ENEE 457: Computer Systems Security

Lecture 6
Public Key Crypto II: RSA, ElGamal, Diffie-Hellman

Charalampos (Babis) Papamanthou

Department of Electrical and Computer Engineering
University of Maryland, College Park

How to distribute the cryptographic keys?

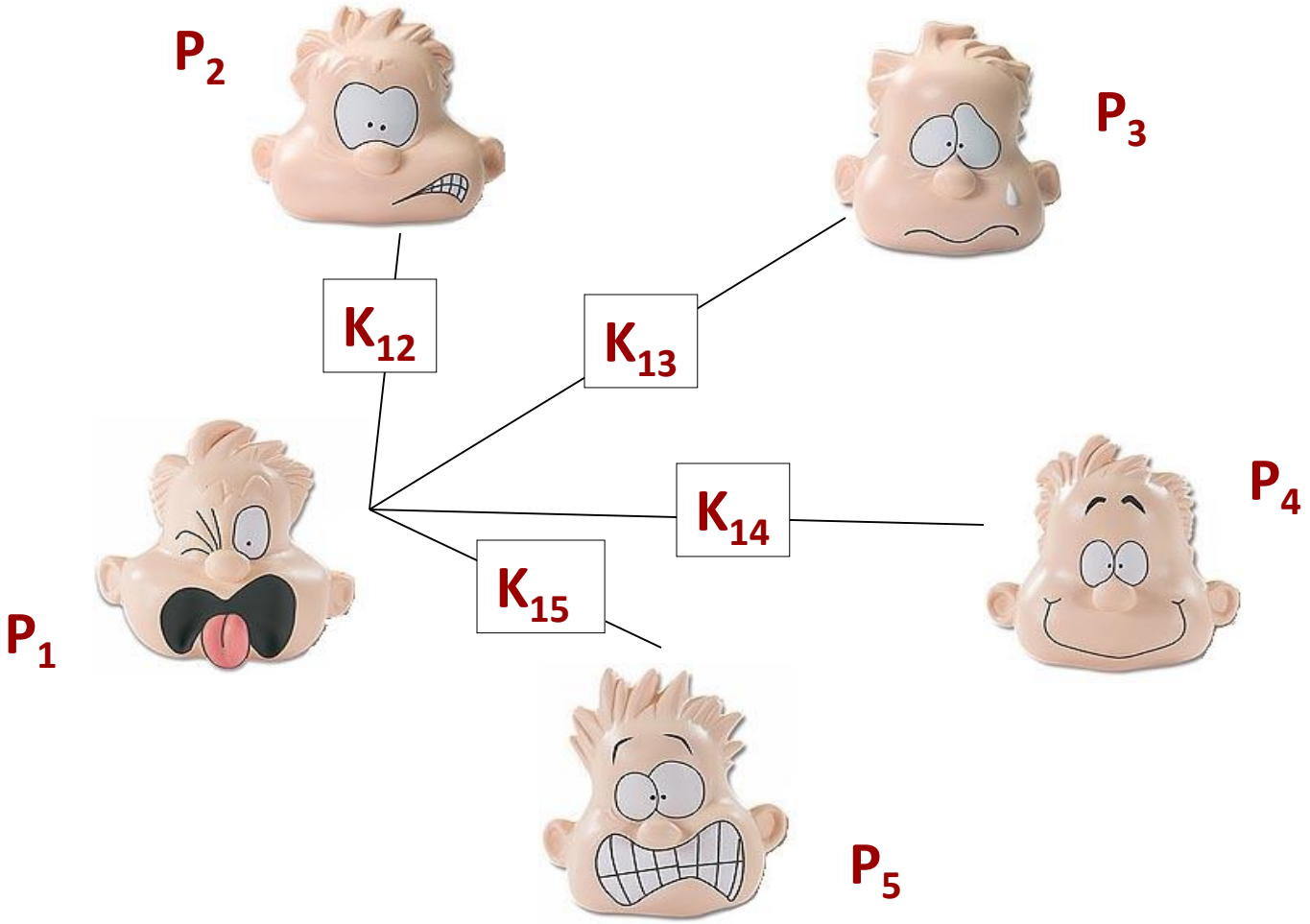
- If the users can meet in person beforehand – **it's simple.**

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

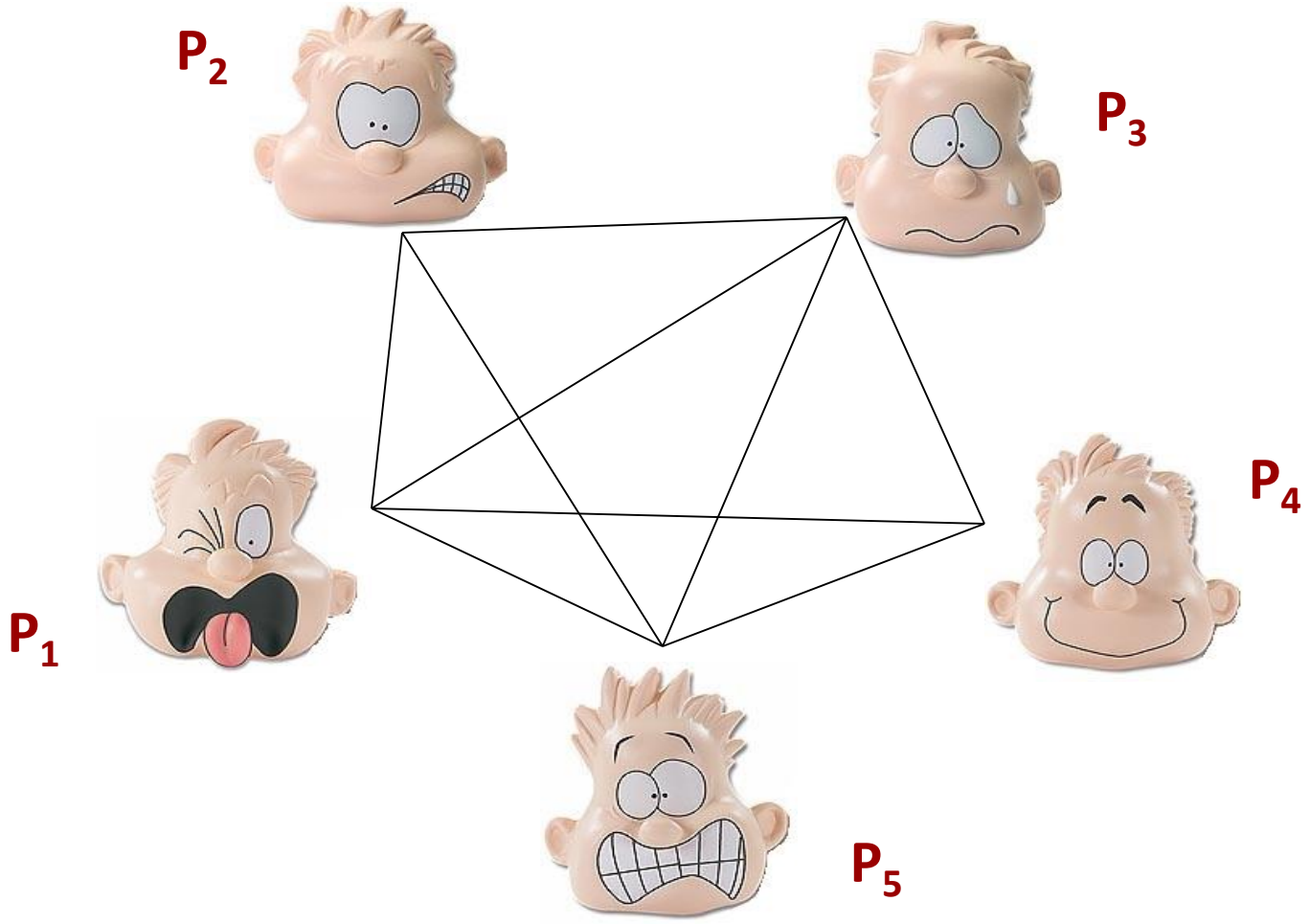
  (a typical example: on-line shopping)

A naive solution:

give to every user P_i a separate key K_{ij} to communicate with every P_j



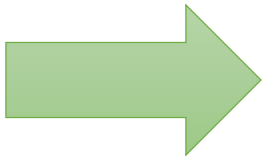
In general:
a quadratic number of keys is needed



Problems:

- Someone (a **Key Distribution Center, KDC**) needs to "give the keys"
  - **feasible** if the users are e.g. working in one company
  - **infeasible** on the internet
  - relies on the honesty of **KDC**
  - **KDC** needs to be permanently available
  - ...
- The users need to store large numbers of keys **in a secure way**

Plan



1. The problem of key distribution
2. The idea of Merkle, Diffie and Hellman
3. The solution of Rivest, Shamir and Adleman

The solution:

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/)

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

The idea

Instead of using one key **K**,

- use **2** keys **(pk,sk)**, where
  - **pk** is used for **encryption**,
  - **sk** is used for **decryption**, or
  - **sk** is used for **computing a tag**,
  - **pk** is used for **verifying correctness of the tag**.

