Cryptography Engineering

- Lecture 1 (Oct 23, 2024)

- Today:
  - Admin. Overview of the module
  - Symmetric Primitives
  - Diffie-Hellman Key Exchange, ElGamal Encryption
  - Signature

- Slides and example code can be found on:
  - https://runzhizeng.github.io/CE-w2425

Contact Information

- Lecturer & TA: Runzhi Zeng
- Email:
  - runzhi.zeng@uni-kassel.de
- Office hours
  - Office: Room 2628
  - 3:15 pm – 3:45 pm, Wednesday
  - (Please send an email in advance)
- All information is available on:
  - https://runzhizeng.github.io/CE-w2425

Time

- WS 2024/25: 14.10.2024 – 14.02.2025
- 14 Weeks, about 13 (or 14) lectures

- Lecture dates:
  - October: 23, 30
  - November: 06, 13, 20, 27
  - December: 04, 11, 18 (no lecture. Depending on the schedule, we can have a meeting on Feb 14, 2025)
  - January 2025: 15, 22, 29
  - February 2025: 5, 12, 14

Format

- Lecturing (about an hour)
- Coding (including discussion, Q&A, etc.)
- Please bring your laptop!

Programming Language

- You: Choose your favorite as long as it solves the task
- Ours: Always in Python

Homework

- Homework is mandatory for the exam:
  - Must complete 60% of the homework to join the exam
- Homework counts 40% of the final grade
- Homework is related to your final project (will be explained)!
- Three submission deadlines for homework:
  - Deadline-1: **22.11.2024 at 23:59**, homework for **lectures 1-2**
  - Deadline-2: **20.12.2024 at 23:59**, homework for **lectures 3-4, 7**
  - Deadline-3: **07.02.2025 at 23:59**, homework for **lectures 9-11**
- How to submit:
  - GitHub: Upload your codes and send Dr. Zeng the link via email before the deadlines.

Final Project

- Two options
- What to submit: Codes and a simple report
- The simple report should contain:
  - Choose 3-6 functions that you think are the best in your program and present them in your report, including *What it does*, *How it works*, and *Why it works correctly*
  - 2-4 pages, no introduction is needed
- Submission deadline for the final project:
  - **28.02.2025 at 23:59**
- How to submit:
  - Send Dr. Zeng an E-Mail before the deadlines.

Oral Exam

- Oral exam (About your final project):
  - We will ask you questions about your report and codes of your final project

- When? To be decided

Short summary about homework, final project and exam

- To be qualified for the exam: Finish 60% of the homework
- 40% of Final grade = Your homework
- 60% of Final grade = Your project (meaning codes and report) + oral exam

Overall Goals

- We focus on how to use cryptographic algorithms to ensure:
  - Confidentiality ("…learn nothing about your ciphertext…")
  - Integrity ("…cannot modify your data…")
  - Authentication ("…verify your identities…")
  - Forward/Backward Secrecy ("…protect past/future communications…")
  - Quantum Security ("…against attackers with quantum devices…")

…in real-world applications.

Brief Overview

- Main topics:
  - Symmetric primitives and necessary background **(today)**
  - Key exchange
  - Digital Signature
  - Secure Messaging
  - Password-based Authentication
  - Post-quantum Cryptography

Cryptography primitives – Hash, KDF

- Hash function:

  H(“…arbitrary-length string…”) = a fixed length bit string

  - Security: collision resistance, (second) preimage resistance, …
  - SHA3 (Secure Hash Algorithm 3)
  - **Do not** use MD5 (which was broken)…

