# Cryptography Engineering

- Lecture 1 (Oct 22, 2025)

- Today:
  - Admin. Overview of the module
  - Symmetric Primitives

# 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)
- Slides will be updated after lectures

# Time

- WS 2025/26: 13.10.2025 – 13.02.2026
- 14 Weeks, about 13 – 14 lectures

- Lecture dates:
  - October: 22, 29
  - November: 05, 12, 19, 26
  - December: 03, 10, 17
  - January 2026: 14, 21, 28
  - February 2026: 4, 11

# Format

- Lecturing (about an hour)
  - All lectures and slides are given in English
- Coding (two hours, including discussion, Q&A, etc.)
- **Coding-oriented -----  Please bring your laptop!**

# Programming Language

- Choose your favorite as long as it solves the task
- The example code: **Rust**

- Why Rust?
  - Efficient and memory-safe
  - Rich ecosystem and active community
- AI-assisted coding is recommended, but use them **wisely**!

# Homework

- Homework is mandatory for the exam:
  - **Must complete 60% of the homework to join the exam**
  - E.g., If we have 20 homework questions in total, then 0.6 * 20 = 12

- Homework counts 40% of the final grade:

$$\text{Final Grade of Homework} = 40 \times \left({}^{\text{points you obtain}}/_{\text{the number of questions}}\right)$$

// You need to get at least $40 \times 60\% =$ **24 points to qualify for the final exam**.

- **3 – 4 homework sets** (each due in **two weeks**, announced after relevant lectures)

# Final Project

- One or two options
- What to submit: **Code 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:
  - **28.02.2026 at 23:59**
- How to submit:
  - Send me an E-Mail (with all files of your code or your github link, etc) before the deadline

# Oral Exam

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

- When? To be decided

# Short Summary about Final Grade

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

# Brief Overview

- Main focuses:
  - Learning cryptography through practice
  - Securely implementing cryptographic algorithms
  - Applying cryptographic algorithms to real-world security problems

# Brief Overview

- Main topics:
  - Symmetric primitives **(today)**
  - Digital Signature and Certificate
  - Key exchange and TLS handshake
  - Secure Messaging
  - Password-based Authentication
  - Encrypted Cloud Storage
  - Secure Implementation

# Symmetric Primitives

- Hash function:

  H( “…arbitrary-length string…” ) = a fixed length (e.g., 256) bit string

  - Security: collision resistance, (second) preimage resistance, …
  - SHA (Secure Hash Algorithm) family: ~~SHA-1~~, SHA-2, SHA-3

# Symmetric Primitives

- Hash function:

  H( “…arbitrary-length string…” ) = a fixed length (e.g., 256) bit string

  - Security: collision resistance, (second) preimage resistance, …
  - SHA (Secure Hash Algorithm) family: ~~SHA-1~~, SHA-2, SHA-3

- Find collision: Find $x, x'$ s.t. $x \neq x'$ and $h(x) = h(x')$
- **Collision resistance**: Finding any collision is hard (i.e., no polynomial-time algorithm)

# Symmetric Primitives

- Hash function:

  H( “…arbitrary-length string…” ) = a fixed length (e.g., 256) bit string

  - Security: collision resistance, (second) preimage resistance, …
  - SHA (Secure Hash Algorithm) family: ~~SHA-1~~, SHA-2, SHA-3

- **Preimage resistance**: Given an arbitrary bit string $s$, it is hard to find $x$ s.t. $h(x) = s$

# Symmetric Primitives

- Hash function:

  H( “…arbitrary-length string…” ) = a fixed length (e.g., 256) bit string

  - Security: collision resistance, (second) preimage resistance, …
  - SHA (Secure Hash Algorithm) family: ~~SHA-1~~, SHA-2, SHA-3

- **Second Preimage resistance**: Given $x$, it is hard to find $x'$ s.t. $h(x) = h(x')$

# Symmetric Primitives

- Hash function:

  H( “…arbitrary-length string…” ) = a fixed length (e.g., 256) bit string

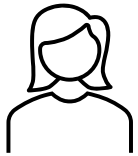
  - Security: collision resistance, (second) preimage resistance, …
  - SHA (Secure Hash Algorithm) family: ~~SHA-1~~, SHA-2, SHA-3

- Quick questions: Let $h$ be a hash function and *test.rs* be a file
  - Given $h$(*test.rs*) (compute the hash of the whole file), can we recover *test.rs*?
  - Can we find another file such that its hash is $h$(*test.rs*)?
  - Can we find two arbitrary different files such that their hashes are the same?

