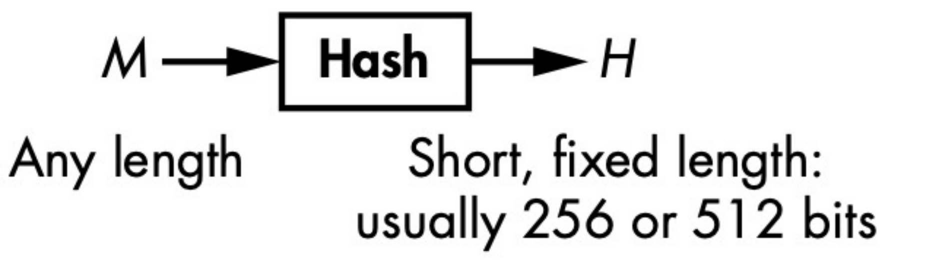
Cryptography

Keyed Hashing

Content

Introduction

- Hash functions take a long input and produce a short output.
  - The output is called a *hash value* or *digest*



Introduction

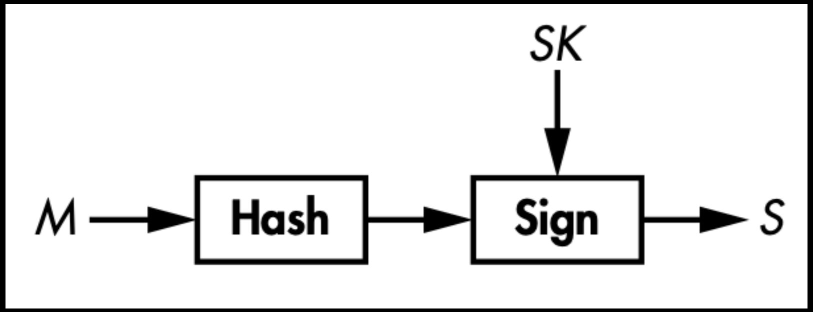
- Hash functions protect data integrity.
  - Ensures that the data has not been modified.
  - The data can be clear or encrypted.

- Secure hash function = unique hash value for each input.
  - A hash value is like the fingerprint.

- Example:
  - $hash(0101010) = XYZ$
  - $hash(010101\textcolor{red}{1}) = ABC$

Introduction

- In digital signatures, applications signs the hash of a message.



- Signing a message's hash is as secure as signing the message itself.
- Signing a short hash is faster than signing a large message.

Introduction

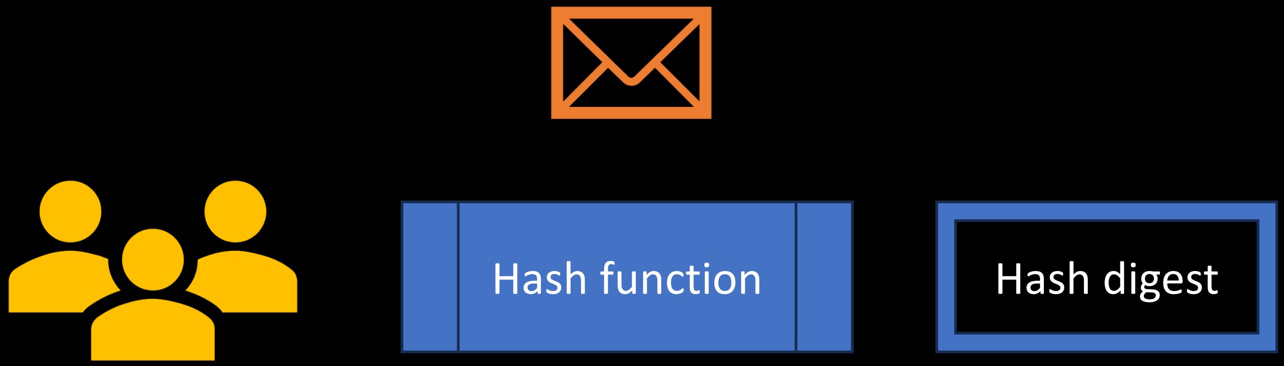
- The output of hash function should be unpredictable.

```
SHA-256("a") = 87428fc522803d31065e7bce3cf03fe475096631e5e07bbd7a0fde60c4cf25c7
SHA-256("b") = a63d8014dba891345b30174df2b2a57efbb65b4f9f09b98f245d1b3192277ece
SHA-256("c") = edeaaff3f1774ad2888673770c6d64097e391bc362d7d6fb34982ddf0efd18cb
```

- If you know the hash of “a”, “b”, and “c”, you cannot predict the hash of ”d”.
- Secure hash functions are PRFs.