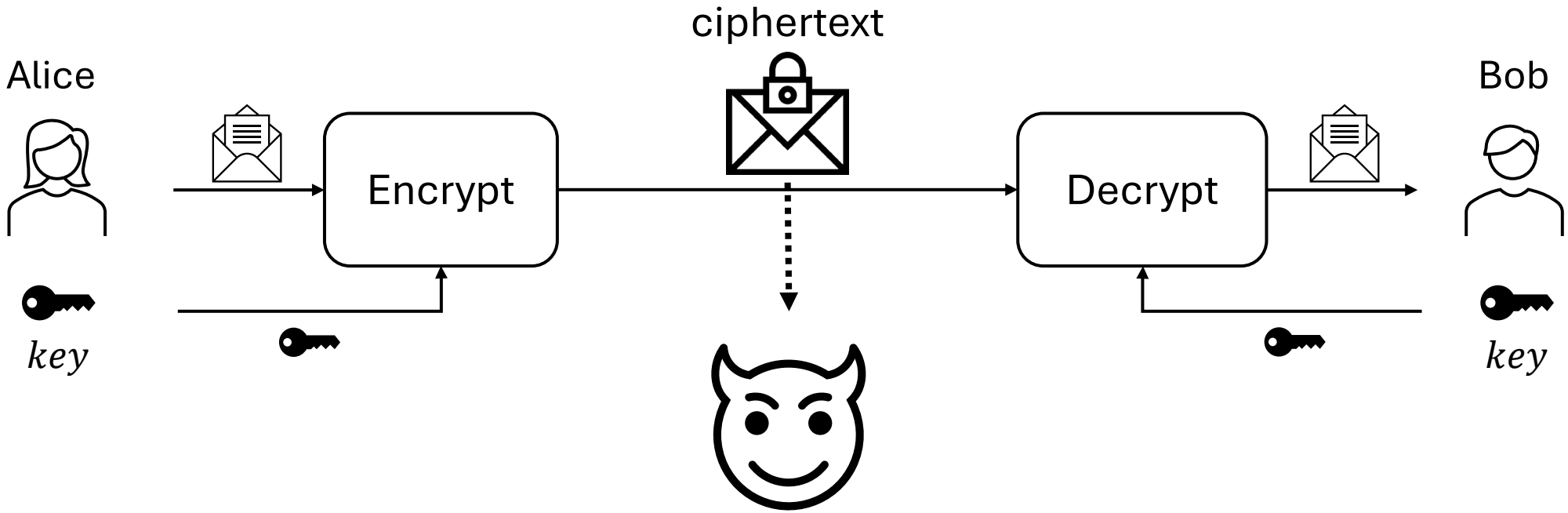
- Key Derivation Function (KDF)

  KDF(“…shared secret with randomness…”) = a symmetric key

  - Used to derive a key for symmetric key encryption, e.g., K <-- KDF(g^xy)
  - HKDF (based on HMAC): https://en.wikipedia.org/wiki/HKDF

