

Cryptography Engineering

- Lecture 1 (Oct 21, 2026)
- Today:
 - Course information
 - Symmetric Primitives

Contact Information

- Lecturer & TA: Runzhi Zeng
- Email:
 - runzhi.zeng@uni-kassel.de
- Office hours
 - Office: Room 2628, FB16
 - 3:30 pm – 4:00 pm, Wednesday
 - (Please send an email in advance)
- Slides will be updated after lectures

Format

- Lecturing (about an hour, 12:00 – 13:00)
 - All lectures and slides are given in English
- Coding (two hours, 13:00 – 15:00, including discussion, Q&A, etc.)
- **Coding-oriented ----- Please bring your laptop!**

Time

- Lecture dates:
 - October: 21, 28
 - November: 04, 11, 18, 25
 - December: 02, ~~09, 16~~
 - January 2027: 13, 20, 27
 - February 2027: 03, 10
- No lectures on December 09 or 16.
 - All three hours on December 2 will be devoted to lectures.

Programming Language

- **Rust**

- Efficient and memory-safe
- Rich ecosystem and active community



- In this module, we require all implementations to be written in Rust.

Assessment and Grading

- The final grade is based on two components:
 - **Written exam**
 - **Final project with a report**
- Written exam:
 - Students must score at least 60 points on the written exam to pass the course, **regardless of their project performance.**
- Final project:
 - For students who pass the written exam, the final grade is calculated as the weighted average of the written exam score and the final project/report score.

Written Exam

- Goal: Assess your **understanding of the cryptographic building blocks** covered in this course and your ability to apply them correctly in practice.

Written Exam

- Goal: Assess your **understanding of the cryptographic building blocks** covered in this course and your ability to apply them correctly in practice.
- In particular, you should understand:
 - What each cryptographic primitive is designed to achieve.
 - When and where a particular primitive should be used.
 - When using a particular primitive would be inappropriate or insecure.
- When: TBD

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**
- Submission deadline for the final project: TBD (after the written exam)
- How to submit:
 - Send me an E-Mail (with all files of your code or your github link, etc) before the deadline

Short Summary about Final Grade

$$\text{Final Score} = \begin{cases} \text{Fail.} & \text{if } W < 60 \\ \frac{1}{2}(W + P). & \text{if } W \geq 60 \end{cases}$$

W : Written exam score

P : Final project and report score

- The final score (out of 100) will be converted to the German grading scale (1.0–5.0).

AI Usage

- **AI as a Tool, Not a Replacement**
 - AI is encouraged
 - Let AI assist you, not replace you
- **Learn to solve real-world cryptographic problems with AI**
 - **Model** real-world security problems
 - **Identify** which cryptographic algorithms are needed and where they should be applied
 - **Guide** AI to implement your designs correctly and securely

Brief Overview

- Main focuses:
 - Learning cryptography through practice
 - Securely implementing cryptographic algorithms
 - Applying cryptographic algorithms to real-world problems
- This module assumes you have some basic background about cryptography
 - E.g., Formal Definitions of (a)symmetric encryption, digital signature,...
 - Provable security are **not** included in this module

Brief Overview

- Today's topics: Hash functions and symmetric primitives

Hash Function

- Hash function:

$H(\text{"...arbitrary-length string..."}) = \text{a fixed length (e.g., 256) bit string (digest)}$

Hash Function

- Hash function:

$H(\text{"...arbitrary-length string..."}) = \text{a fixed length (e.g., 256) bit string (digest)}$



Generated by ChatGPT-6 Sol

Hash Function

- Hash function:

$H(\text{"...arbitrary-length string..."}) = \text{a fixed length (e.g., 256) bit string}$

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

Security Properties of Hash Function

- Hash function:

$H(\text{"...arbitrary-length string..."}) = \text{a fixed length (e.g., 256) bit string}$

- 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)

Security Properties of Hash Function

- Hash function:

$H(\text{"...arbitrary-length string..."}) = \text{a fixed length (e.g., 256) bit string}$

- 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. $h(x) = s$

Security Properties of Hash Function

- Hash function:

$H(\text{"...arbitrary-length string..."}) = \text{a fixed length (e.g., 256) bit string}$

- 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 $x' \neq x$ s.t. $h(x) = h(x')$

Security Properties of Hash Function

- Hash function:

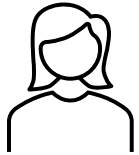
$H(\text{"...arbitrary-length string..."}) = \text{a fixed length (e.g., 256) bit string}$

- 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 $h(\text{test.rs})$ (compute the hash of the whole file), can we recover *test.rs*?
 - Can we find *another* file such that its hash is $h(\text{test.rs})$?
 - Can we find two arbitrary different files such that their hashes are the same?

