ENEE 457: Computer Systems Security
10/3/16

Lecture 9
RSA Encryption and Diffie-Helmann Key Exchange

Charalampos (Babis) Papamanthou

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

- 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

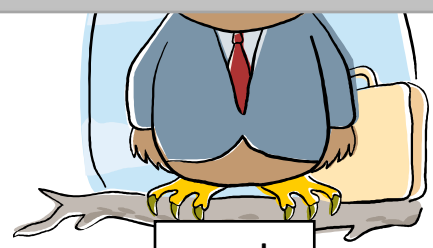
$P(Dec_{sk}(Enc_{pk}(m)) \neq m)$ is negligible in **n**

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:

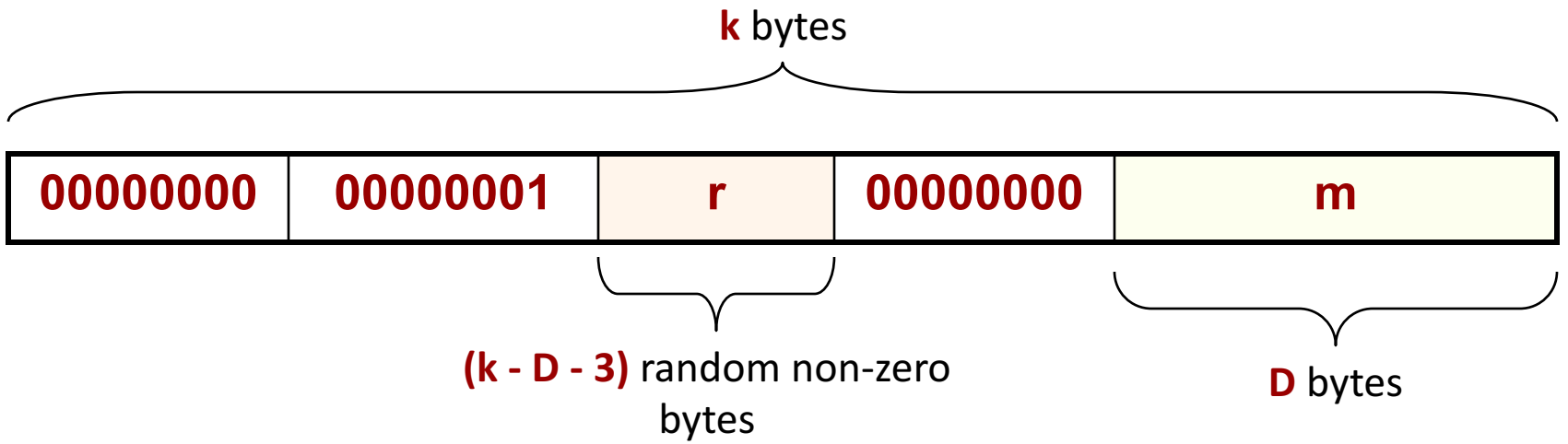
- makes the encryption non-deterministic
- breaks the “algebraic properties” of encryption.

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 p and q)
  - Primality testing (to check that candidates p and q are prime)
  - Computation of the GCD (to verify that e and $\phi(n)$ are relatively prime)
  - Computation of the multiplicative inverse (to compute d from 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

  $p = p_{b-1} p_{b-2} \cdots p_1 p_0$
- Start with

  $Q_1 = a^{p_{b-1}} \bmod n$
- Repeatedly compute

  $Q_i = ((Q_{i-1})^2 \bmod n) a^{p_{b-i}} \bmod n$
- We obtain

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

- Example
  - $3^{18} \bmod 19$ $(18 = 10010)$
  - $Q_1 = 3^1 \bmod 19 = 3$
  - $Q_2 = (3^2 \bmod 19)3^0 \bmod 19 = 9$
  - $Q_3 = (9^2 \bmod 19)3^0 \bmod 19 = 81 \bmod 19 = 5$
  - $Q_4 = (5^2 \bmod 19)3^1 \bmod 19 = (25 \bmod 19)3 \bmod 19 = 18 \bmod 19 = 18$
  - $Q_5 = (18^2 \bmod 19)3^0 \bmod 19 = (324 \bmod 19) \bmod 19 = 17 \cdot 19 + 1 \bmod 19 = 1$

p_{5-i}	1	0	0	1	0
$2^{p_{5-i}}$	3	1	1	3	1
Q_i	3	9	5	18	1

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?