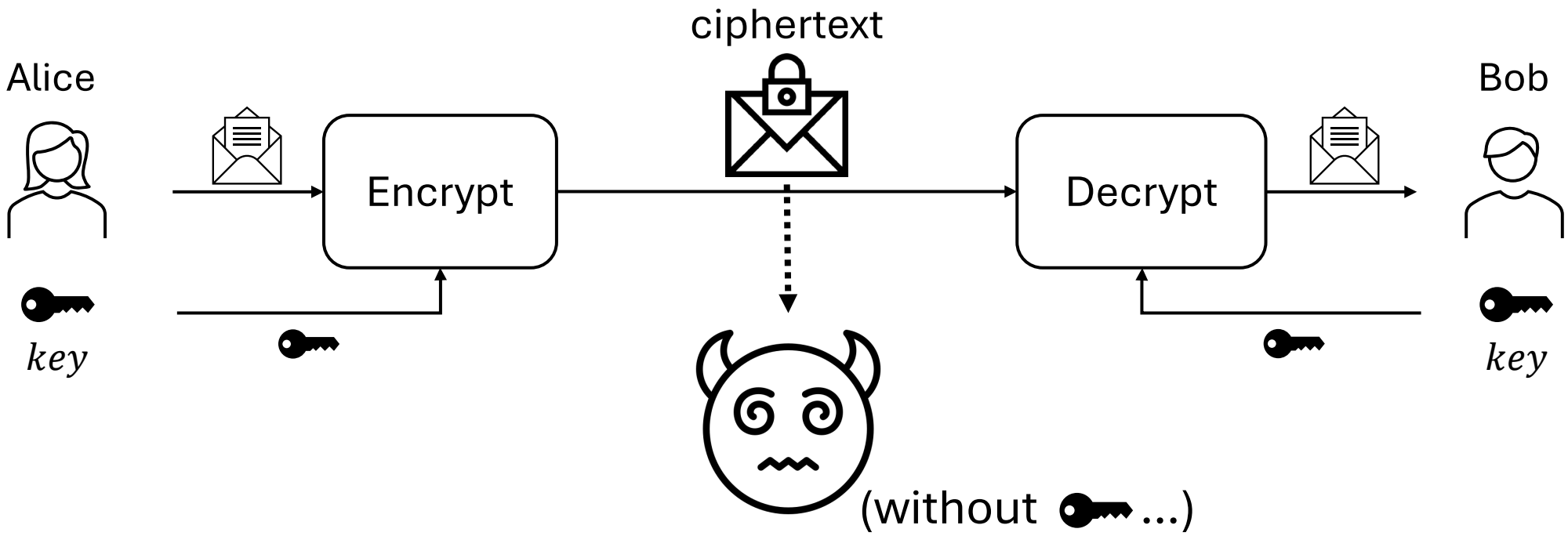
Cryptography primitives - SKE

- Symmetric-key Encryption



Cryptography primitives - SKE

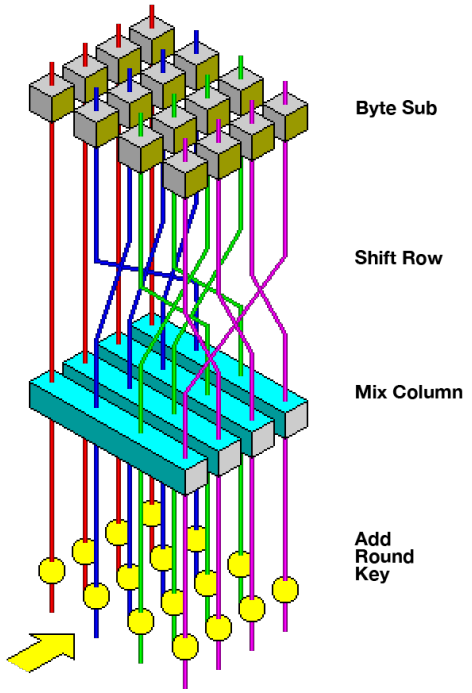
- Symmetric-key Encryption (**Confidentiality**)



Cryptography primitives – SKE

- Symmetric-key Encryption
  - AES (Advanced Encryption Standard)
  - Fixed-length encryption (block cipher)

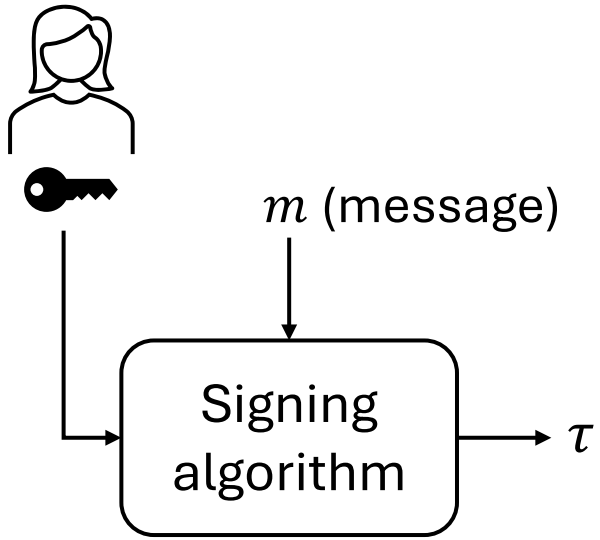
- Extend to arbitrary-length encryption **via Mode of Operation**
  - CBC, CTR, …

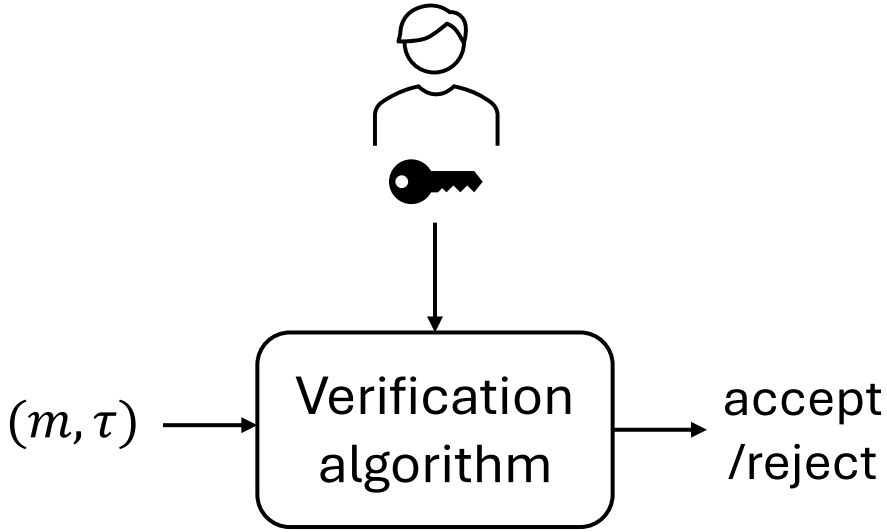


(Image from Wikipedia)