Symmetric Primitives - SKE

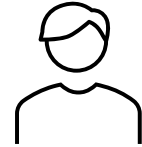
- Symmetric-key Encryption

Alice



key

Bob



key

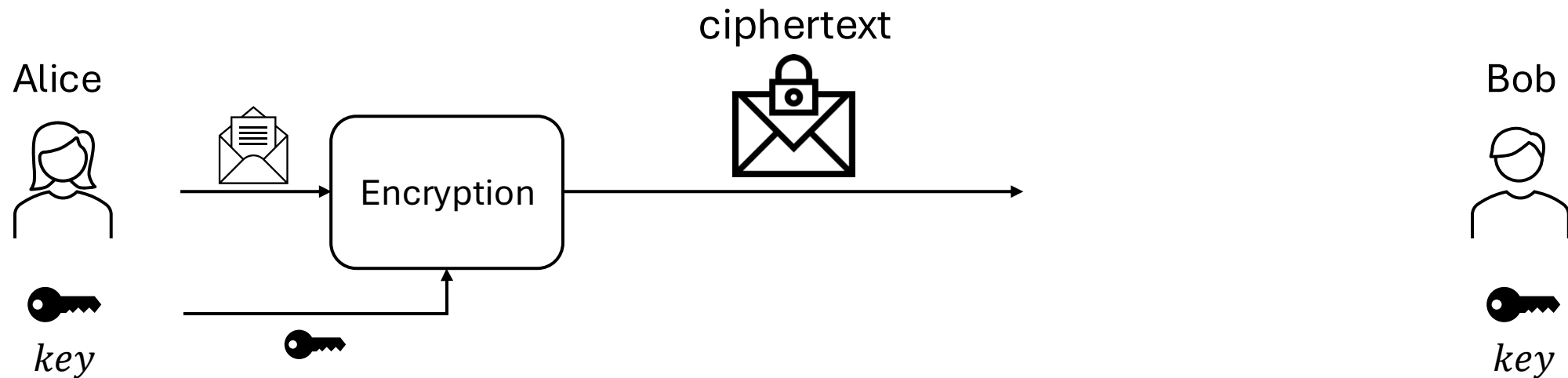
Symmetric Primitives - SKE

- Symmetric-key Encryption



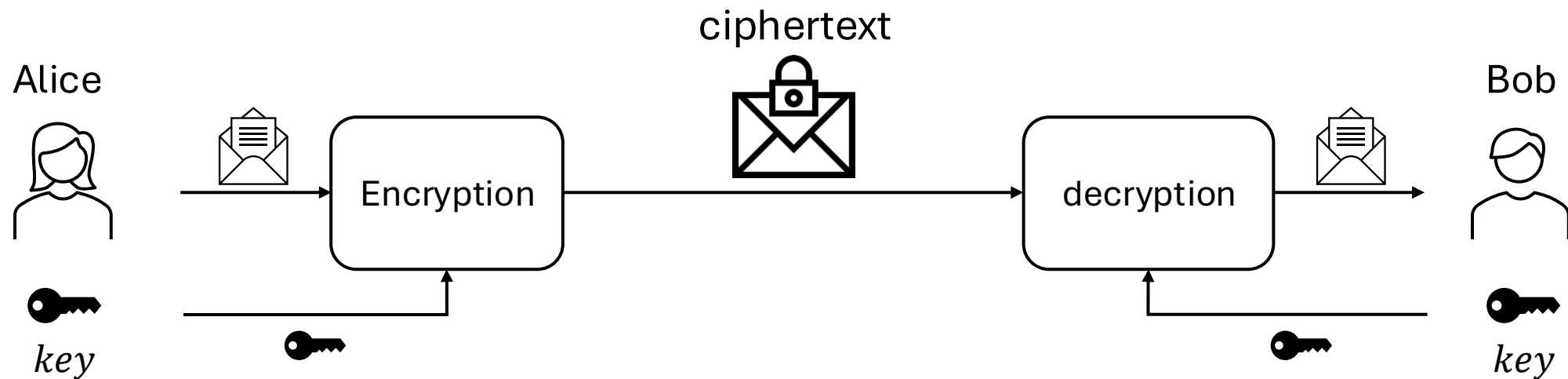
