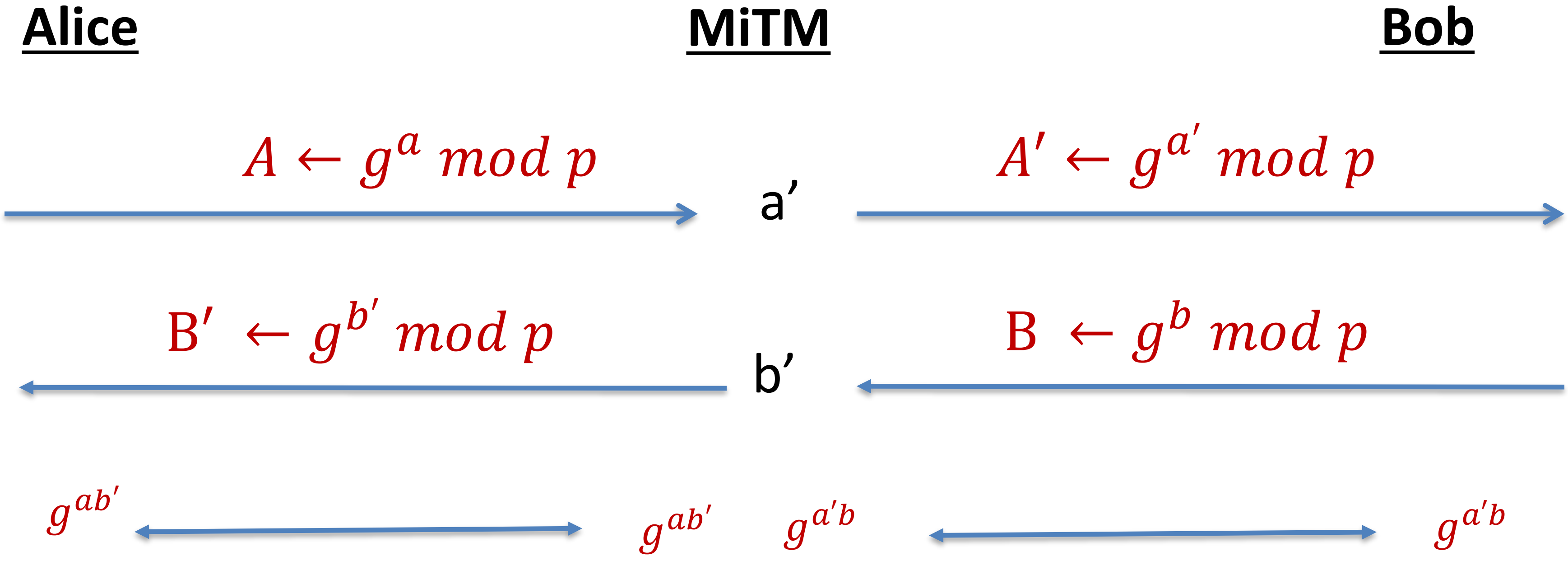
x, y, z are chosen uniformly at random in {1,…q-1}

# Security of Diffie-Hellman

- If DDH is hard, then Diffie-Hellman key exchange is secure in presence of eavesdropping adversary.
  - Diffie-Hellman secure against eavesdroppers in large groups $(Z_p)^*$, p prime

# Insecure against man-in-the-middle

As described, the protocol is insecure against **active** attacks



**Alice** **MiTM** **Bob**

$A \leftarrow g^{a} \bmod p$

a’

$A' \leftarrow g^{a'} \bmod p$

$\mathrm{B}' \leftarrow g^{b'} \bmod p$

b’