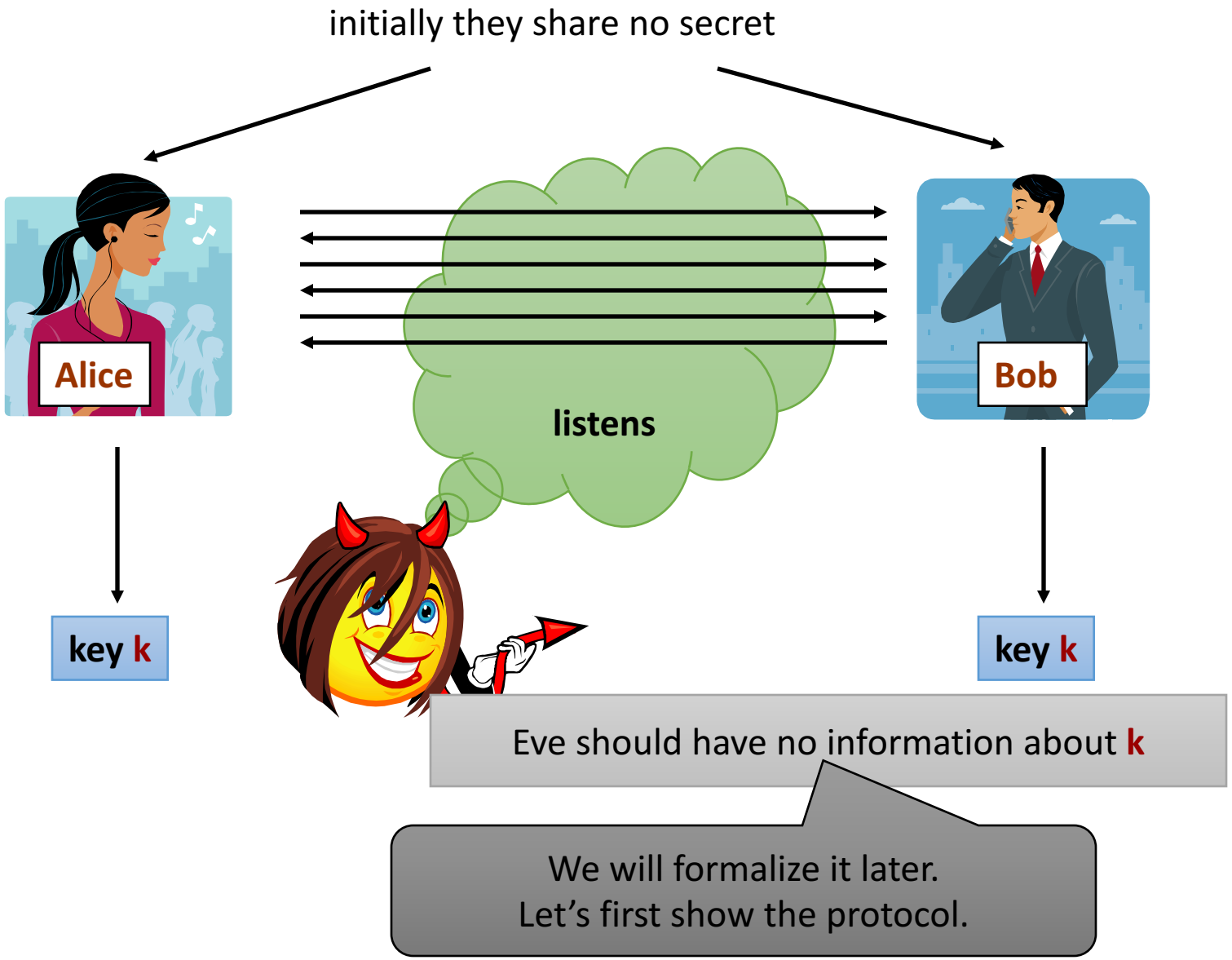
RSA was a trapdoor permutation, so the construction was quite easy...