Symmetric Primitives - SKE

- Symmetric-key Encryption



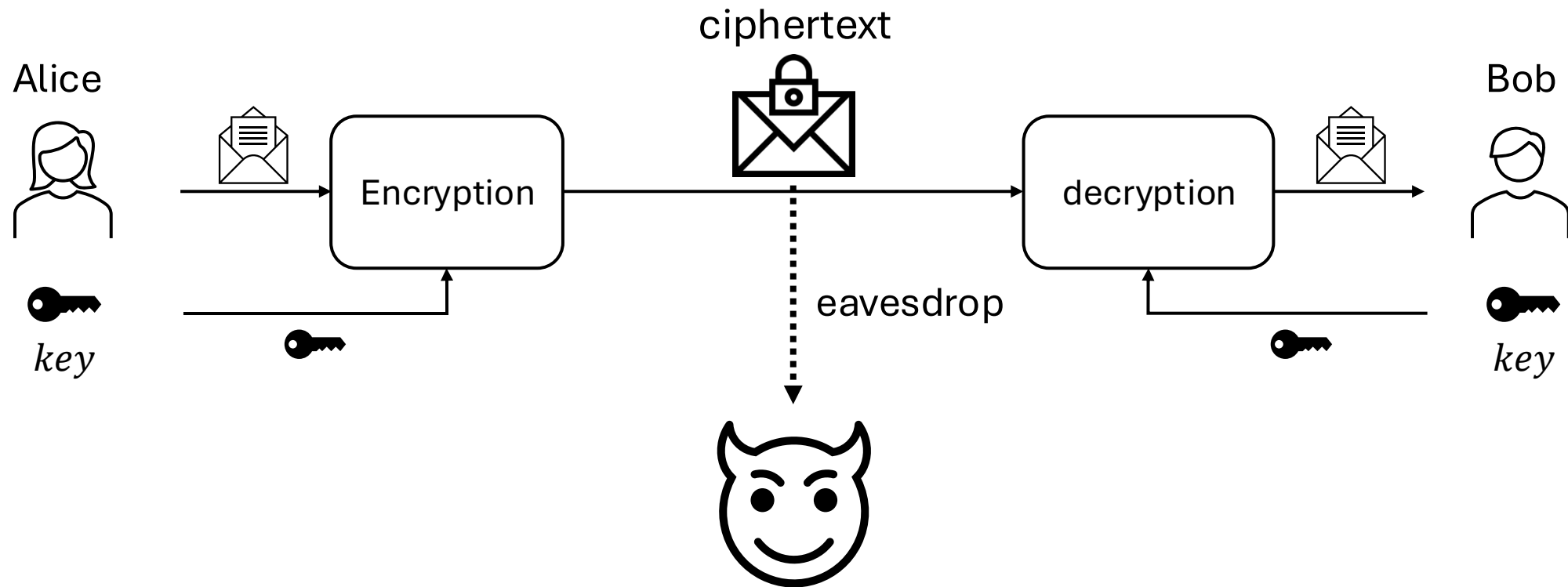
Symmetric Primitives - SKE

- Symmetric-key Encryption



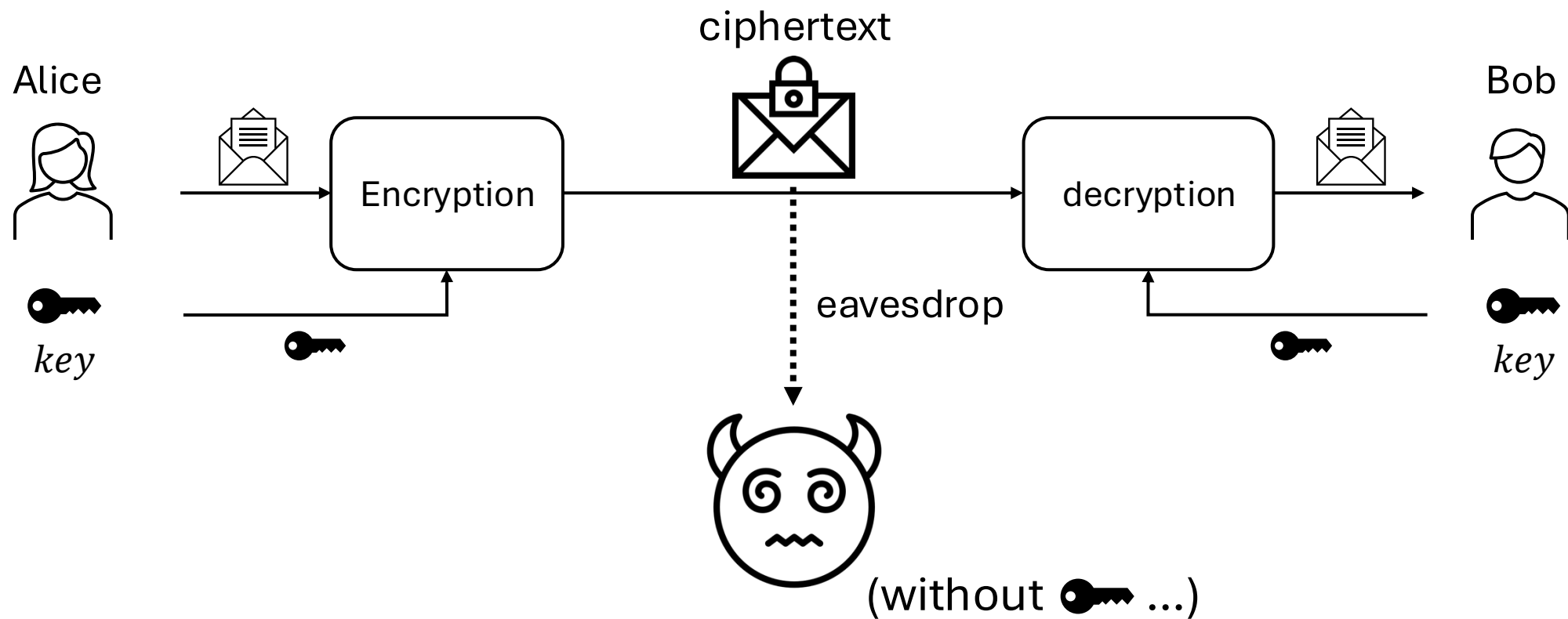
Symmetric Primitives - SKE