# Cryptography primitives - SKE

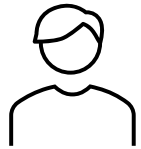
- Symmetric-key Encryption

Alice



$key$

Bob



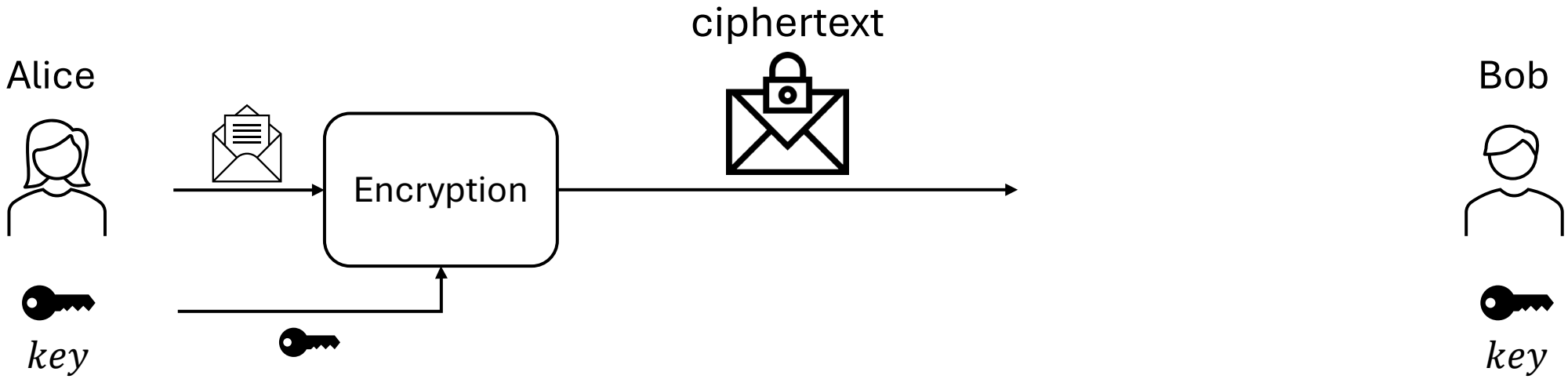
$key$

# Cryptography primitives - SKE

- Symmetric-key Encryption