Cryptography primitives - MAC

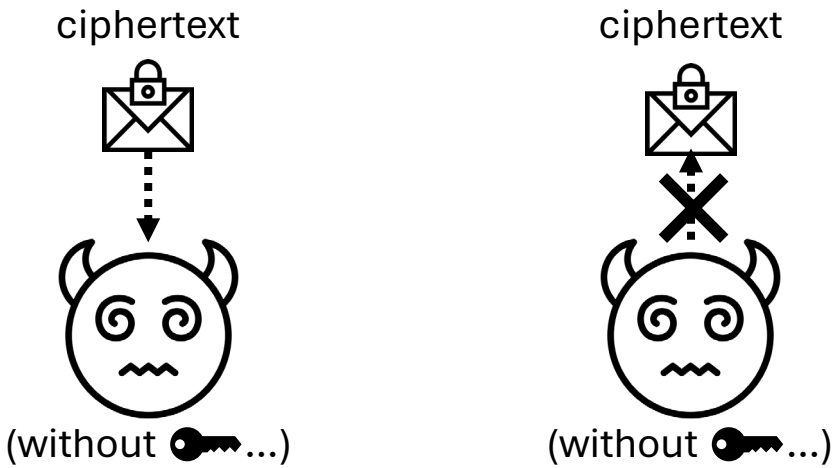
- Message Authentication Code (MAC)
  - **Integrity** (…cannot forge a valid MAC tag without knowing the secret key…)





Cryptography primitives - AE

- Authenticated Encryption
  - Symmetric Encryption
  - Not only **Confidentiality**, but also **Integrity**



Cryptography primitives - AE

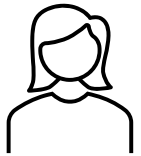
- Authenticated Encryption
  - Symmetric Encryption
  - Not only **Confidentiality**, but also **Integrity**

- Approaches to authenticated encryption:
  - Encrypt-then-MAC (EtM), …

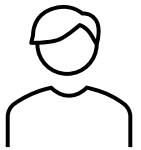
- Authenticated Encryption **with Associated Data (AEAD)**
  - Ensure the message **and additional data (like headers)** are authenticated and encrypted securely.
  - AES-GCM mode (https://en.wikipedia.org/wiki/Galois/Counter_Mode), ChaCha20-Poly1305, …