$\mathrm{B} \leftarrow g^{b} \bmod p$

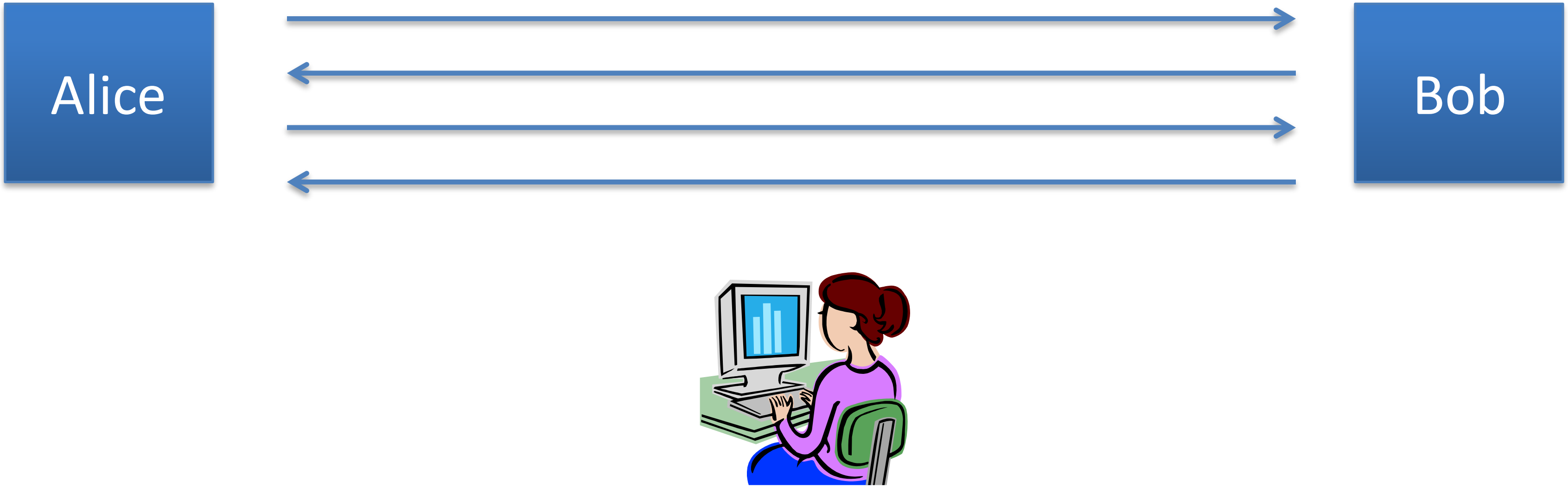
$g^{ab'}$ $g^{ab'}$ $g^{a'b}$ $g^{a'b}$

Attacker relays traffic from Alice to Bob and reads it in clear

# Another solution

Goal: Alice and Bob want shared secret, unknown to eavesdropper

- For now: security against eavesdropping only (no tampering)



# The idea

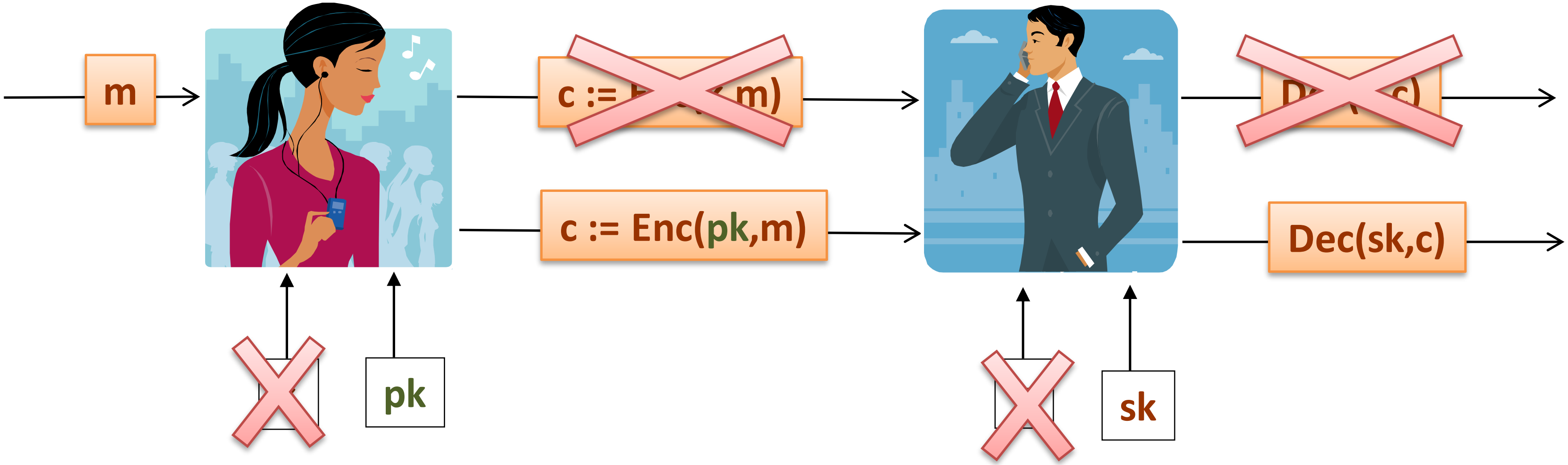
Instead of using one key **k**, use **2** keys **(pk,sk)**, where