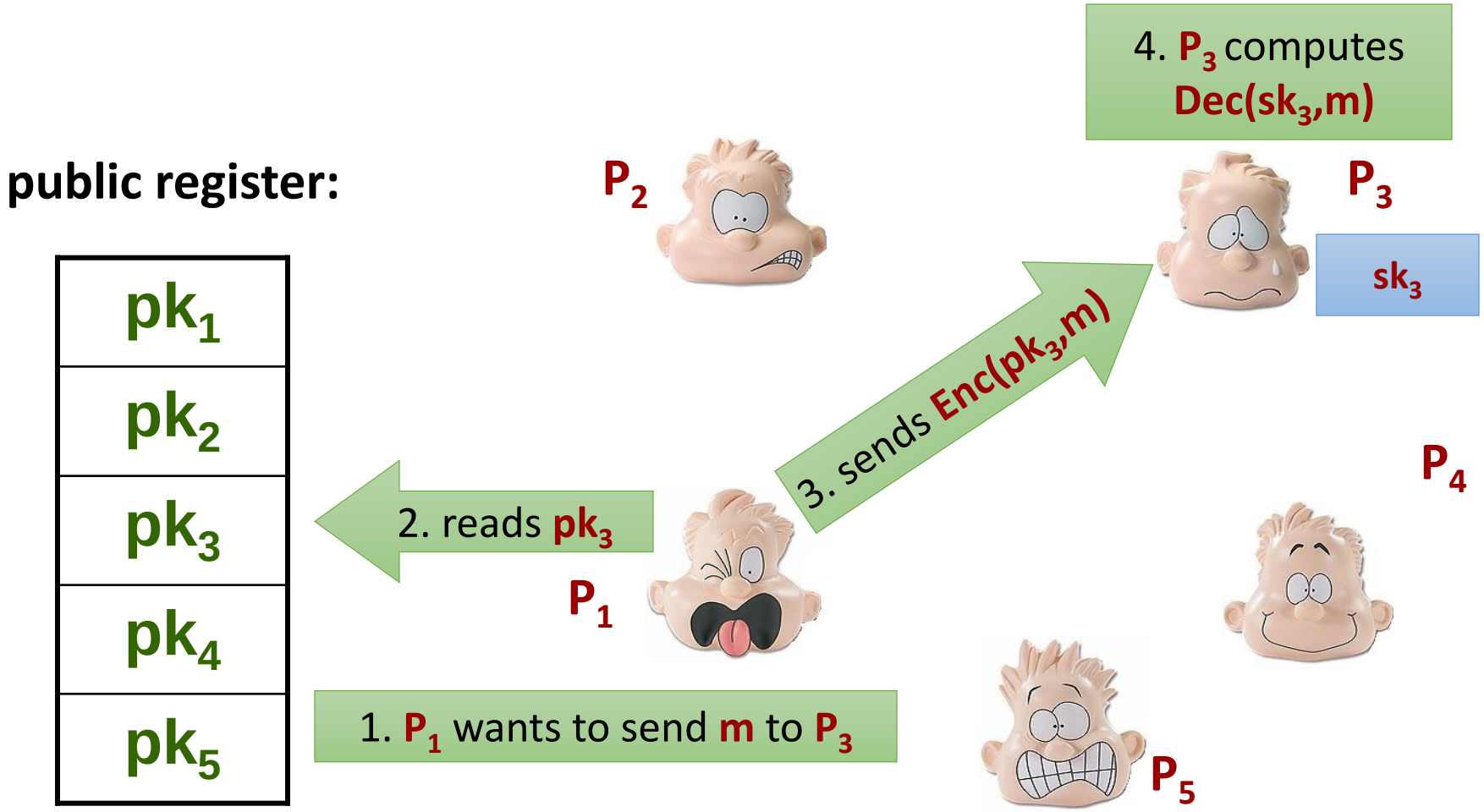
this will be called "signatures"

Sign – the signing algorithm

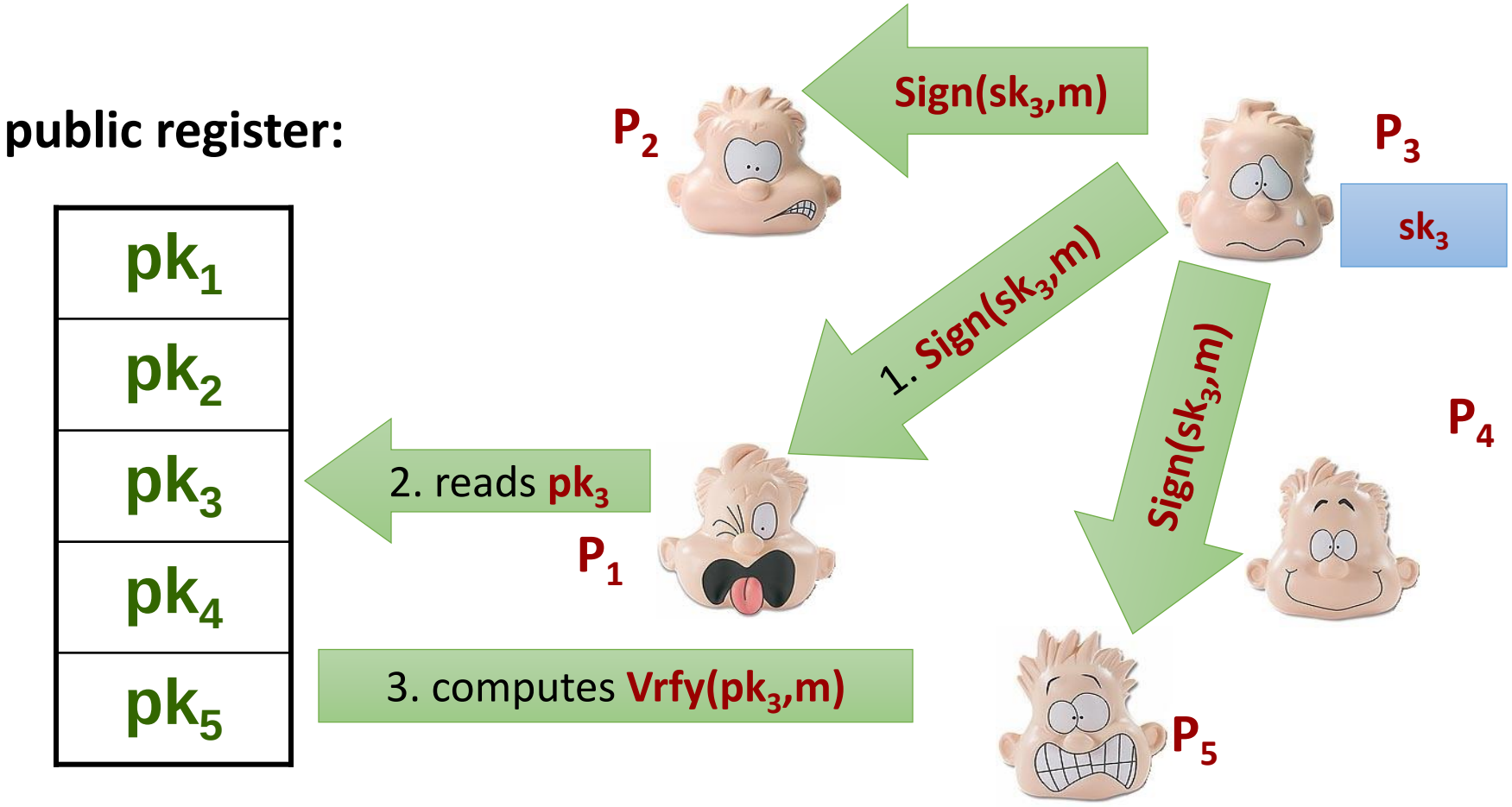
Moreover: **pk** can be public, and only **sk** has to be kept secret!