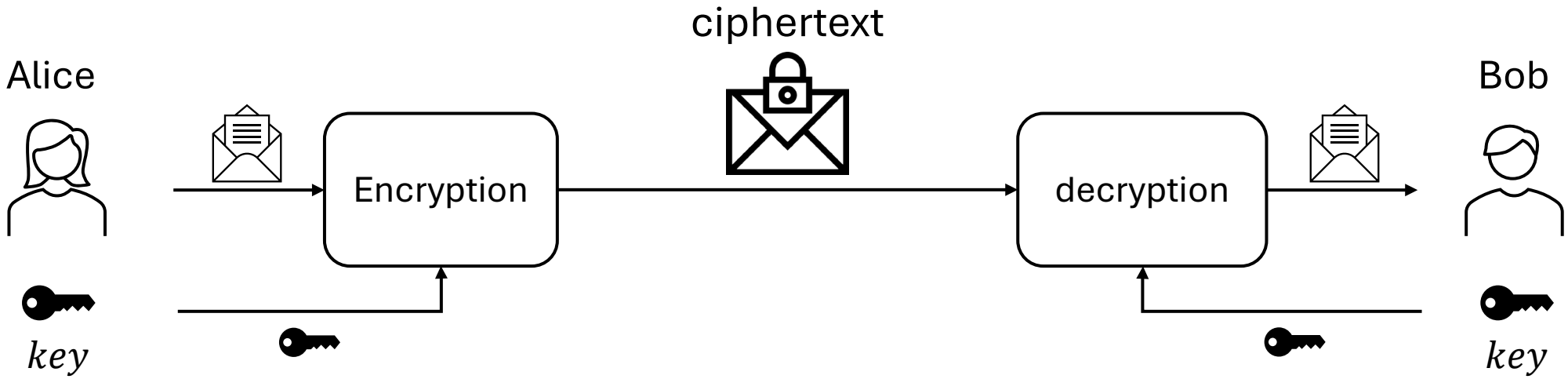
# Cryptography primitives - SKE

- Symmetric-key Encryption



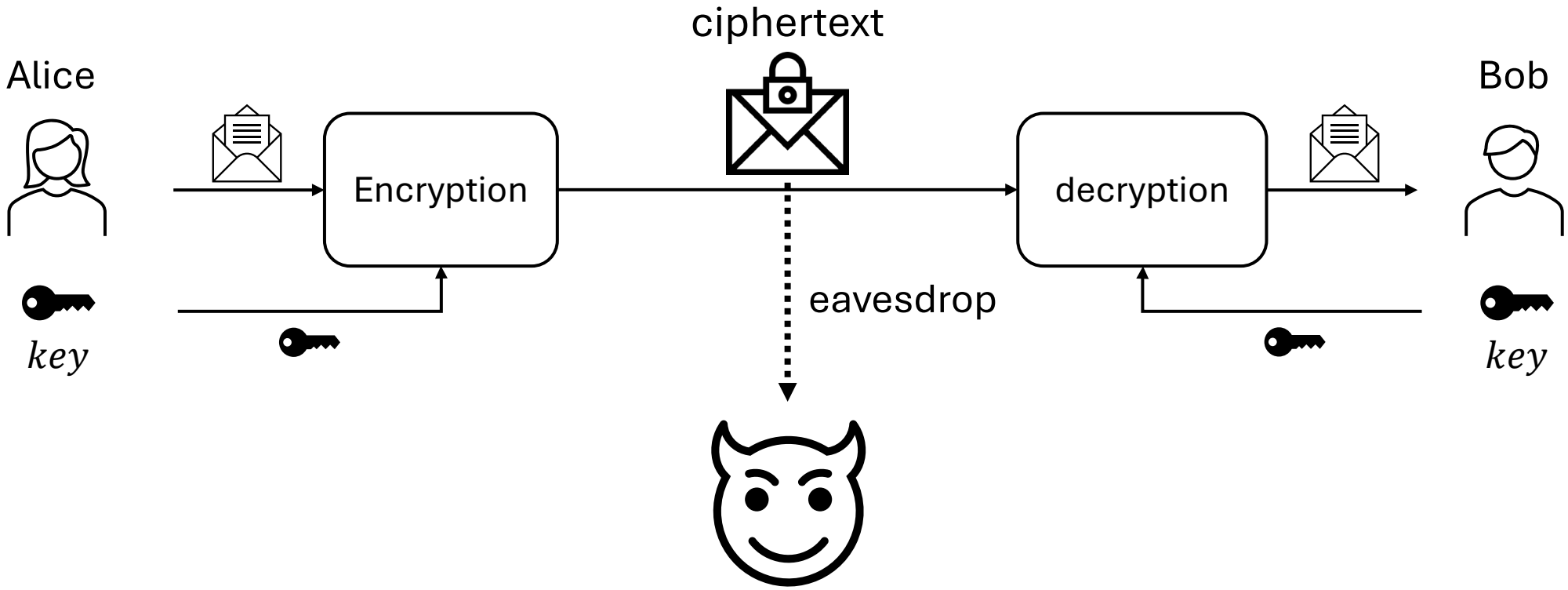
# Cryptography primitives - SKE

- Symmetric-key Encryption