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

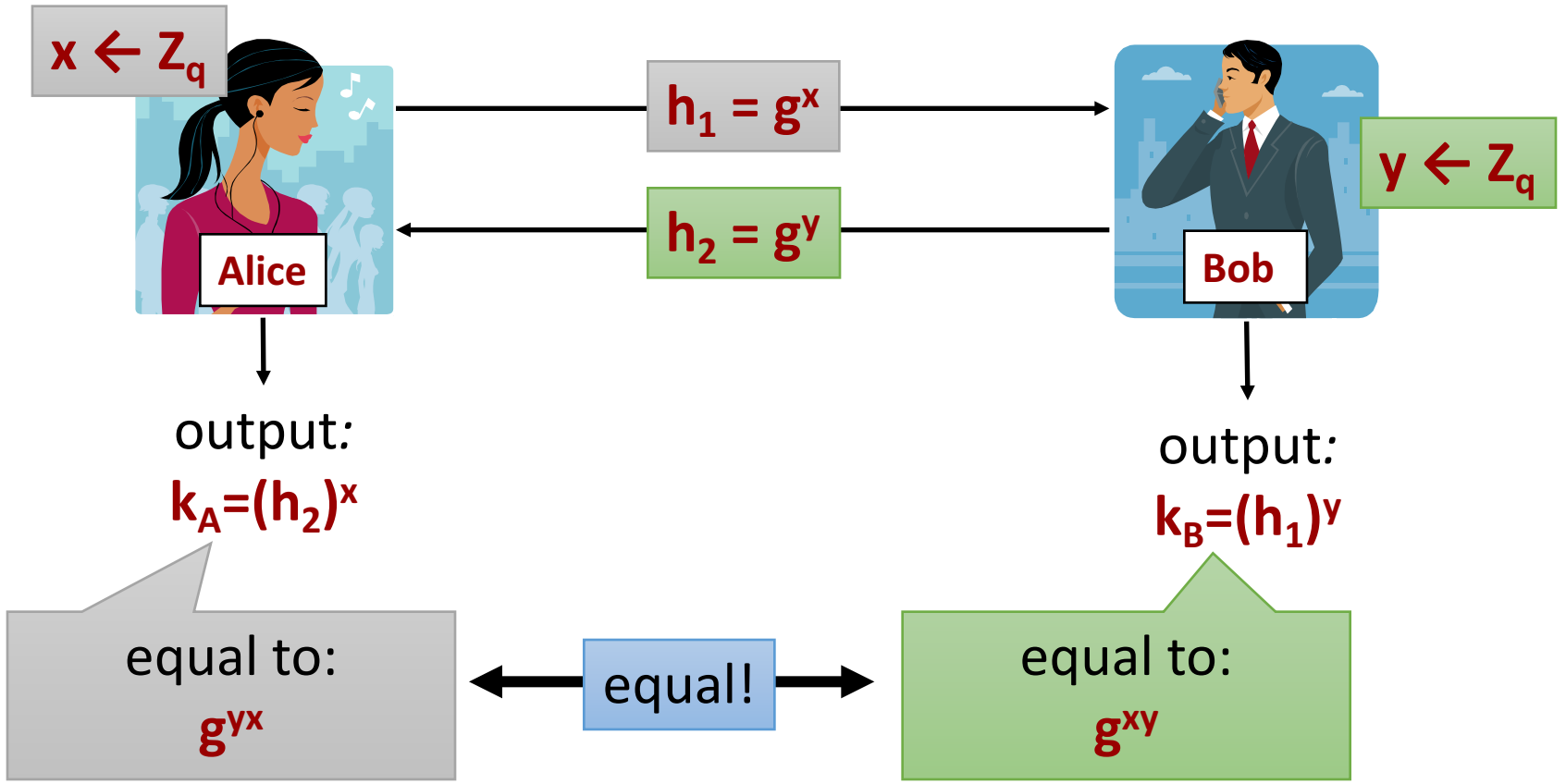


The Diffie-Hellman Key exchange

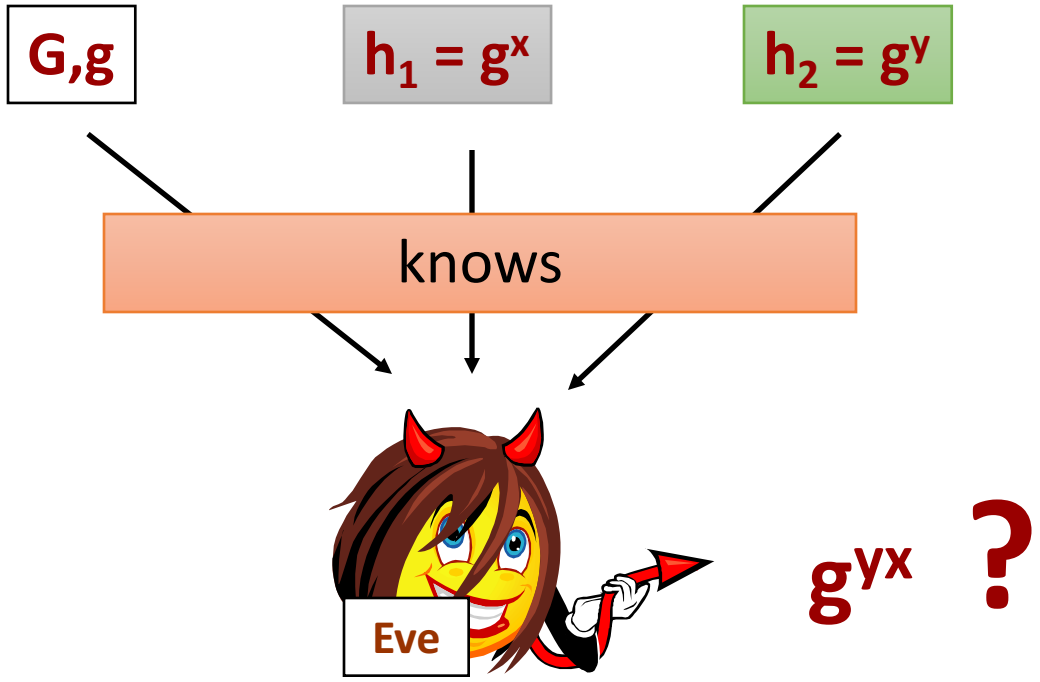
G – a group, where **discrete log is believed to be hard**

q = |G|

g – a generator of **G**



Security of the Diffie-Hellman exchange



Eve should have no information about g^{yx}

Is it secure?