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

Anyone can send encrypted messages to anyone else

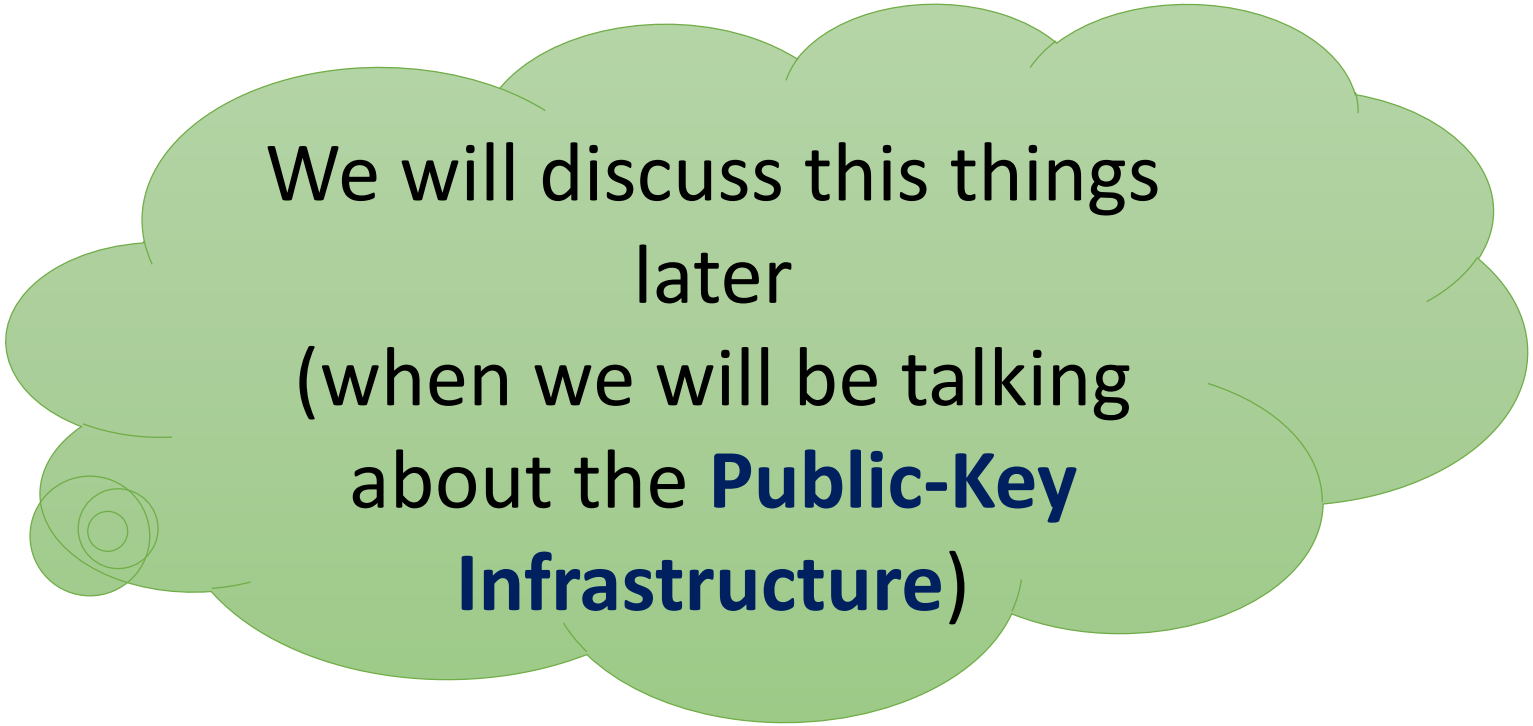


Anyone can verify the signatures



Things that need to be discussed

- Who maintains “the register”?
- How to contact it securely?
- How to revoke the key (if it is lost)?
- ...

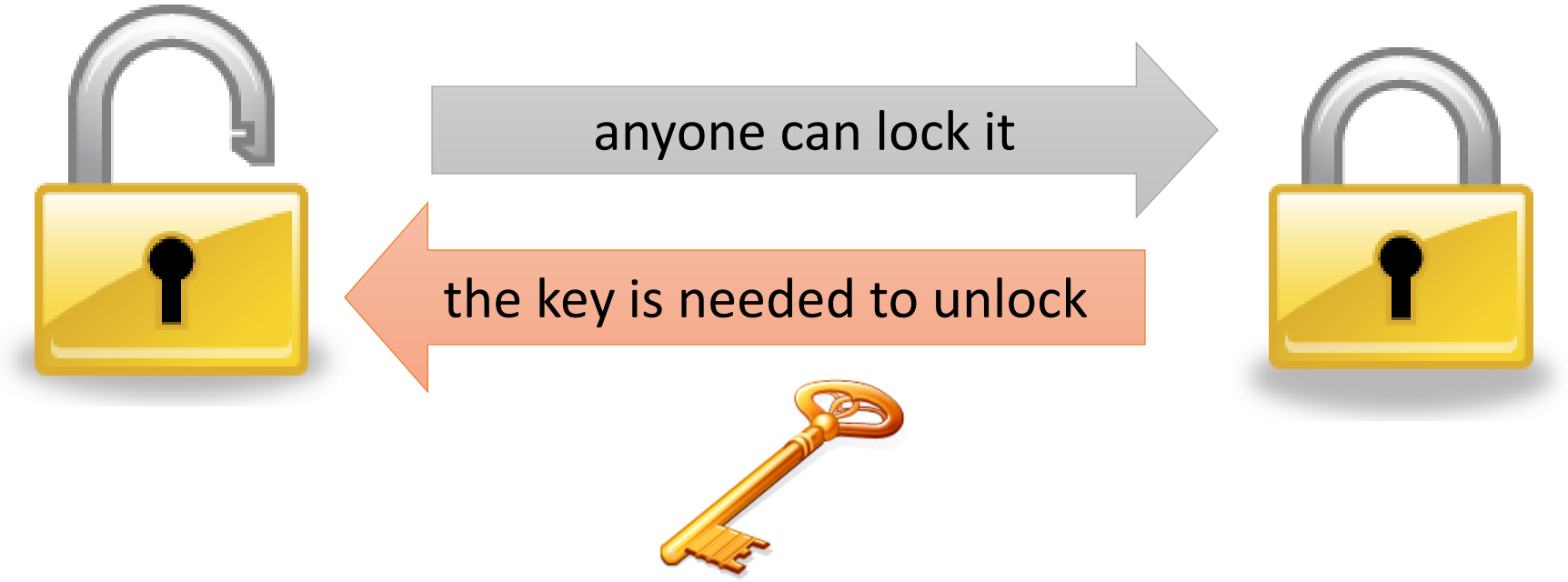


But is it possible?

In “physical world”: yes!

Examples:

1. “normal” signatures
2. padlocks:



Diffie and Hellman (1976)

- Diffie and Hellman proposed the public key cryptography in **1976**.
- They won the Turing award for that work in 2016 (Turing award is considered to be the Nobel Prize for Computer Science)

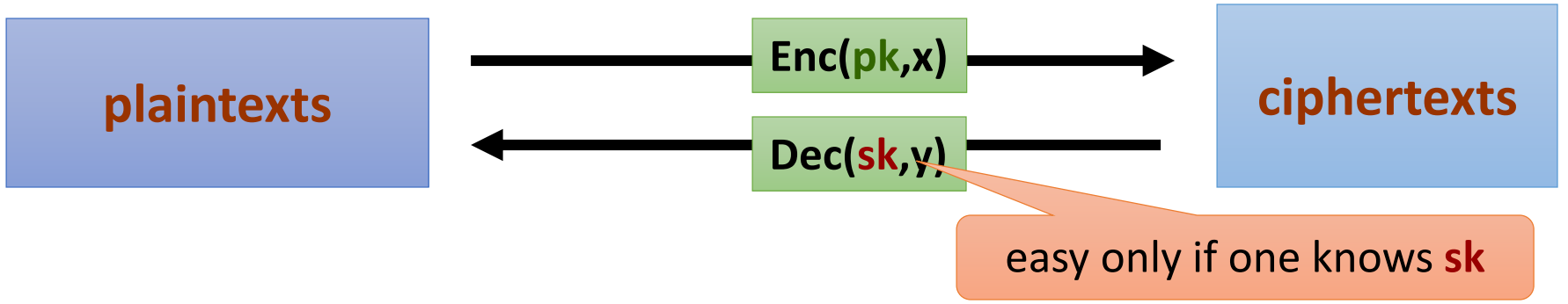
- They just proposed the **concept**, not the **implementation**.

- They have also shown a protocol for **key-exchange**.

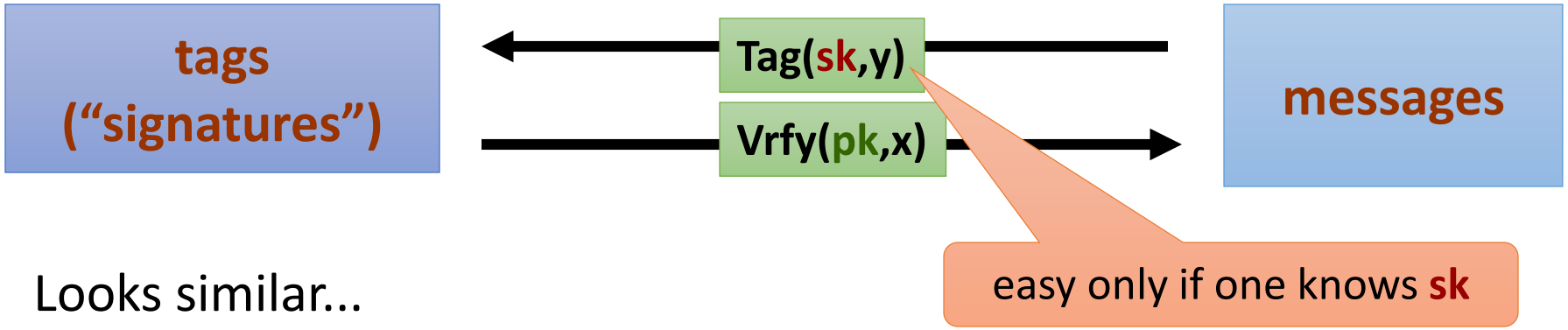
The observation of Diffie and Hellman:

(pk, sk) – the key pair

public-key encryption:



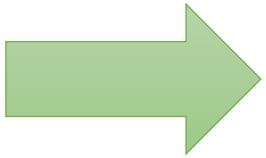
signature schemes:



Looks similar...