Cryptography primitives - DHKE

- Diffie-Hellman Key Exchange

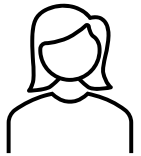


$(\mathbb{G}, g, q)$:
A q-order group $\mathbb{G}$ with a generator g

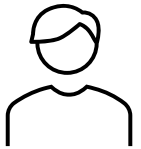


Cryptography primitives - DHKE

- Diffie-Hellman Key Exchange



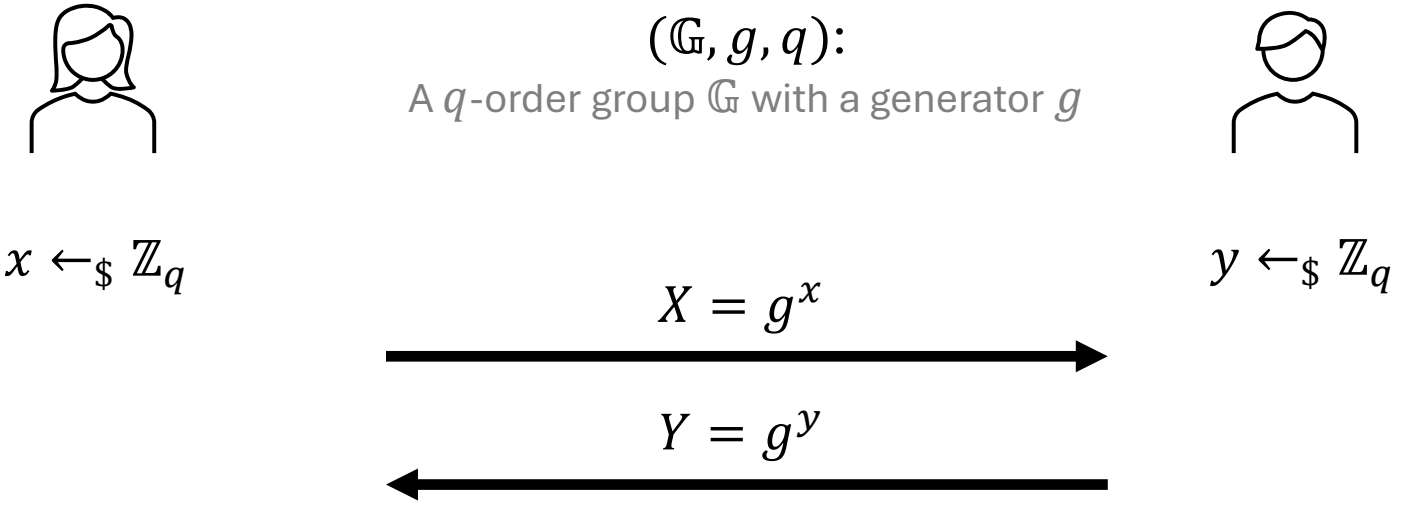
$(\mathbb{G}, g, q)$:
A q-order group $\mathbb{G}$ with a generator g



It can be an
elliptic curve group
or
a **group of integers**
modulo n.

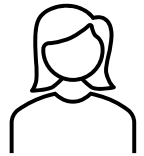
Cryptography primitives - DHKE

- Diffie-Hellman Key Exchange



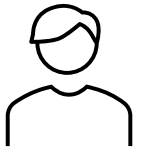
Cryptography primitives - DHKE

- Diffie-Hellman Key Exchange



$(\mathbb{G}, g, q)$:

A q-order group $\mathbb{G}$ with a generator g



$x \leftarrow_{\$} \mathbb{Z}_q$

$y \leftarrow_{\$} \mathbb{Z}_q$

$X = g^x$

$Y = g^y$

$K_{\text{Alice}} = Y^x$

$K_{\text{Alice}} = g^{xy} = K_{\text{Bob}}$

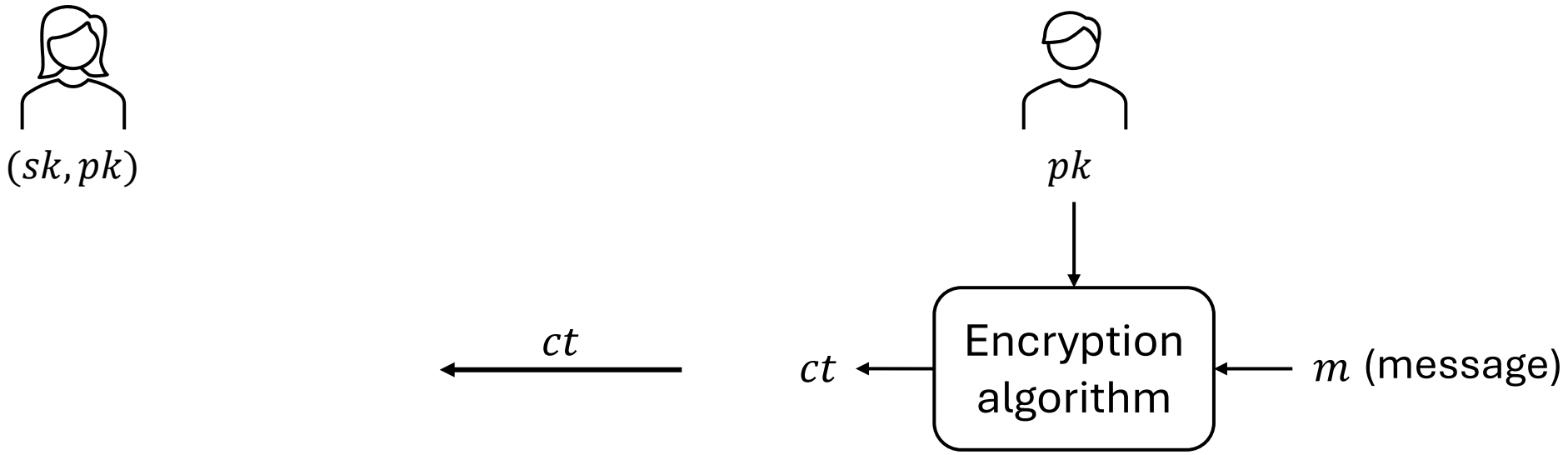
$K_{\text{Bob}} = X^y$

Cryptography primitives - PKE

- Public-key Encryption (PKE)