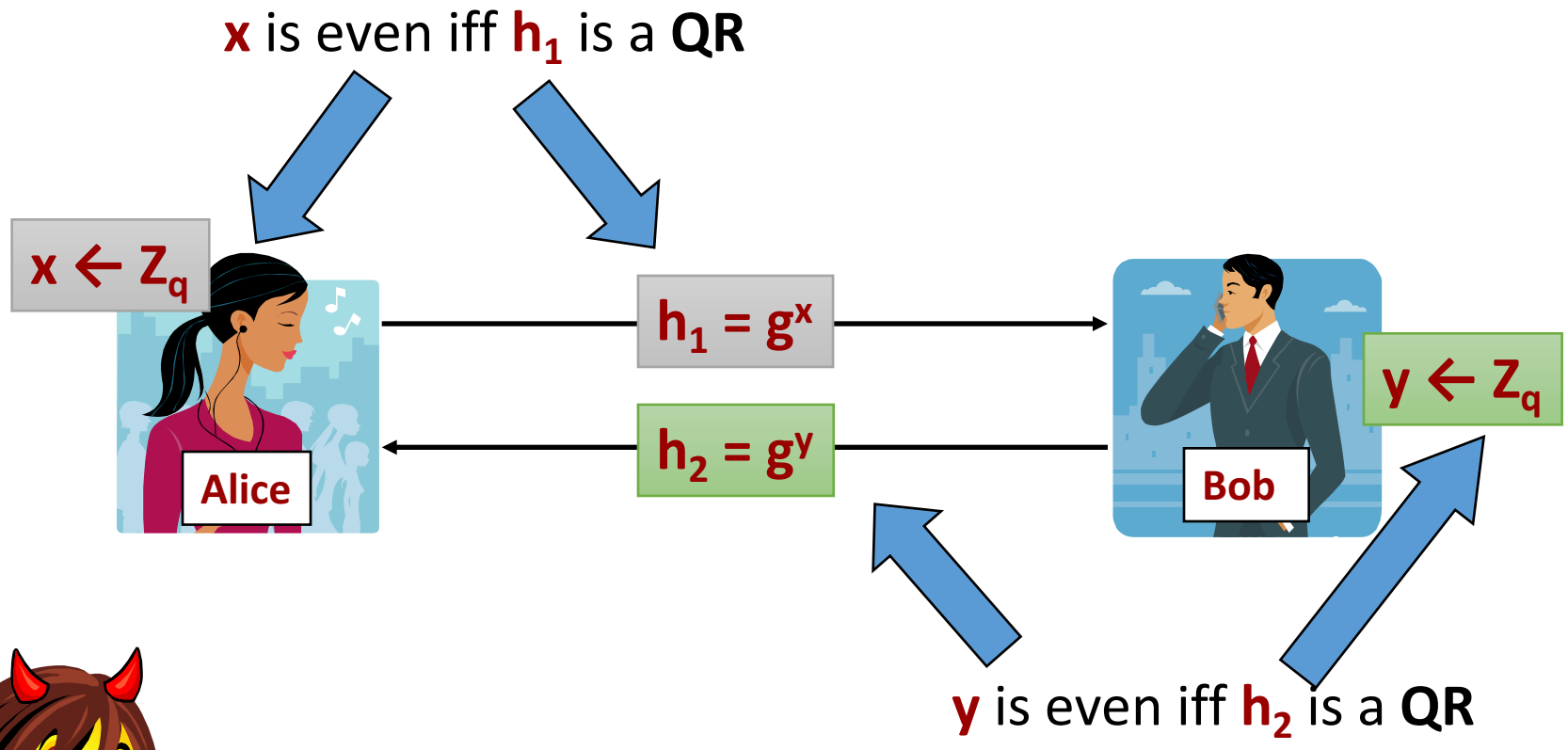
If the **discrete log in G** is easy then the **DH key exchange** is **not secure**.

(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!