**pk** is used for **encryption**,

**sk** is used for **decryption**.

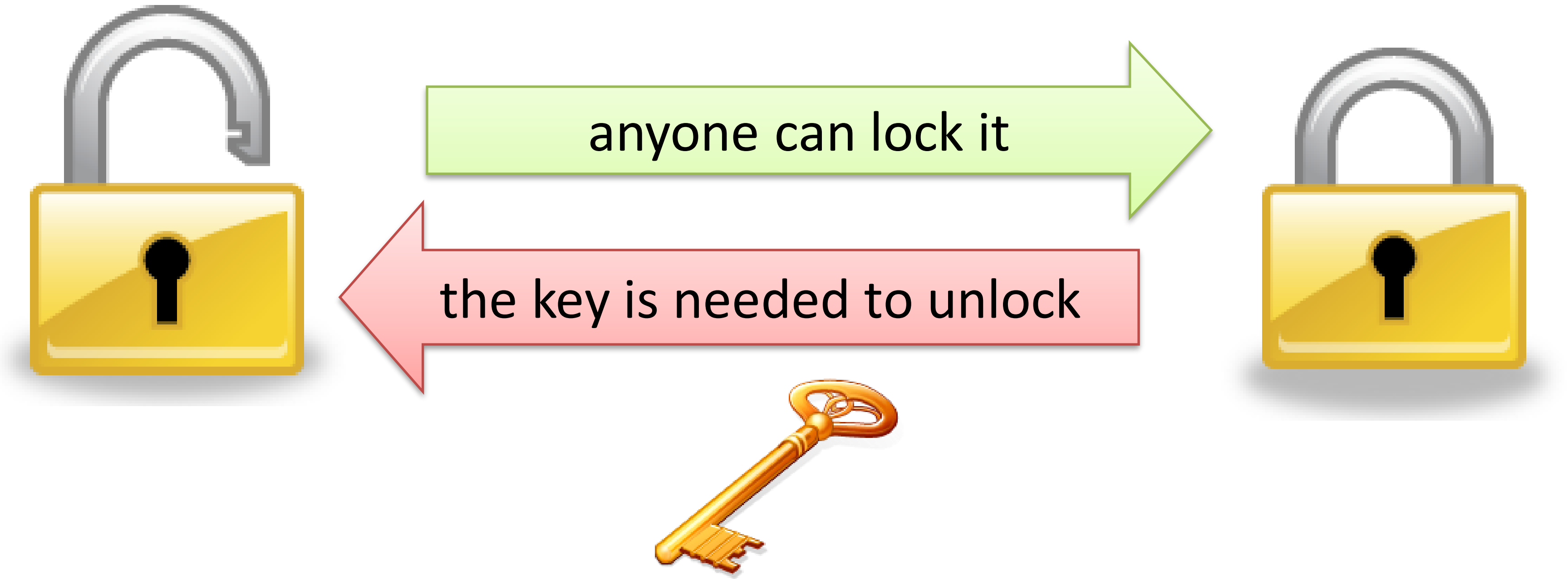
**pk** can be public, and only **sk** has to be kept secret!

That's why it's called: **public-key cryptography**



# Analogy

Examples padlocks:



anyone can lock it

the key is needed to unlock

# Public key encryption

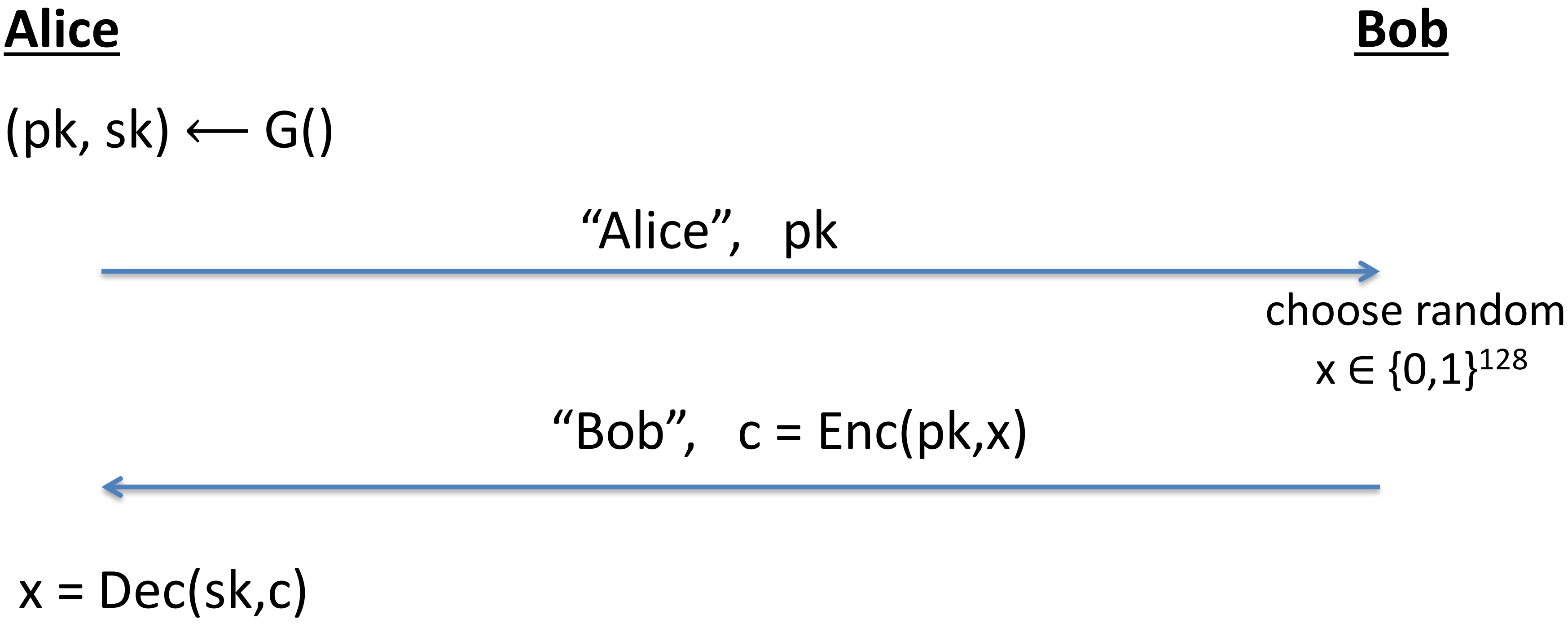
**Definition**: a public-key encryption system is a triple of algs. (Gen, Enc, Dec)

- Gen(): randomized alg. outputs a key pair (pk, sk)
- Enc(pk, m): randomized alg. that takes m∈M and outputs c ∈C
- Dec(sk,c): det. alg. that takes c∈C and outputs m∈M or ⊥

Correctness: ∀(pk, sk) output by G :