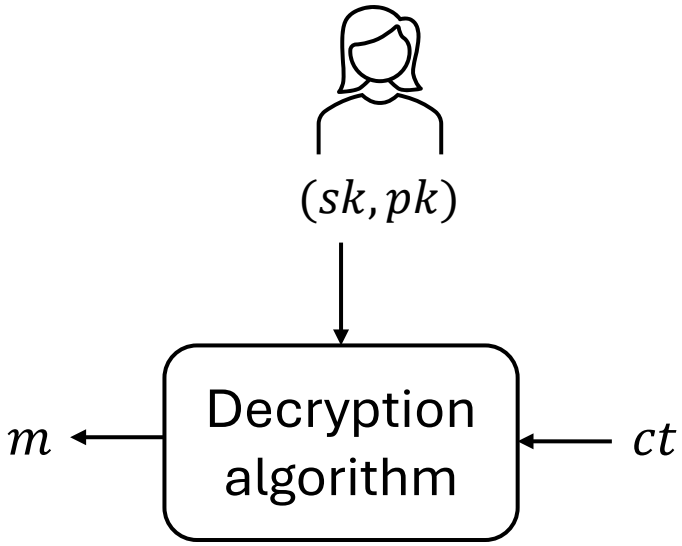
Cryptography primitives - PKE

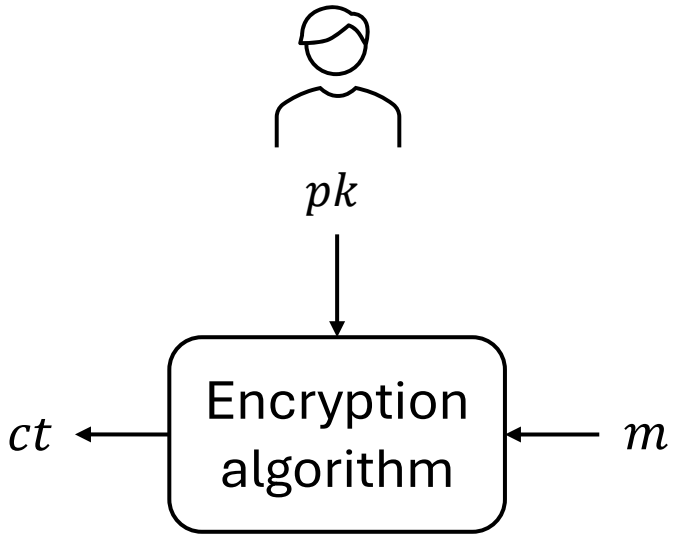
- Public-key Encryption (PKE)



Cryptography primitives - PKE

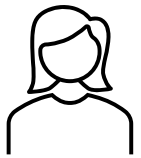
- Public-key Encryption (PKE)





Cryptography primitives - PKE

- A PKE scheme: ElGamal Encryption



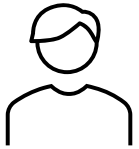
sk: $x \leftarrow_{\$} \mathbb{Z}_q$

pk: $X = g^x$

$(\mathbb{G}, g, q)$:

A q-order group $\mathbb{G}$ with a generator g

pk

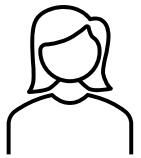


pk: X

message: $m \in \mathbb{G}$

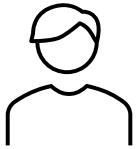
Cryptography primitives - PKE

- A PKE scheme: ElGamal Encryption



sk: $x \leftarrow_{\$} \mathbb{Z}_q$
pk: $X = g^x$

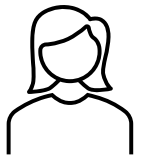
$(\mathbb{G}, g, q)$:
A q-order group $\mathbb{G}$ with a generator g



pk: X
message: $m \in \mathbb{G}$
$y \leftarrow_{\$} \mathbb{Z}_q$
$c_0 = Y$
$K = X^y (= g^{xy})$