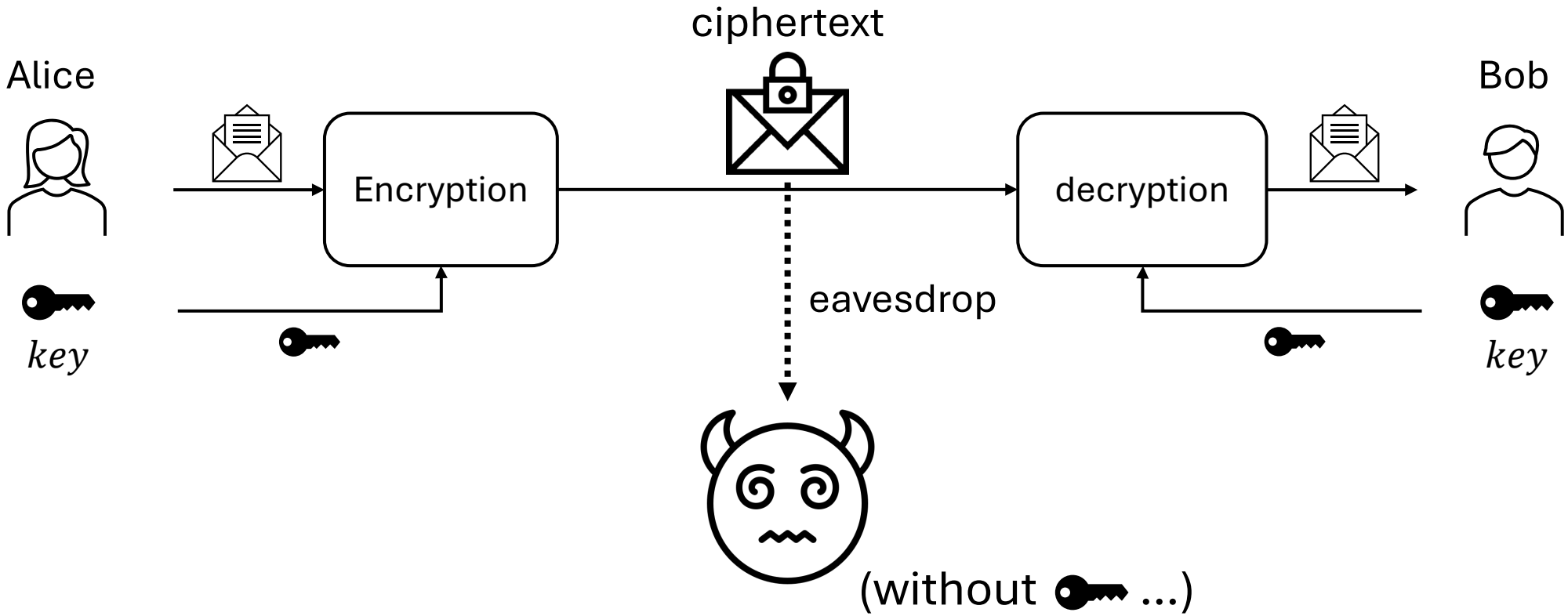
# Cryptography primitives - SKE

- Symmetric-key Encryption



# Cryptography primitives - SKE

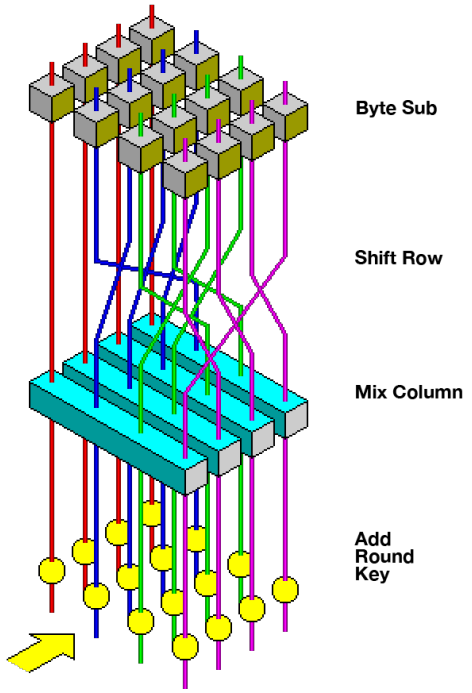
- Symmetric-key Encryption (**Confidentiality**)