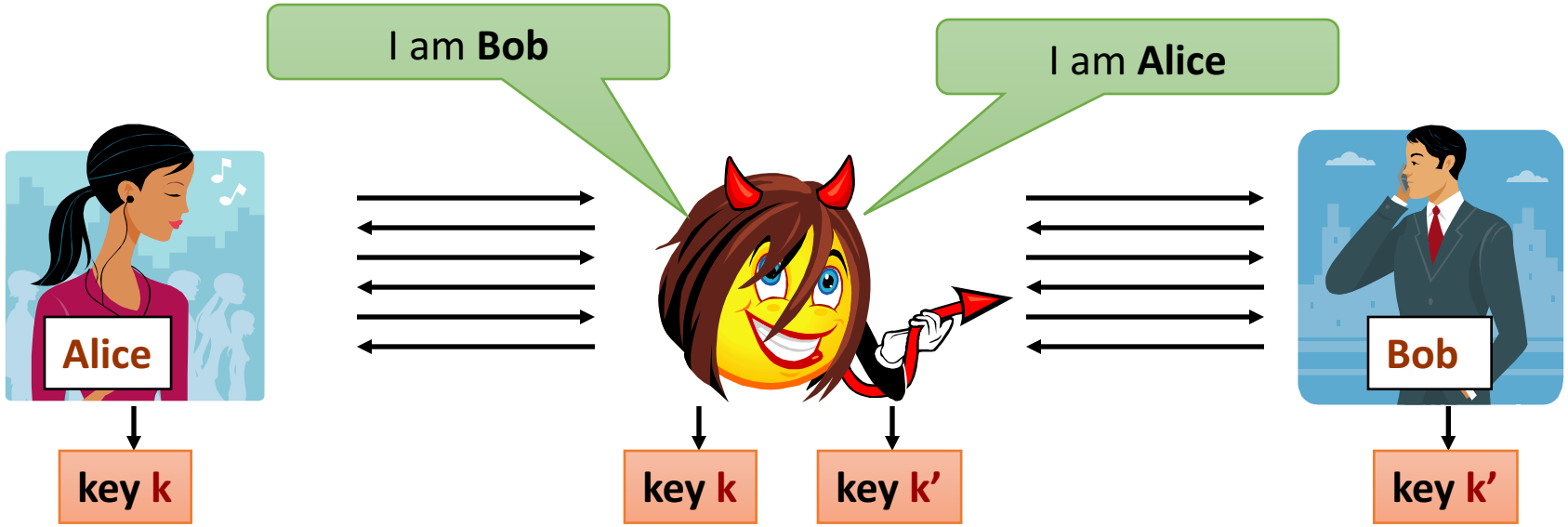
A problem

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.: **K = M = C = G**.

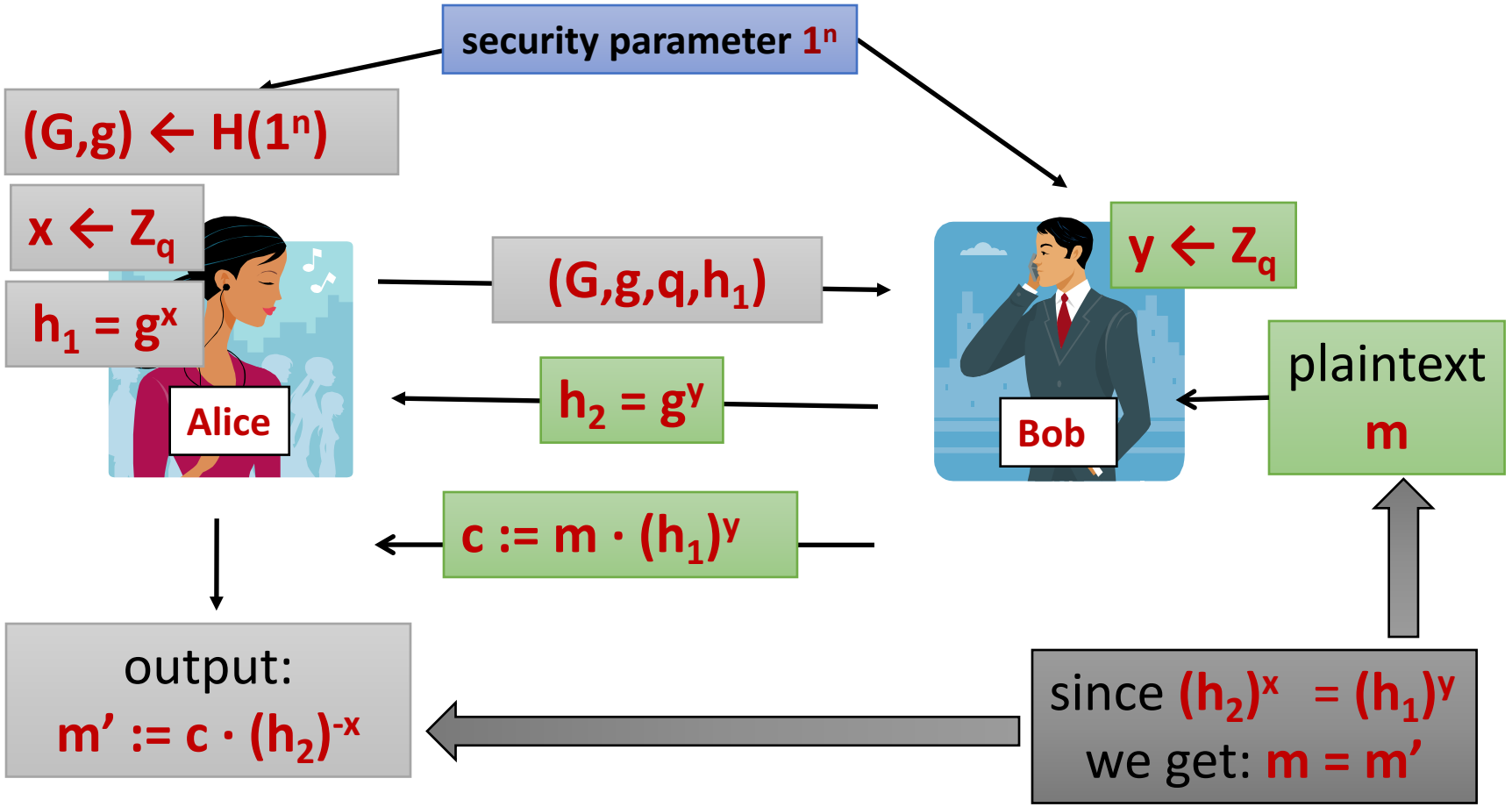
- **Enc(k,m) = m · k**
- **Dec(k,m) = m · k^{-1}**