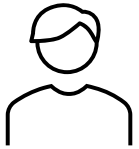
Cryptography primitives - PKE

- A PKE scheme: ElGamal Encryption



sk: $x \leftarrow_{\$} \mathbb{Z}_q$
pk: $X = g^x$

$(\mathbb{G}, g, q)$:
A q-order group $\mathbb{G}$ with a generator g



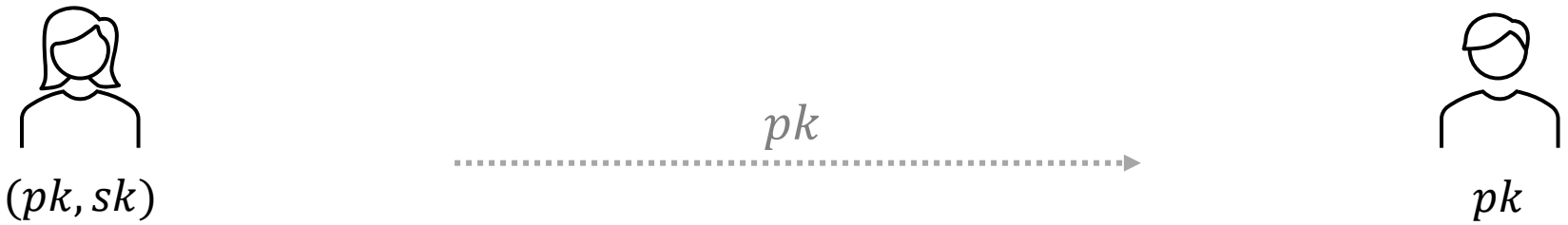
pk: X
message: $m \in \mathbb{G}$
$y \leftarrow_{\$} \mathbb{Z}_q$
$c_0 = Y$
$K = X^y (= g^{xy})$
$c_1 = K \cdot m$

c_0, c_1

$K = c_0^x (= g^{xy})$
$m = c_1 \cdot K^{-1}$

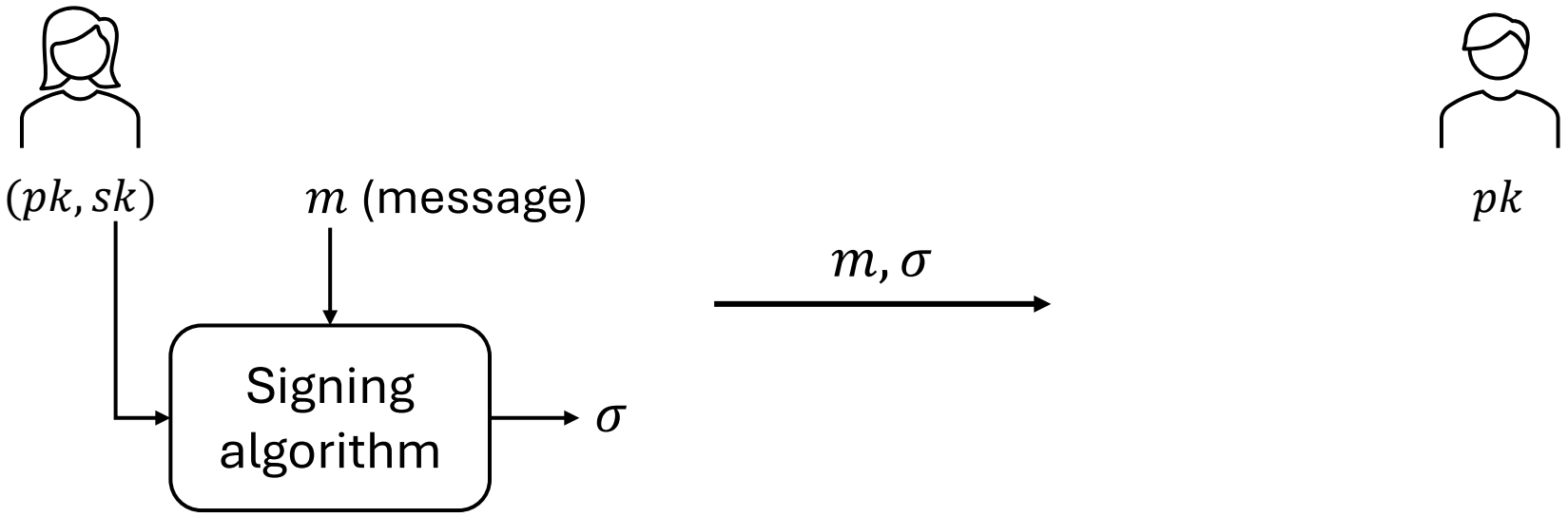
Cryptography primitives – Digital Signature