- Symmetric-key Encryption



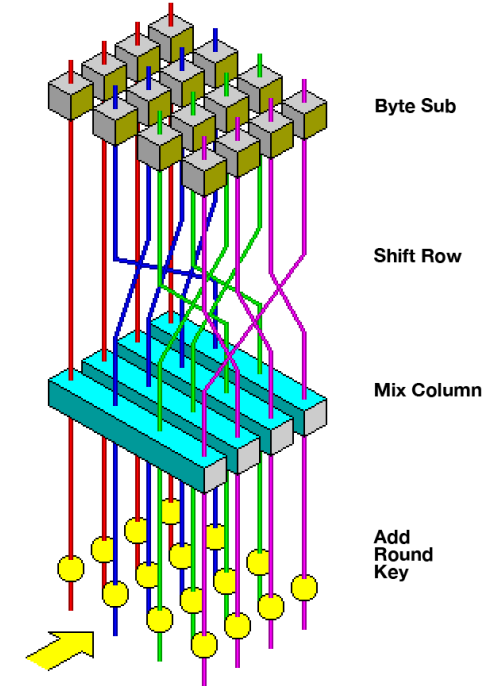
Symmetric Primitives - SKE

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



Symmetric Primitives - SKE

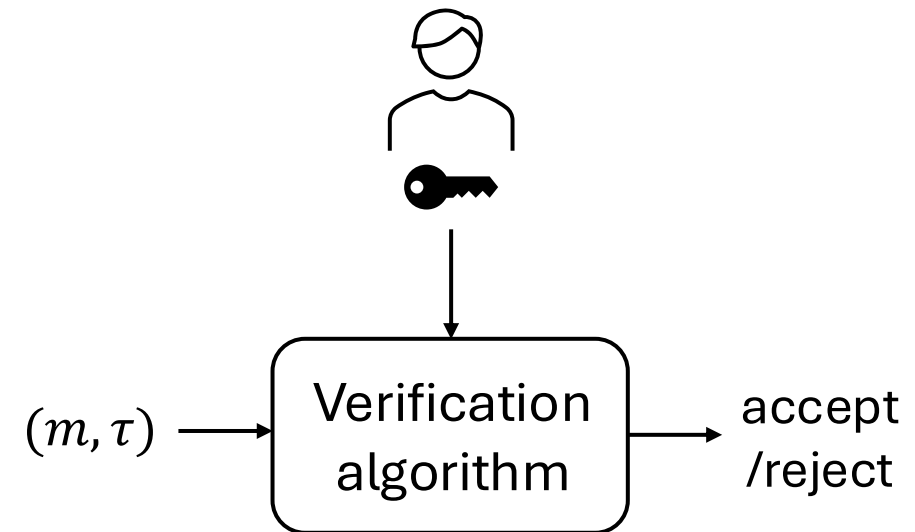