∀m∈M: Dec(sk, Enc(pk, m) ) = m

# Establishing a shared secret



**Alice**

**Bob**

$(pk, sk) \longleftarrow G()$

"Alice", pk

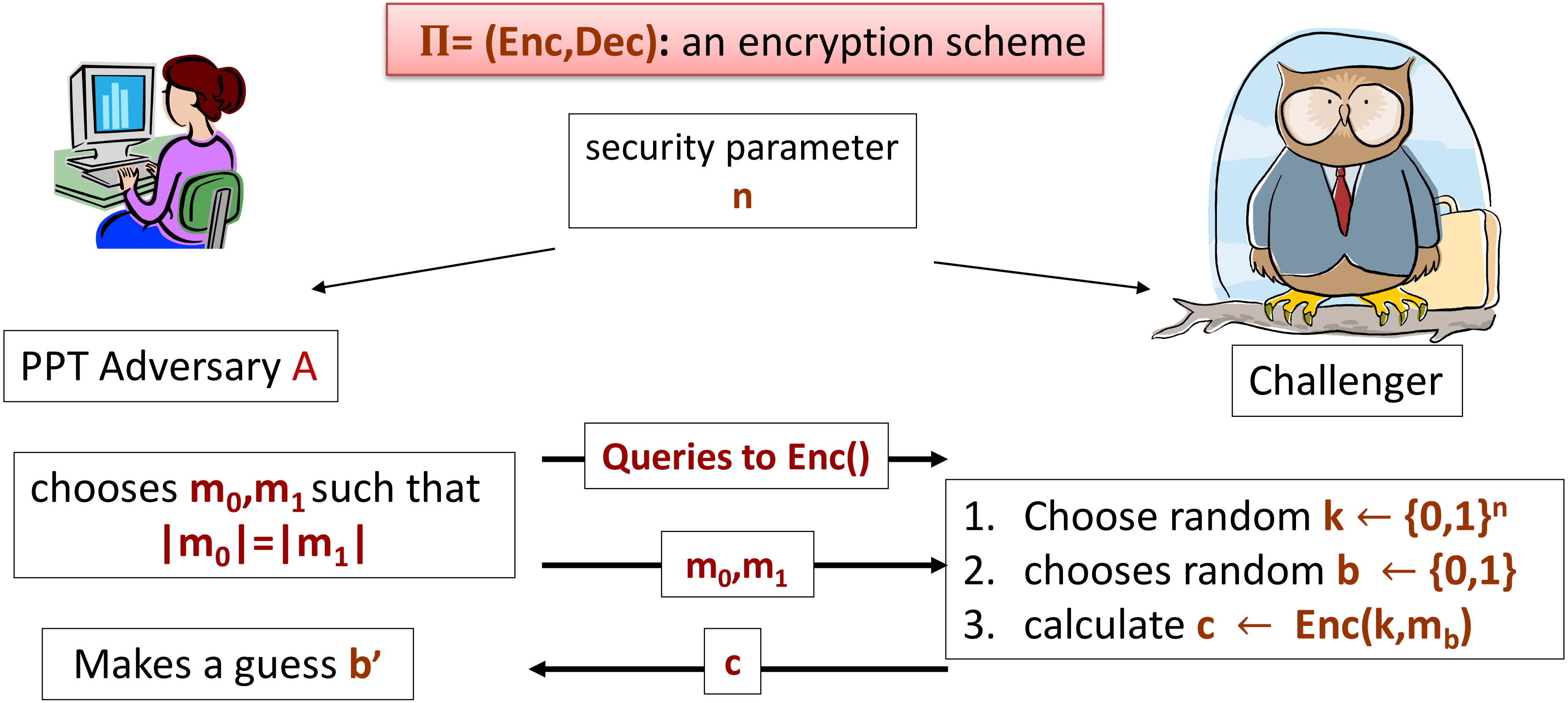
choose random<br>
$x \in \{0,1\}^{128}$

"Bob", $c = Enc(pk,x)$

$x = Dec(sk,c)$

x: shared secret

# CPA Security Game – Secret key



**Π= (Enc,Dec)**: an encryption scheme

security parameter
**n**

PPT Adversary A

Challenger

chooses $\mathbf{m_0,m_1}$ such that
$\mathbf{|m_0|=|m_1|}$

**Queries to Enc()**

$\mathbf{m_0,m_1}$

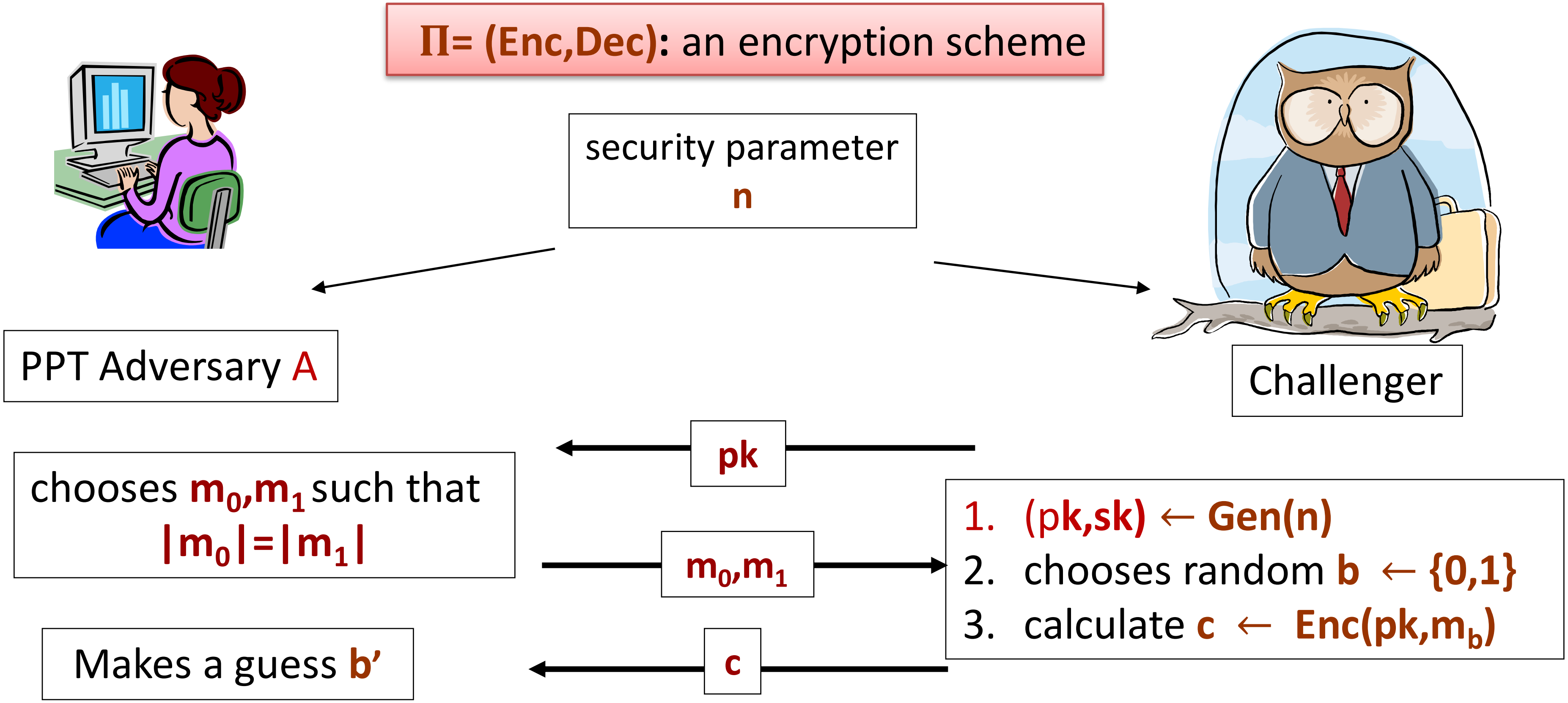
1. Choose random **k** ← **$\{0,1\}^n$**
2. chooses random **b** ← **{0,1}**
3. calculate **c** ← **$\mathrm{Enc}(k,m_b)$**

**c**

Makes a guess **b'**

**Security definition**:
We say that **(Enc,Dec)** is **CPA-secure** if any **polynomial time** adversary,
**| Pr[ b'=b ] - ½ |** is negligible in n.

# CPA Security Game – Public key



**Π= (Enc,Dec)**: an encryption scheme

security parameter