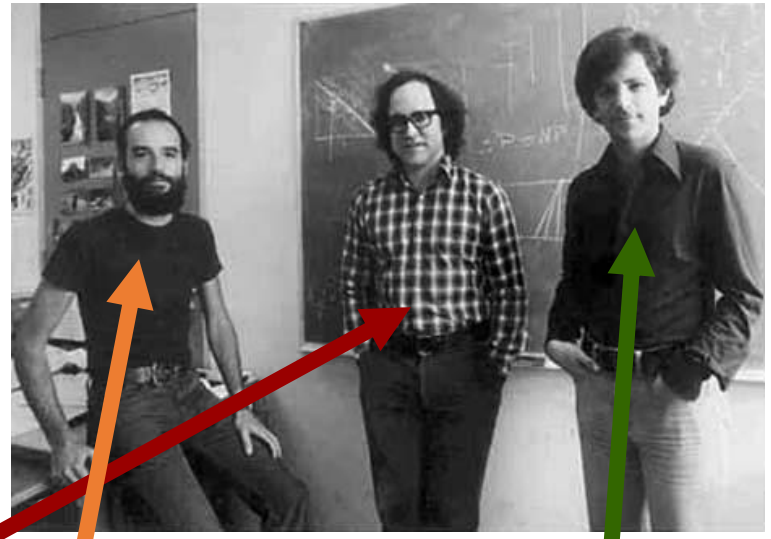
Plan

1. The problem of key distribution
2. The idea of Merkle, Diffie and Hellman
3. The solution of Rivest, Shamir and Adleman



Do such functions exist?

Yes: exponentiation modulo **N**, where **N** is a product of two large primes.



Ron Rivest, Adi Shamir, and Leonard Adleman (1977)

RSA function is (conjectured to be) a trapdoor permutation!

The RSA function

$N = pq$, such that p and q are primes,
and $|p| = |q|$

$\phi(N) = (p-1)(q-1)$.

e is such that $\gcd(e, \varphi(N)) = 1$
d is such that $ed = 1 \pmod{\varphi(N)}$

pk := (N,e)
sk := (N,d)

$Enc_{pk}: Z_N^* \rightarrow Z_N^*$ is defined as:

$$Enc_{pk}(m) = m^e \bmod N.$$

$Dec_{sk}: Z_N^* \rightarrow Z_N^*$ is defined as:

$$Dec_{sk}(c) = c^d \bmod N.$$

An observation

From the previous lecture we know that

- $\mathbf{Enc_{pk}: Z_N^* \rightarrow Z_N^*}$ is a permutation and
- $\mathbf{Dec_{sk}: Z_N^* \rightarrow Z_N^*}$ is its inverse.

Fact

$\mathbf{Enc_{pk}}$ is also a permutation over $\mathbf{Z_N}$ and $\mathbf{Dec_{sk}}$ is its inverse.

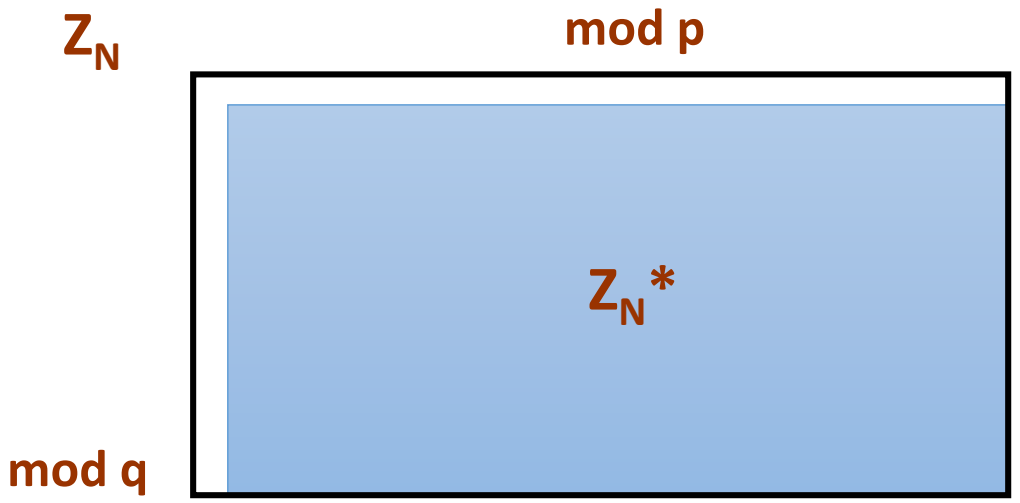
In fact, this doesn't even matter that much because:

if one finds an element $\mathbf{a \in Z_N \setminus Z_N^*}$ then one can factor **N**, because:

$$\mathbf{gcd(a, N) > 1.}$$

So, finding such an element is as hard as factoring **N**.

A proof of the fact from the previous slide



Suppose $x = 0 \bmod p$.

Then (trivially) $(x^e)^d = x \bmod p$

On the other hand: $(x^e)^d = x^{ed} = x^1 \bmod q$

CRT

$(x^e)^d = x \bmod N$

because: $ed = 1 \bmod (p-1)(q-1)$,
and therefore $ed = 1 \bmod (q-1)$

QED

Is RSA secure?

Is RSA secure:

1. as an encryption scheme?
2. as a signature scheme?

The answer is not that simple.

First, we need to define security!

We will do it on the next two lectures.

- Slides adjusted from:
  - http://dziembowski.net/Teaching/BISS09/

The “handbook RSA encryption”

N = pq - **RSA** modulus

e is such that **gcd(e, φ(N)) = 1**,
d is such that **ed = 1 (mod ϕ(N))**

$$\mathbf{Enc_{(e,N)}(m) = m^e \bmod N},$$

and $\mathbf{Dec_{(d,N)}(c) = c^d \bmod N.}$

Problems

$\mathbf{Enc_{pk}}$ is deterministic, so:

if one encrypts twice the same message then the ciphertexts are the same

Therefore if the message space **M** is small, the adversary can check all possible messages:

given a ciphertext **c** do:
for every **m ϵ M** check if $\mathbf{Enc_{pk}(m) = c}$

for example if **M={yes,no}**, then the encryption is not secure.

RSA has some "algebraic properties".

Algebraic properties of RSA

1. **RSA** is homorphic:

$$\mathrm{Enc}_{(e,N)}(m_0 \cdot m_1) = (m_0 \cdot m_1)^e$$
$$= m_0^e \cdot m_1^e$$
$$= \mathrm{Enc}_{(e,N)}(m_0) \cdot \mathrm{Enc}_{(e,N)}(m_1)$$

   why is it bad?

   By checking if

$$c_0 \cdot c_1 = c$$

   the adversary can detect if

$$\mathrm{Dec}_{(d,N)}(c_0) \cdot \mathrm{Dec}_{(d,N)}(c_1) = \mathrm{Dec}_d(c)$$

Question: Is RSA secure?

Looks like it has some weaknesses...

Plan:

1. Provide a formal security definition.
2. Modify **RSA** so that it is secure according to this definition.

A mathematical view

A **public-key encryption (PKE)** scheme is a triple **(Gen, Enc, Dec)** of poly-time algorithms, where

- **Gen** is a **key-generation** randomized algorithm that takes as input a security parameter 1^n and outputs a key pair **(pk,sk)**.
- **Enc** is an **encryption** algorithm that takes as input the **public key pk** and a **message m**, and outputs a **ciphertext c**,
- **Dec** is an **decryption** algorithm that takes as input the **private key pk** and the **ciphertext c**, and outputs a **message m'**.

We will sometimes write $Enc_{pk}(m)$ and $Dec_{sk}(c)$ instead of **Enc(pk,m)** and **Dec(sk,c)**.

Correctness

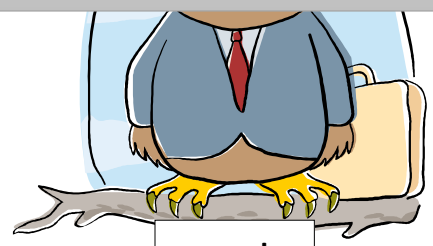
$P(Dec_{sk}(Enc_{pk}(m)) \neq m)$ is negligible in **n**

A simplified view

security parameter
1^n

1. selects random
   $(pk,sk) = Gen(1^n)$
2. chooses a random $b = 0,1$

pk



oracle

challenge phase:

chooses m_0,m_1

m_0,m_1

$c = Enc(pk,m_b)$

has to guess **b**

CPA-security

Alternative name: CPA-secure

Security definition:

We say that **(Gen,Enc,Dec) has indistinguishable encryptions under a chosen-plaintext attack (CPA)** if any

randomized polynomial time adversary

guesses **b** correctly

with probability at most **0.5 + ε(n),** where **ε** is negligible.

Is the “handbook RSA” secure?

the “handbook RSA”

$N = pq$ - RSA modulus

e is such that $\gcd(e,d) = 1$, d is such that $ed = 1 \pmod{\phi(N)}$

$\mathrm{Enc}_{(N,e)}(m) = m^e \bmod N$, and $\mathrm{Dec}_{(d,N)}(c) = c^d \bmod N$.