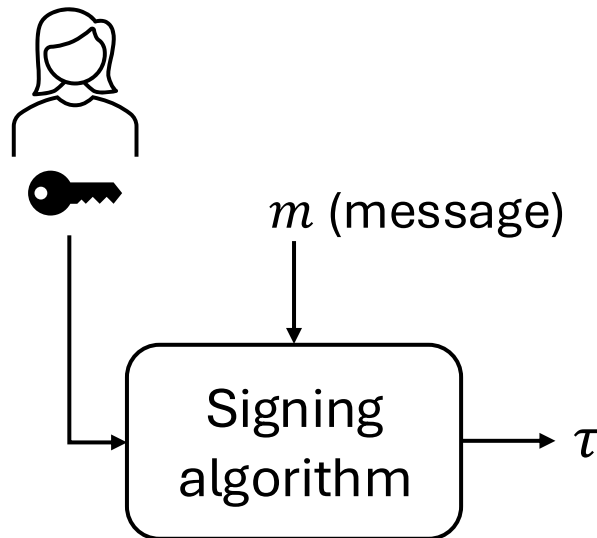
- Symmetric-key Encryption
 - AES (Advanced Encryption Standard)
 - Fixed-length encryption (block cipher)
- Extend to arbitrary-length encryption **via Mode of Operation**
 - CBC, CTR, ...



(Image from
Wikipedia)