- Signature scheme



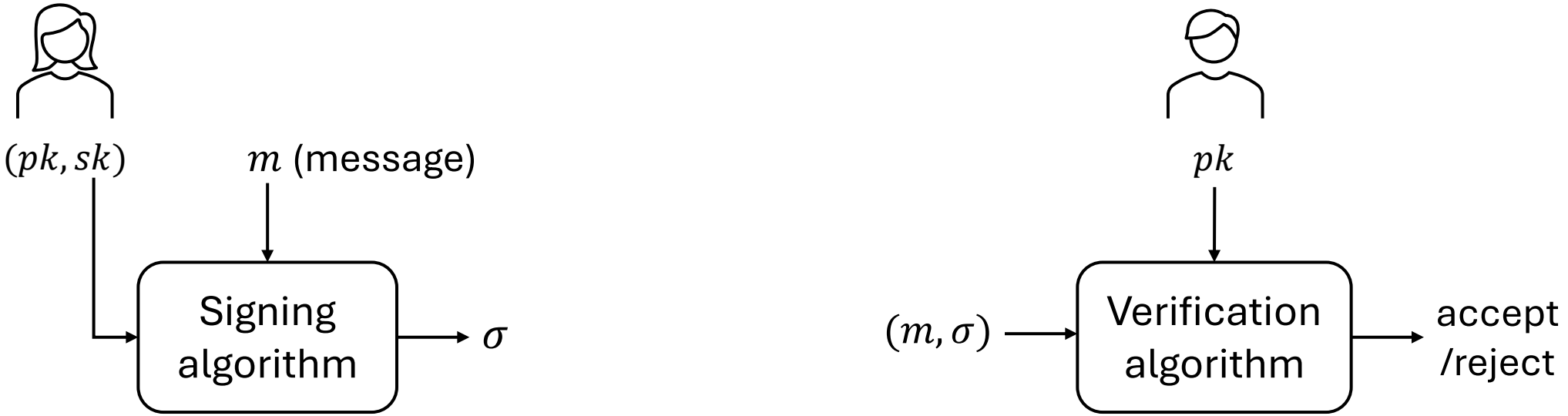
Cryptography primitives – Digital Signature

- Signature scheme



Cryptography primitives – Digital Signature

- Signature scheme



Cryptography primitives – Digital Signature

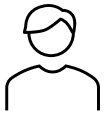
- Signature scheme
  - Schnorr’s signature scheme
  - DSA (Digital signature algorithm)
  - Will be discussed in the next lecture

Coding Tasks

1. Find some useful cryptographic libraries (Python: PyNaCl, ecdsa, cryptography, PyCryptodomem, etc.), **Google (Bing/ChatGPT/…) them**!
2. Given the example code of DHKE, implement the **hashed ElGamal encryption**

sk: $x \leftarrow_{\$} \mathbb{Z}_q$
pk: $X = g^x$

$(\mathbb{G}, g, q)$:
A q-order group $\mathbb{G}$ with a generator g



pk: X
message: $m \in \{0,1\}^*$

$y \leftarrow_{\$} \mathbb{Z}_q, Y = g^y$
$c_0 = Y$
$K = \mathrm{KDF}(Y, X^y)$
$c_1 = \mathrm{SEnc}(K, m)$

c_0, c_1 ←

$K = \mathrm{KDF}(c_0, c_0^x)$
$m = \mathrm{SDec}(K, c_1)$

- KDF: Key Derivation Function
- SKE = (Senc, SDec) is a symmetric-key encryption scheme

- Some example codes are available: DHKE+KDF+SKE, socket connection

Homework

- **Homework:** Consider implementing DHKE to enable two programs on your PC to perform a key exchange (using sockets, etc.)
    1. Program Alice <-- (connection) --> Program Bob
    2. Program Alice -- g^x --> Program Bob
    3. Program Alice <-- g^y -- Program Bob
- **Homework:** Add a trusted server to help the key exchange procedure (using sockets, etc.)
    1. Program Alice <-- (connection) --> Server <-- (connection) --> Program Bob
    2. Program Alice -- g^x --> Server -- g^x --> Program Bob
    3. Program Alice <-- g^y -- Server <-- g^y -- Program Bob