Not secure!

In fact:

No deterministic encryption scheme is secure.

How can the adversary win the game?

1. he just chooses any m_0, m_1 ,
2. computes $c_0 = \mathrm{Enc}(pk, m_0)$ himself
3. compares the result.

Moral: encryption has to be randomized.

Encoding

Therefore, before encrypting a message we usually **encode it** (adding some randomness).

This has the following advantages:

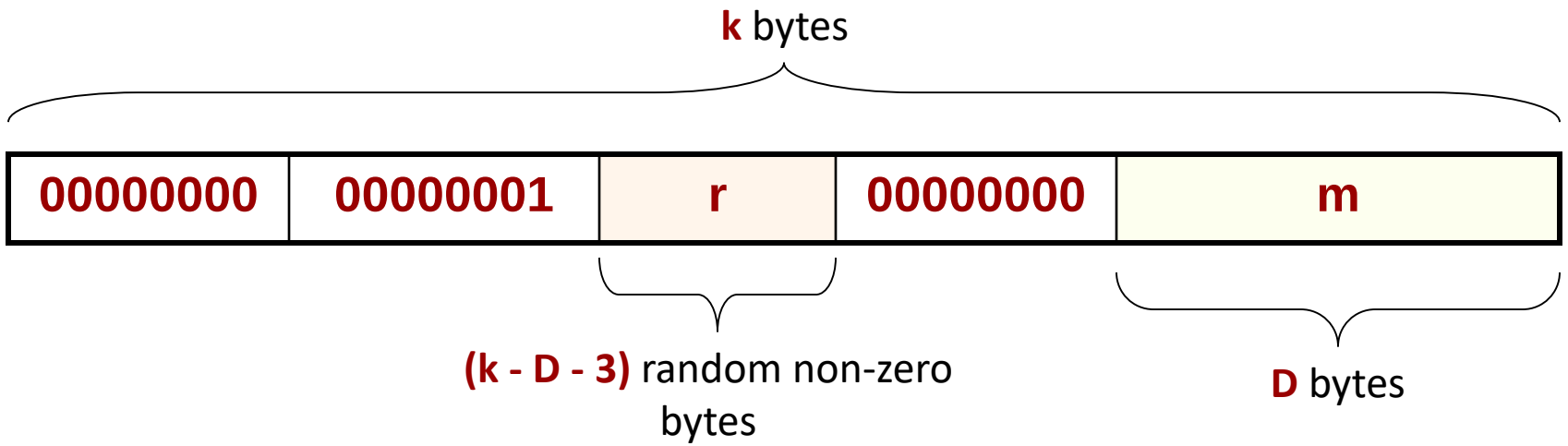
- makes the encryption non-deterministic
- breaks the “algebraic properties” of encryption.

How is it done in real-life?

PKCS #1: RSA Encryption Standard Version 1.5:

public-key: **(N,e)**
let **k** := length on **N** in bytes.
let **D** := length of the plaintext
requirement: **D ≤ k - 11**.

Enc((N,e), m) := x^e mod N, where **x** is equal to:



Security of the PKCS #1: RSA Encryption Standard Version 1.5.

It is **believed** to be CPA-secure.

It has however some weaknesses (it is not “chosen-ciphertext secure”).

Optimal Asymmetric Encryption Padding (OAEP) is a more secure encoding.

(we will not refer to that)

Algorithmic Issues

- The implementation of the RSA cryptosystem requires various algorithms
- Overall
  - Representation of integers of arbitrarily large size and arithmetic operations on them
- Encryption
  - Modular power
- Decryption
  - Modular power

- Setup
  - Generation of random numbers with a given number of bits (to generate candidates $\boldsymbol{p}$ and $\boldsymbol{q}$)
  - Primality testing (to check that candidates $\boldsymbol{p}$ and $\boldsymbol{q}$ are prime)
  - Computation of the GCD (to verify that $\boldsymbol{e}$ and $\phi(\boldsymbol{n})$ are relatively prime)
  - Computation of the multiplicative inverse (to compute $\boldsymbol{d}$ from $\boldsymbol{e}$)

Modular Power

- The repeated squaring algorithm speeds up the computation of a modular power $a^p \bmod n$
- Write the exponent p in binary

  $p = p_{b-1} p_{b-2} \cdots p_1 p_0$
- Then $a^p \bmod n = a^{p\_\{b-1\}2\^\{b-1\}+ p\_\{b-2\}2\^\{b-1\}+ \ldots +p\_12+ p\_0} \bmod n$
- We obtain

  $Q_b = a^p \bmod n$
- The algorithm performs $O(\log p)$ arithmetic operations

Decryption can be done with CRT

- Why is it more efficient in this way?

How to construct PKE based on the hardness of discrete log?

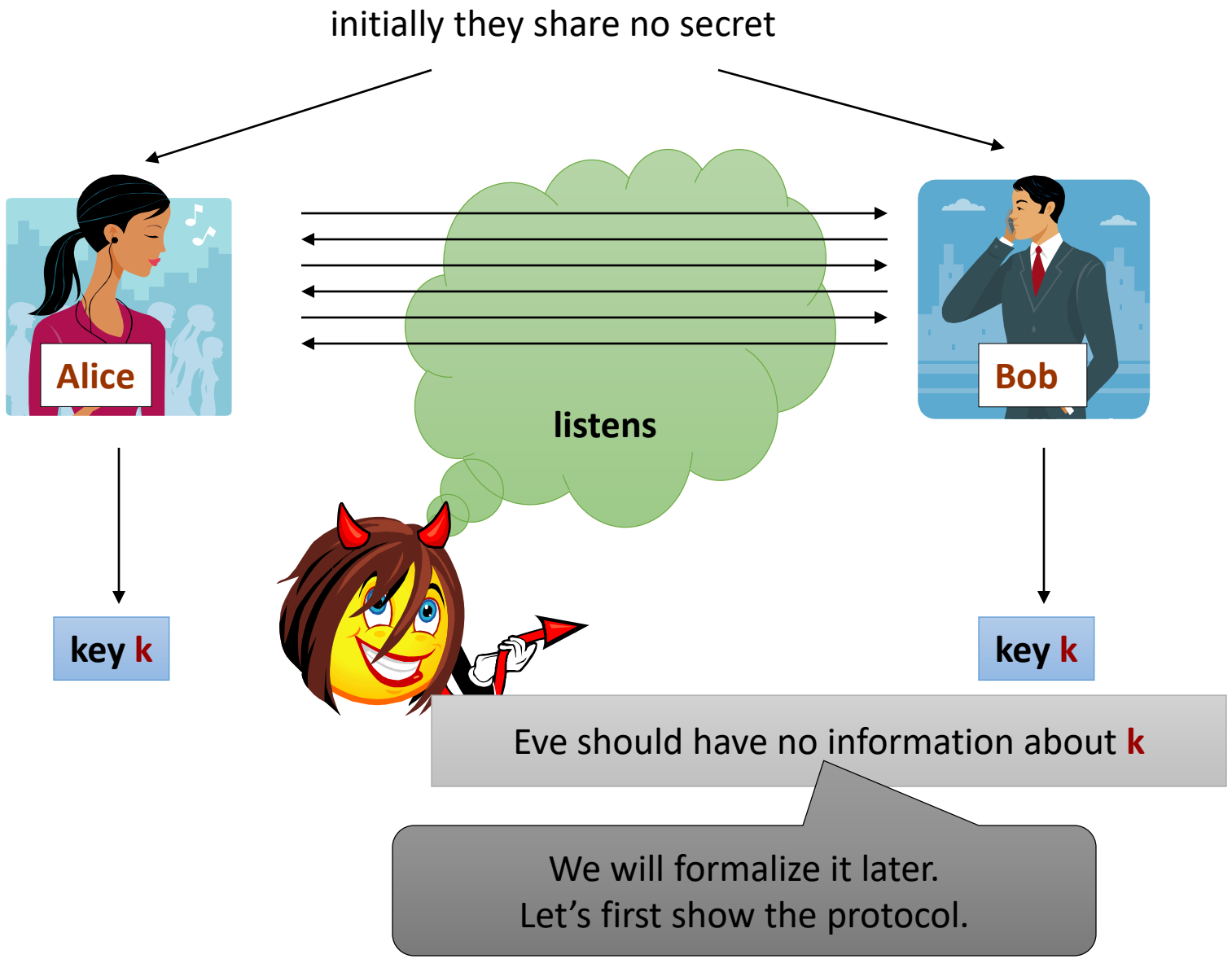
RSA was a trapdoor permutation, so the construction was quite easy...

In case of the **discrete log,** we just have a one-way function.

Diffie and Hellman constructed something weaker than PKE: a **key exchange protocol** (also called key **agreement** protocol).

We'll not describe it. Then, we'll show how to "convert it" into a **PKE**.