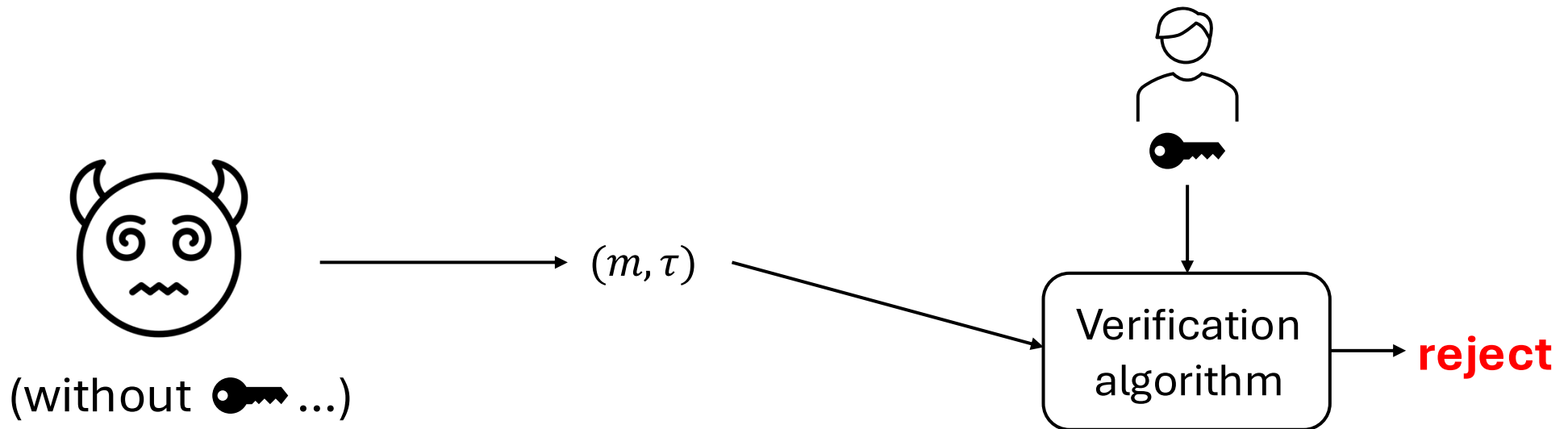
Symmetric Primitives - MAC

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



Symmetric Primitives - MAC

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



Symmetric Primitives - KDF

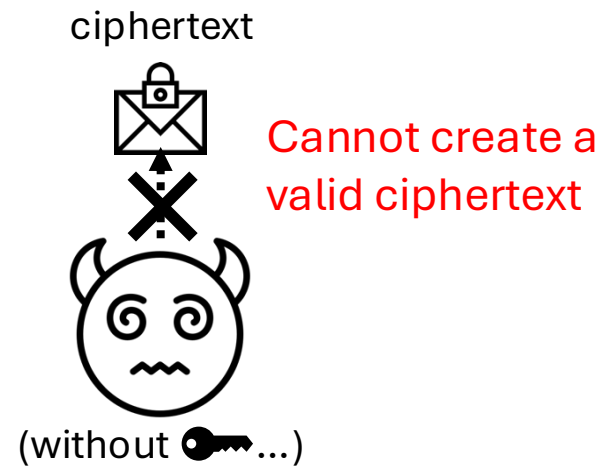
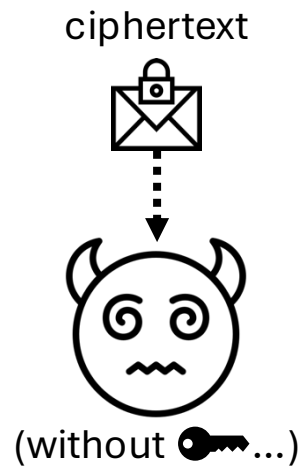
- Key Derivation Function (KDF)

$\text{KDF}(\text{"...shared secret with randomness..."}) = \text{a symmetric key}$

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

Symmetric Primitives - AE

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



Symmetric 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, ...

Next Topic

- More discussions on hash functions and AE(AD).

Exercise

- Practice your Rust skill

1. Hash functions:

(1) Input a file; (2) Compute its digital fingerprint (using SHA3-256); (3) Verify the digital fingerprint.

2. AES:

(1) Generate a random key; (2) Encrypt an arbitrary string using AES; (3) Decrypt the ciphertext. (4) Modify the ciphertext and decrypt it (using the same key) again.

3. HMAC:

(1) Generate a random key; (2) Generate a MAC code for an arbitrary string using HMAC; (3) Verify it.
(4) Modify the MAC code and verify it again with the original string.

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 data) and decrypt again.

5. KDF: Using HKDF to derive a key (using arbitrary inputs).

Exercise

- Practice your understanding about cryptography
1. What are the relations between **Collision Resistance**, **Preimage resistance**, and **Second Preimage resistance**?
 2. Here is the construction of HMAC:

$$\text{HMAC}(K, m) = H\left((K' \oplus \text{opad}) \parallel H((K' \oplus \text{ipad}) \parallel m)\right)$$

where H is the underlying hash function, opad and ipad are some fixed bit strings, and

$$K' = \begin{cases} K \\ H(K) \end{cases} \quad \text{if } K \text{ is larger than the output length of } H.$$

If you modify m , then what property of H prevents the original HMAC tag from remaining valid (under this modified m)?

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>