**n**

PPT Adversary A

Challenger

**pk**

chooses $\mathbf{m_0,m_1}$ such that
$\mathbf{|m_0|=|m_1|}$

$\mathbf{m_0,m_1}$

1. **(pk,sk)** ← **Gen(n)**
2. chooses random **b** ← **{0,1}**
3. calculate **c** ← **Enc(pk,$\mathbf{m_b}$)**

Makes a guess **b’**

**c**

**Security definition**:
We say that **(Enc,Dec)** is **CPA-secure** if any **polynomial time** adversary,
**| Pr[ b’=b ] - ½ |** is negligible in n.

# CPA security definition

- Experiment $\mathrm{Exp}_{\Pi,A}^{\mathrm{CPA}}(n)$:
  1. Choose $(pk, sk) \leftarrow^{R} Gen(1^n)$
  2. $m_0, m_1 \leftarrow A_1\ (pk)$
  3. $b \leftarrow^{R} \{0,1\}; c \leftarrow Enc_{pk}(m_b)$
  4. $b' \leftarrow A_2\ (pk, m_0, m_1, c)$
  5. Output 1 if $b = b'$ and 0 otherwise

We say that **(Enc,Dec)** is **chosen-plaintext attack (CPA) secure** if

For every **PPT** adversary $A = (A_1, A_2)$:

**|Pr[$\mathrm{Exp}_{\Pi,A}^{\mathrm{CPA}}(n)$ = 1]- ½ | negligible in n**

# Security (eavesdropping)

Adversary sees **pk, E(pk, x)** and wants **x** ∈M

CPA security ⇒

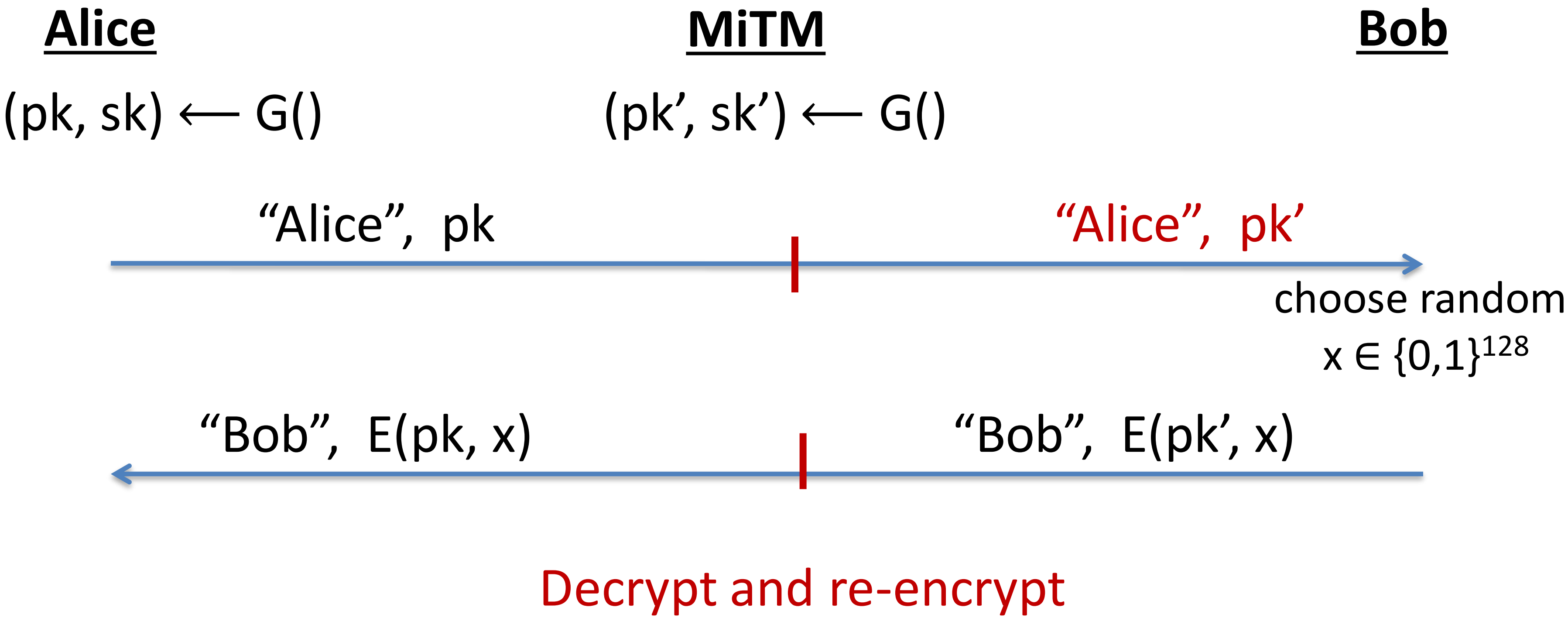
Adversary cannot distinguish

{ pk, E(pk, x) } from { pk, E(pk, r)}, r is random ∈ M

How about man-in-the-middle attacks?

# Insecure against man in the middle

As described, the protocol is insecure against **active** attacks



**Alice**

$(pk, sk) \longleftarrow G()$

**MiTM**

$(pk', sk') \longleftarrow G()$

**Bob**

"Alice", pk

"Alice", pk'

choose random

$x \in \{0,1\}^{128}$

"Bob", E(pk, x)

"Bob", E(pk', x)

Decrypt and re-encrypt

# Key insights

- Efficient algorithms to generate long primes
  - Miller-Rabin primality test
- Key distribution
  - Using key distribution centers (KDC) to establish fresh session keys
  - Based on authenticated encryption
- Key distribution without trusted servers
  - Diffie-Hellman (based on difficulty of computing discrete logs in cyclic groups)
  - Public-key encryption

# Acknowledgement

Some of the slides and slide contents are taken from
http://www.crypto.edu.pl/Dziembowski/teaching
and fall under the following:

We have also used slides from Prof. Dan Boneh online cryptography course at Stanford University:

http://crypto.stanford.edu/~dabo/courses/OnlineCrypto/