Key exchange

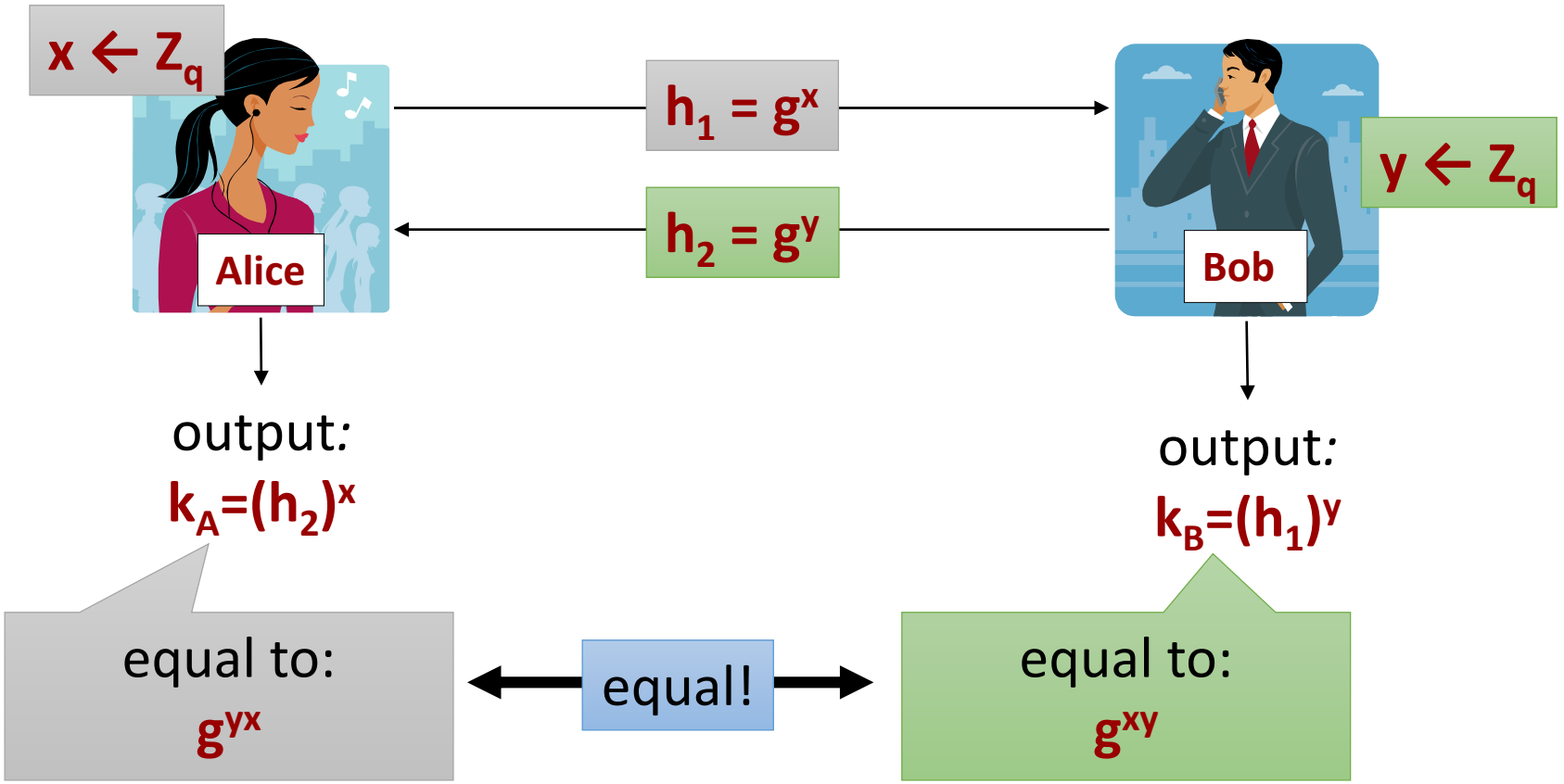


The Diffie-Hellman Key exchange

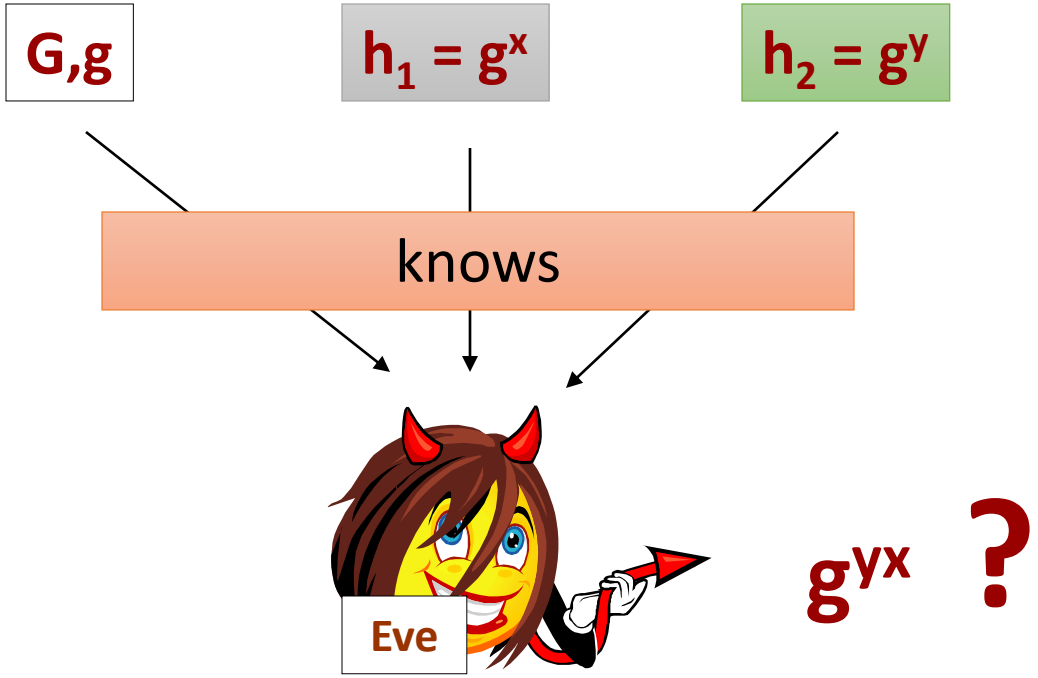
G – a group, where **discrete log is believed to be hard**

q = |G|

g – a generator of **G**



Security of the Diffie-Hellman exchange



Eve should have no information about g^{yx}

Is it secure?

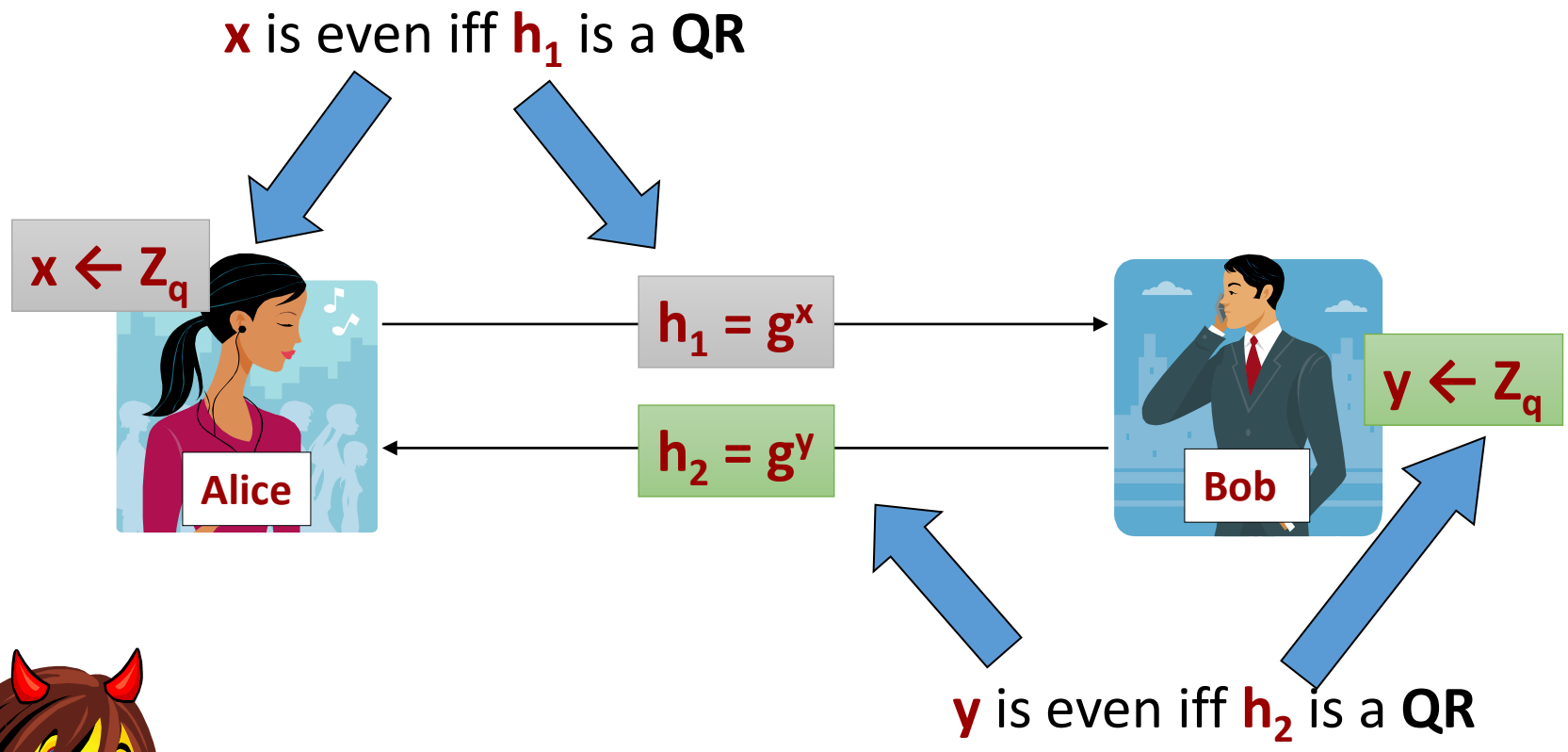
If the **discrete log in G** is easy then the **DH key exchange** is **not secure**.