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

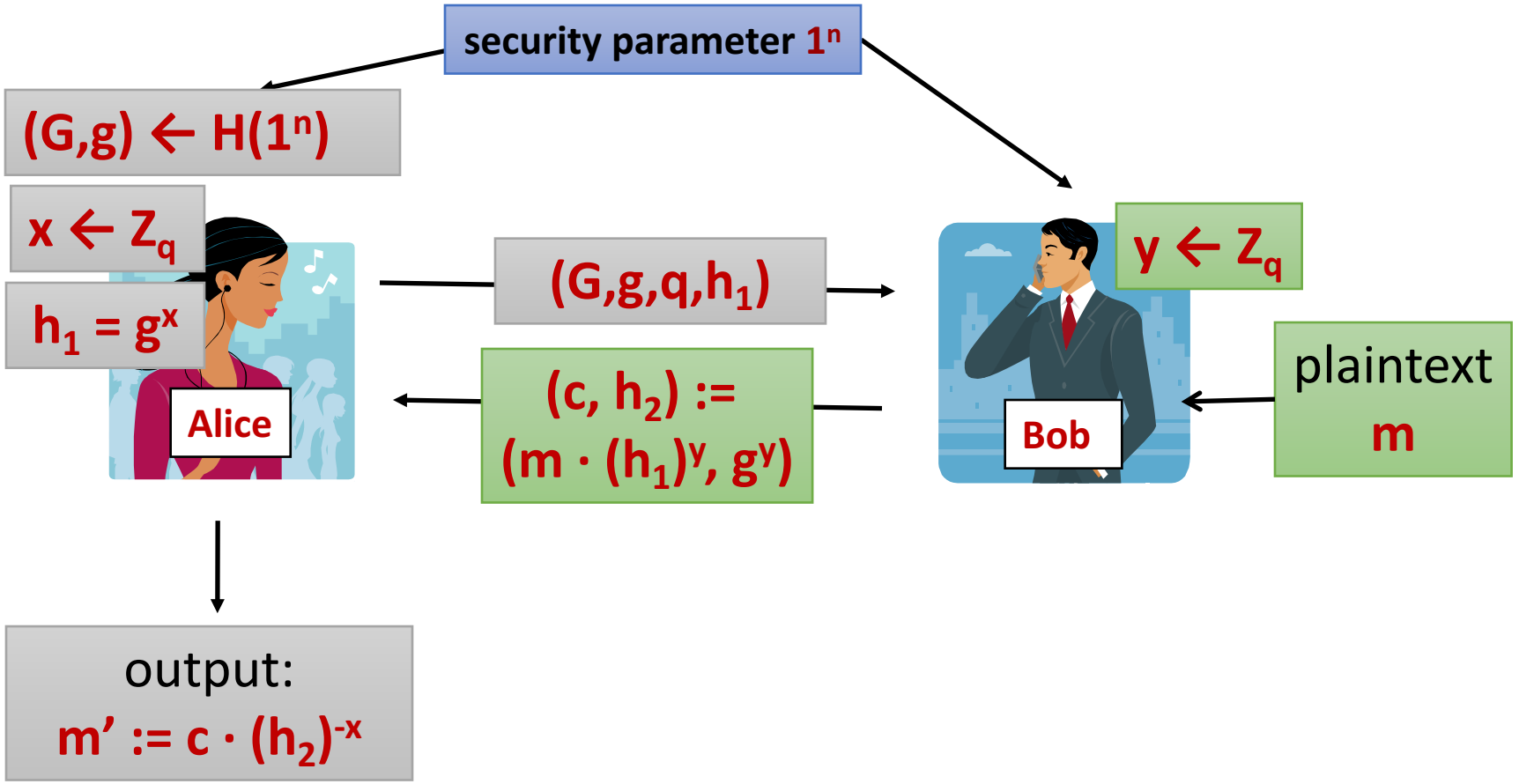
Alice can send a message **m ϵ G** to Bob "encrypting it with **k**" by setting:

c := m · k

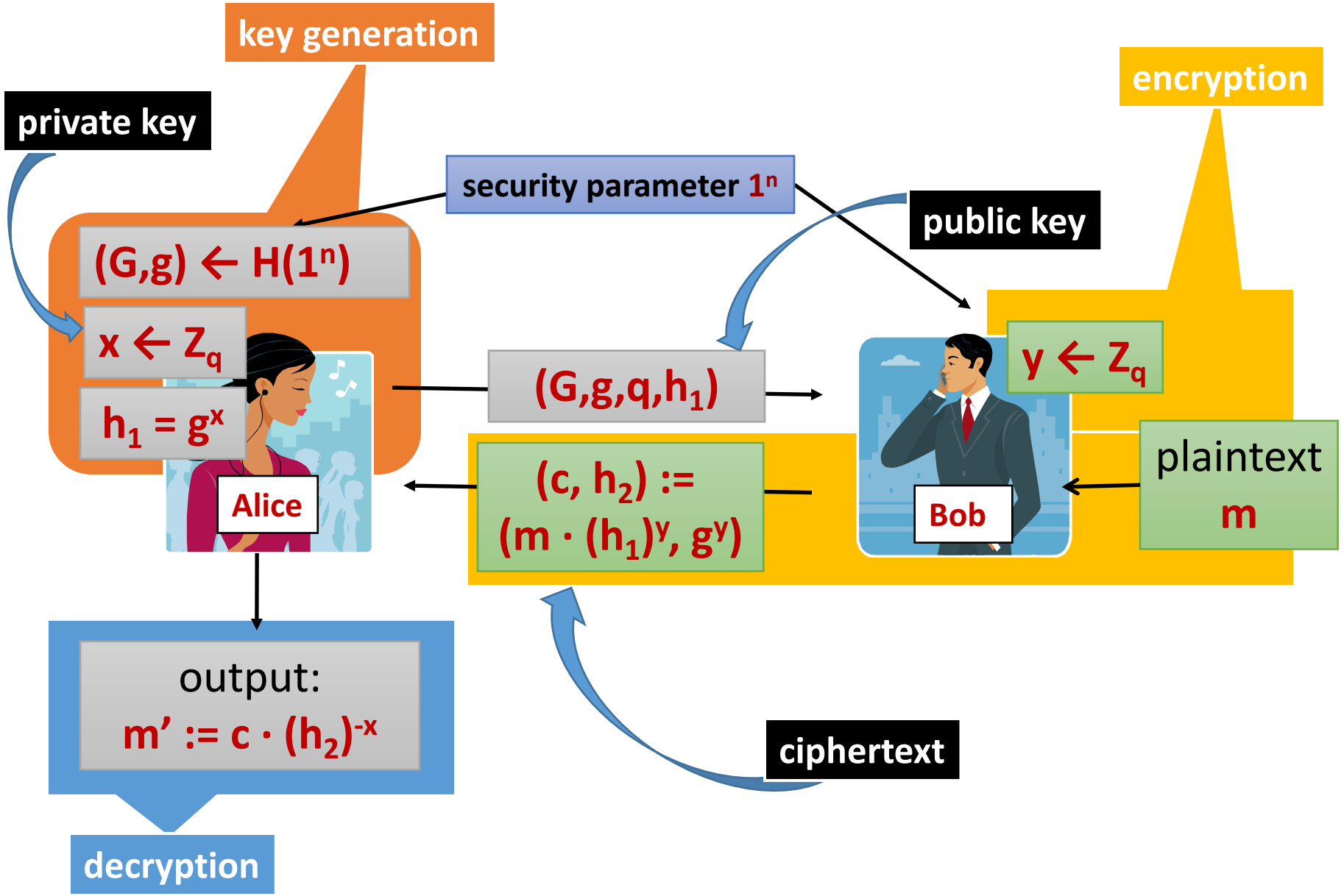
How does it look now?



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.