# Cryptography primitives - SKE

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

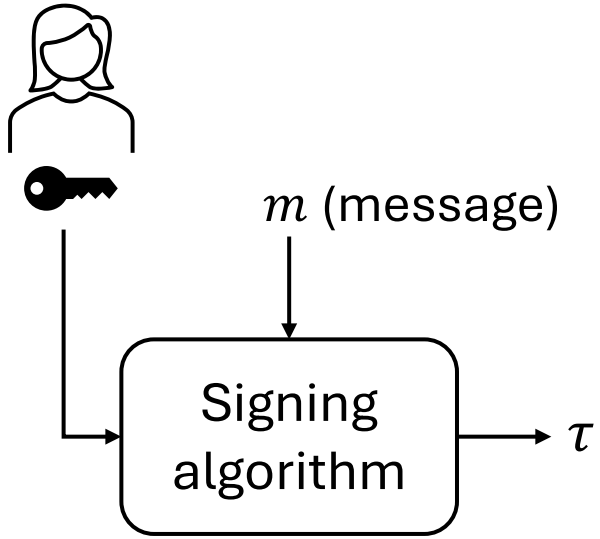
- Extend to arbitrary-length encryption **via Mode of Operation**
  - CBC, CTR, …

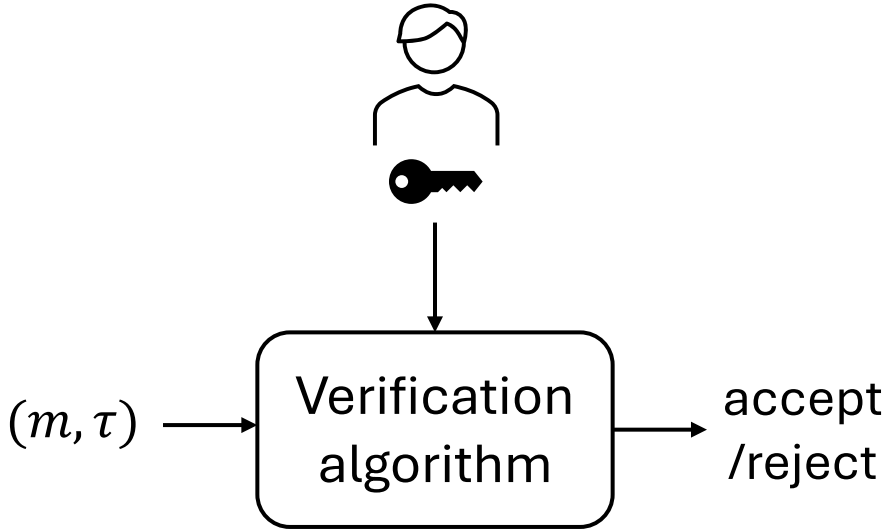



(Image from Wikipedia)

# Cryptography primitives - MAC

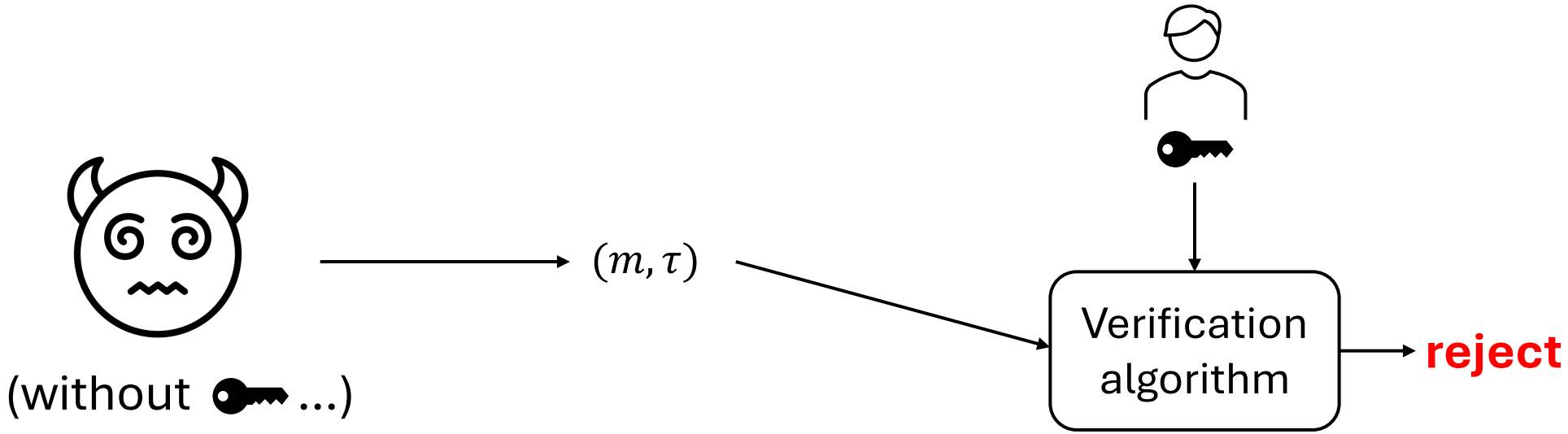
- Message Authentication Code (MAC)
  - **Integrity** (…cannot forge a valid MAC tag without knowing the secret key…)
  - **HMAC** (Hash-based message authentication code)