(because the adversary can compute **x** and **y** from
$\mathbf{g}^{\mathbf{x}}$ and $\mathbf{g}^{\mathbf{y}}$)

If the discrete log in **G** is hard, then...

it may also not be secure

Example: $G = Z_p^*$



Therefore:

g^{yx} is a **QR** iff (h_1 is a **QR**) or (h_2 is a **QR**)

So, Eve can compute some information about g^{yx} (namely: if it is a **QR**, or not).

How to fix the previous problem?

- Intuitively: Pick only even numbers in the exponent!

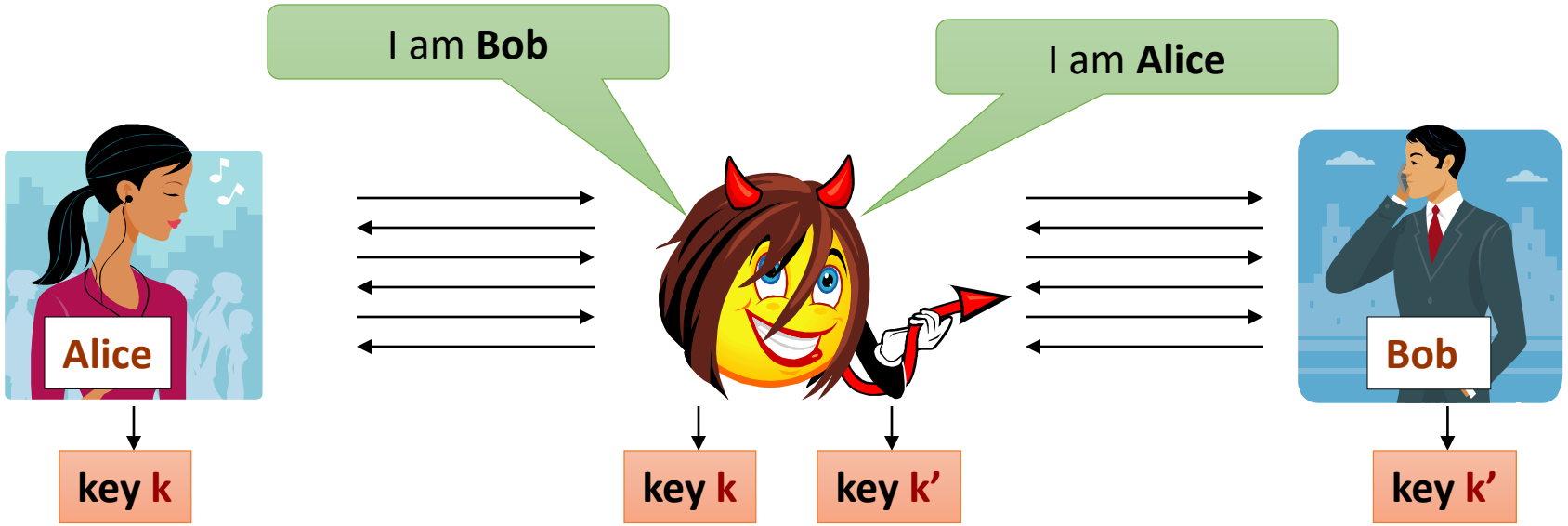
A problem

The protocols that we discussed are secure only against a **passive adversary** (that only eavesdrop).

What if the adversary is **active**?

She can launch a **man-in-the-middle** attack.

Man in the middle attack



A very realistic attack!

So, is this thing totally useless?
No! (it is useful as a building block)

Plan

1. Problems with the handbook RSA encryption
2. Security definitions
3. How to encrypt with RSA?
4. Encryption based on discrete-log
    1. first step: Diffie-Hellman key exchange
    2. ElGamal encryption
5. Public-key vs. private key encryption

El Gamal encryption

El Gamal is another popular public-key encryption scheme.

It is based on the **Diffie-Hellman** key-exchange.

First observation

Remember that the one-time pad scheme can be generalized to any group?

E.g.: $\mathcal{K} = \mathcal{M} = \mathcal{C} = G$.

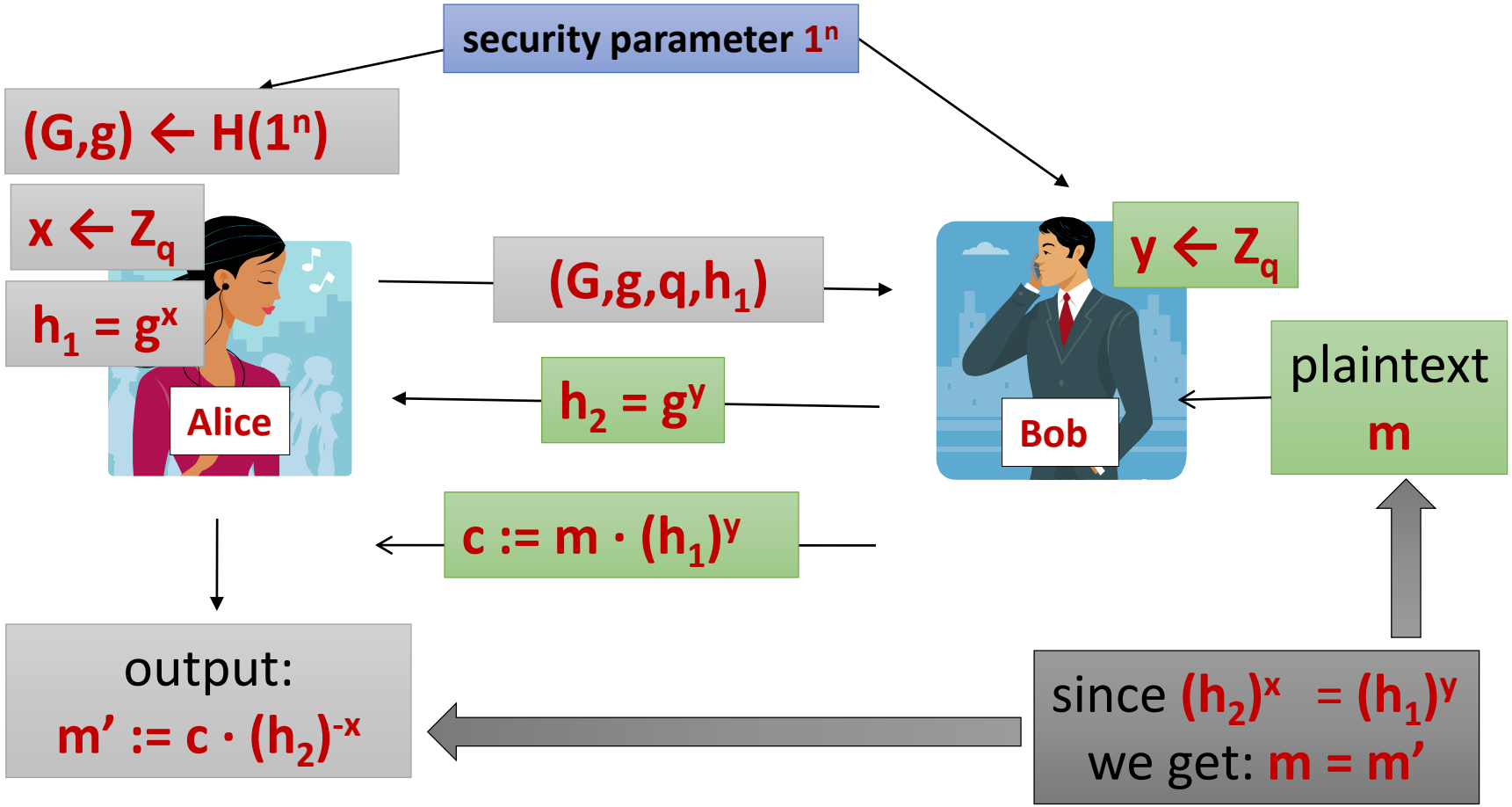
- $Enc(k,m) = m \cdot k$
- $Dec(k,m) = m \cdot k^{-1}$

So, if **k** is the key agreed in the **DH key exchange**, then

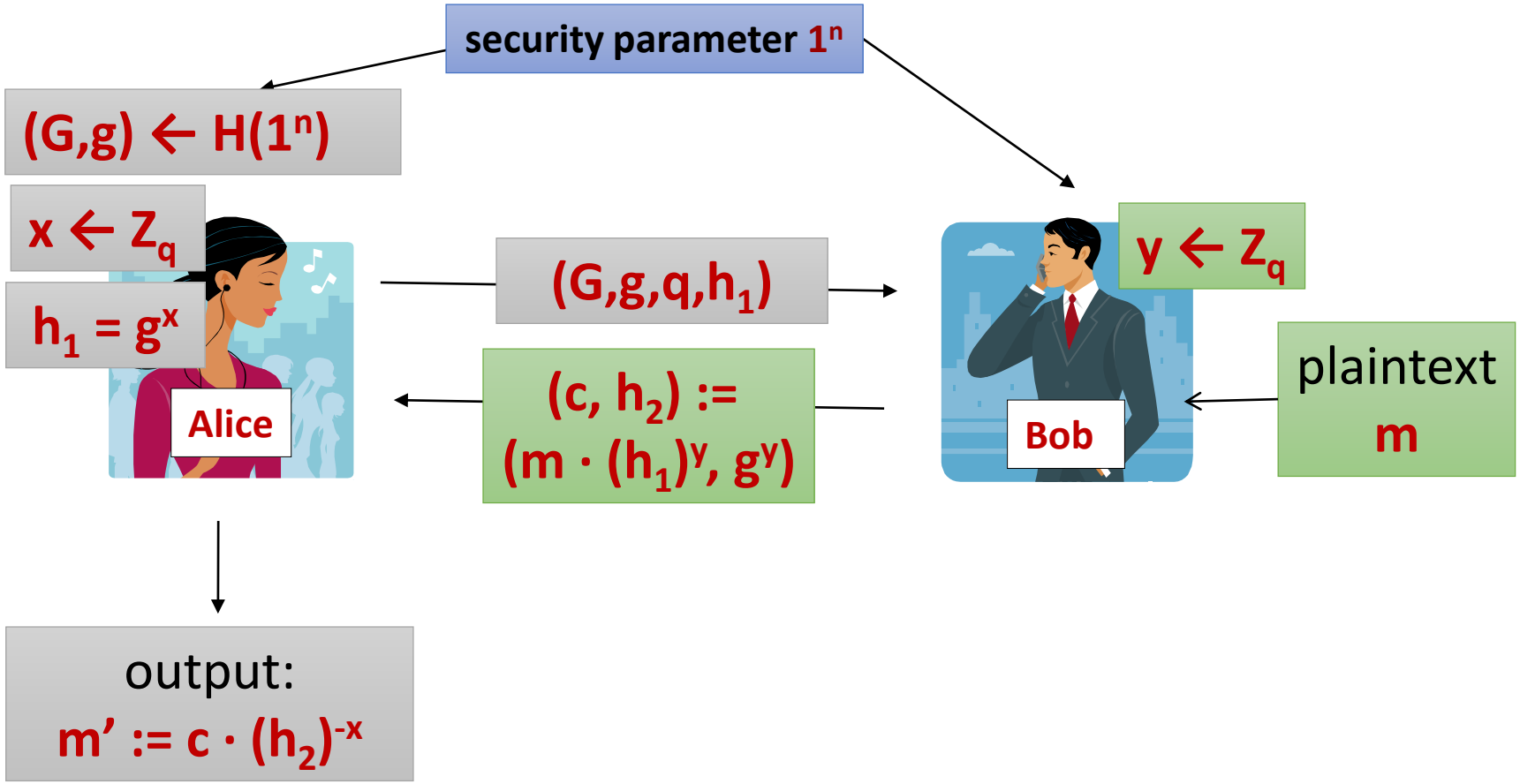
Alice can send a message $m \in G$ to Bob "encrypting it with **k**" by setting:

$$c := m \cdot k$$

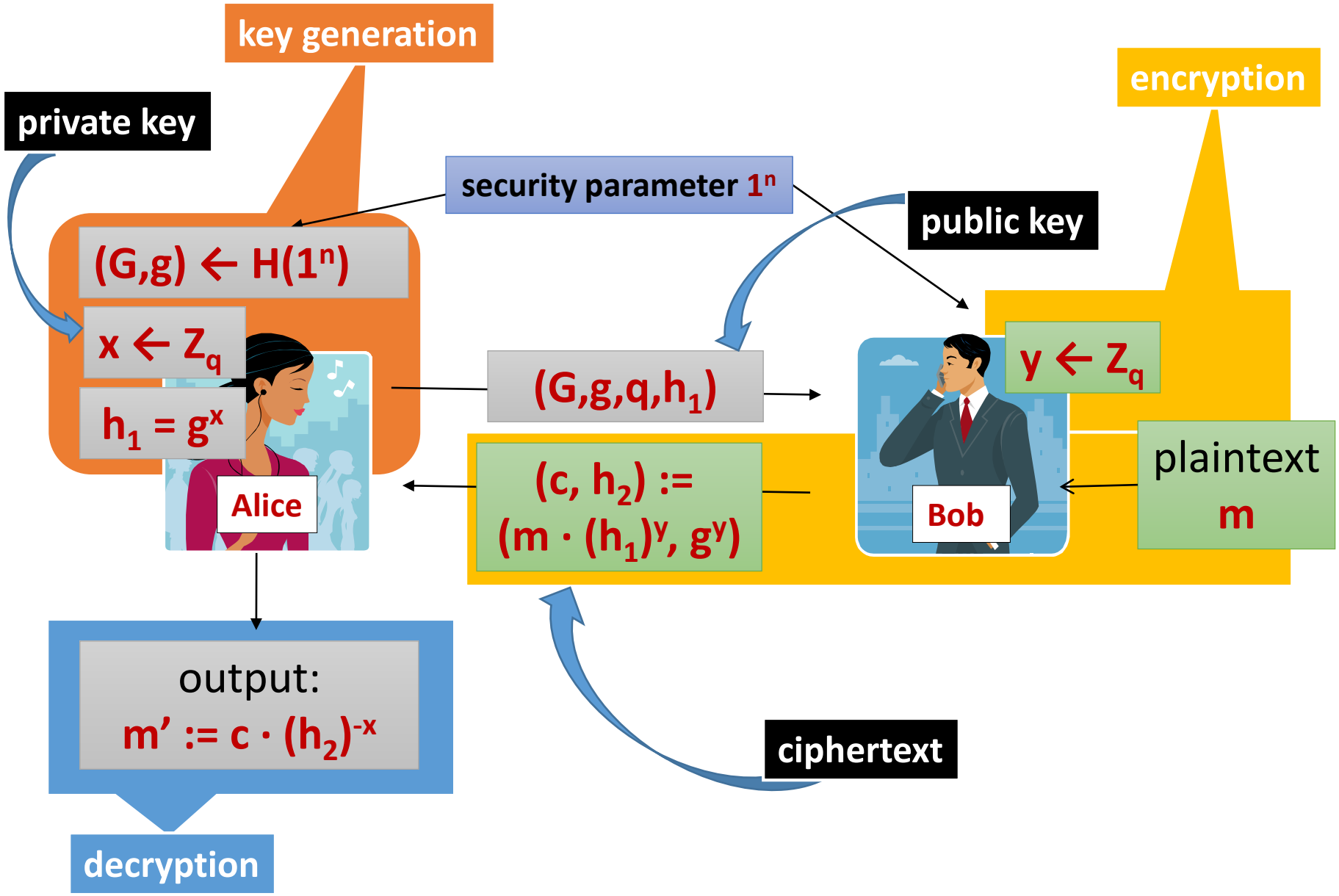
How does it look now?



The last two messages can be sent together



ElGamal encryption



El Gamal encryption

Let **H** be such that **DDH** is hard with respect to **H**.

Gen(1^n) first runs **H** to obtain **(G,q)** and **q**. Then, it chooses **$x \leftarrow Z_q$** and computes **$h := g^x$**. (note: it is **randomized by definition**)

The public key is **(G,g,q,h)**.
The private key is **(G,g,q,x)**.

$Enc((G,g,q,h), m) := (m \cdot h^y, g^y)$,
where **$m \in G$** and **y** is a random element of **G**

$Dec((G,g,q,x), (c_1,c_2)) := c_1 \cdot c_2^{-x}$

Correctness

$$h = g^x$$

$$Enc((G,g,q,h), m) = (m \cdot h^y, g^y)$$

$$\begin{aligned} Dec((G,g,q,x), (c_1,c_2)) &= c_1 \cdot c_2^{-x} \\ &= m \cdot h^y \cdot (g^y)^{-x} \\ &= m \cdot (g^x)^y \cdot (g^y)^{-x} \\ &= m \cdot g^{xy} \cdot g^{-yx} \\ &= m \end{aligned}$$

Public key vs. private key encryption

Private-key encryption has a following advantage:

it is much more efficient.

What we do in practice:

Use PKE to exchange secret key. Then use secret key to encrypt data.