# Cryptography primitives - MAC

- Message Authentication Code (MAC)
  - **Integrity** (…cannot forge a valid MAC tag without knowing the secret key…)
  - **HMAC** (Hash-based message authentication code)



# Cryptography primitives – KDF

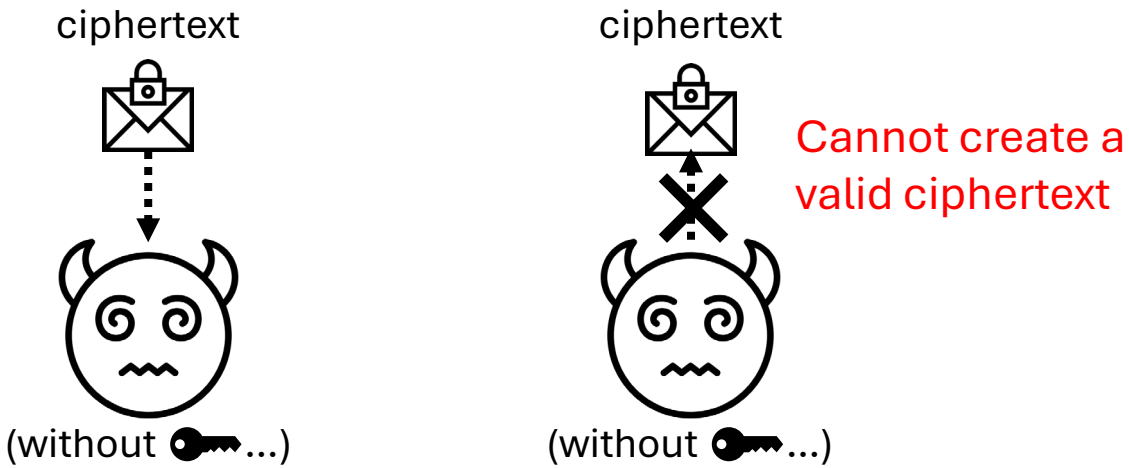
- Key Derivation Function (KDF)

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

  - Used to derive a key for symmetric key encryption
  - HKDF (based on HMAC)
  - Derive keys of arbitrary lengths

# 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, ChaCha20-Poly1305, …

# Exercise

- Practice your Rust (or your favourite language) skill via finishing the following tasks:

1. **Hash functions:**
   (1) Input a file; (2) Compute its digital fingerprint (using SHA3-256); (3) Verify the digital fingerprint.
2. **AES-256:**
   (1) Generate a random key; (2) Encrypt an arbitrary string using AES; (3) Decrypt the ciphertext.
3. **HMAC:**
   (1) Generate a random key; (2) Generate a MAC code for an arbitrary string using HMQC; (3) Verify it.
   (4) Modify the MAC code and verify it again.
4. **AEAD:**
   (1) Generate a random key; (2) Generate an arbitrary string using AES-GCM; (3) Decrypt it;
   (4) Modify the ciphertext and decrypt it again; (5) Modify the AD (associate date) and decrypt again.
5. **KDF:** Using HKDF to derive a key (using arbitrary inputs).

# Further Reading

- AEAD and AES-GCM: https://en.wikipedia.org/wiki/Galois/Counter_Mode
- HKDF: https://en.wikipedia.org/wiki/HKDF
- Mode of operation: https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation
- Awesome Cryptography Rust: https://github.com/rust-cc/awesome-cryptography-rust