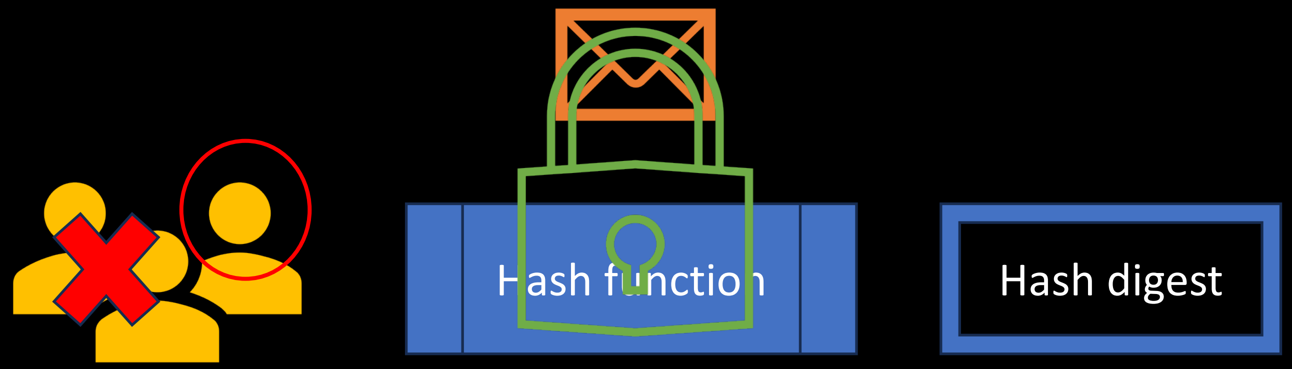
Introduction

- Hashing does not involve any secret information.
  - Anyone can take a message, hash it, and compare it to a specific value.



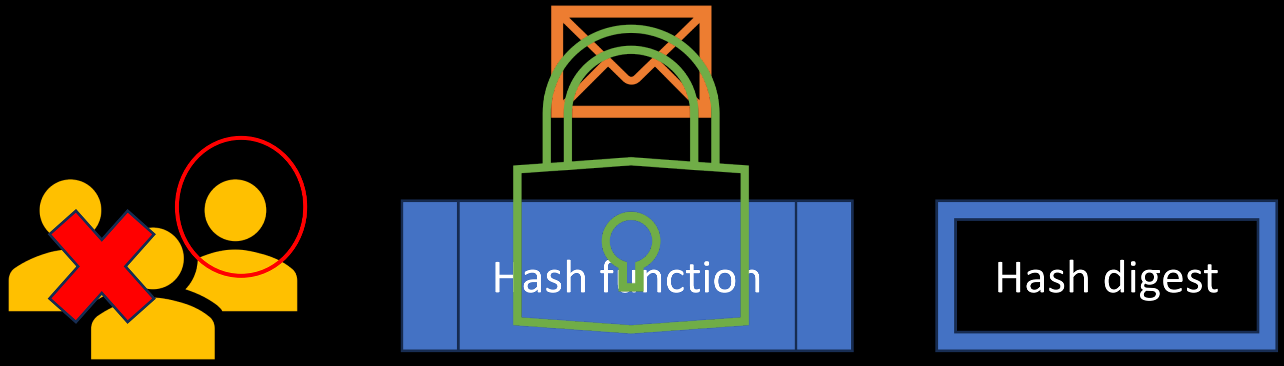
Introduction

- What if you don’t want anyone compute the hash?



Introduction

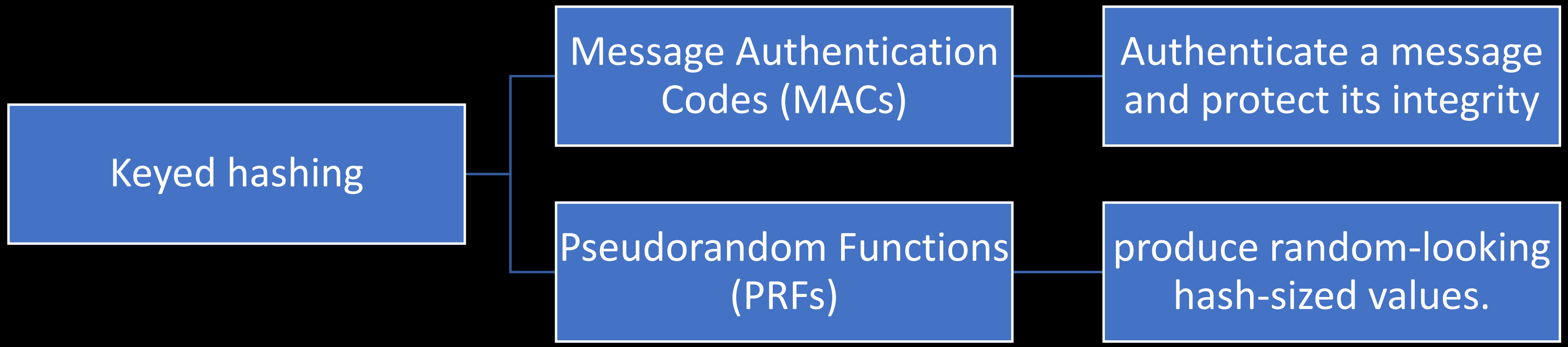
- What if you don’t want anyone compute the hash?



- **Keyed hashing**: hashing with secret keys
- Maintain the integrity and the authenticity of the data

Introduction

- **Keyed hashing**:



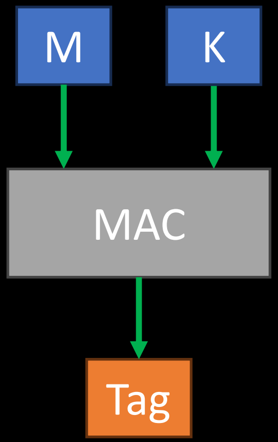
Content

Content
Introduction
Message Authentication Codes (MACs)
Pseudorandom Functions (PRFs)
Creating Keyed Hashes from Unkeyed Hashes
Creating Keyed Hashes from Block Ciphers: CMAC
Dedicated MAC Designs

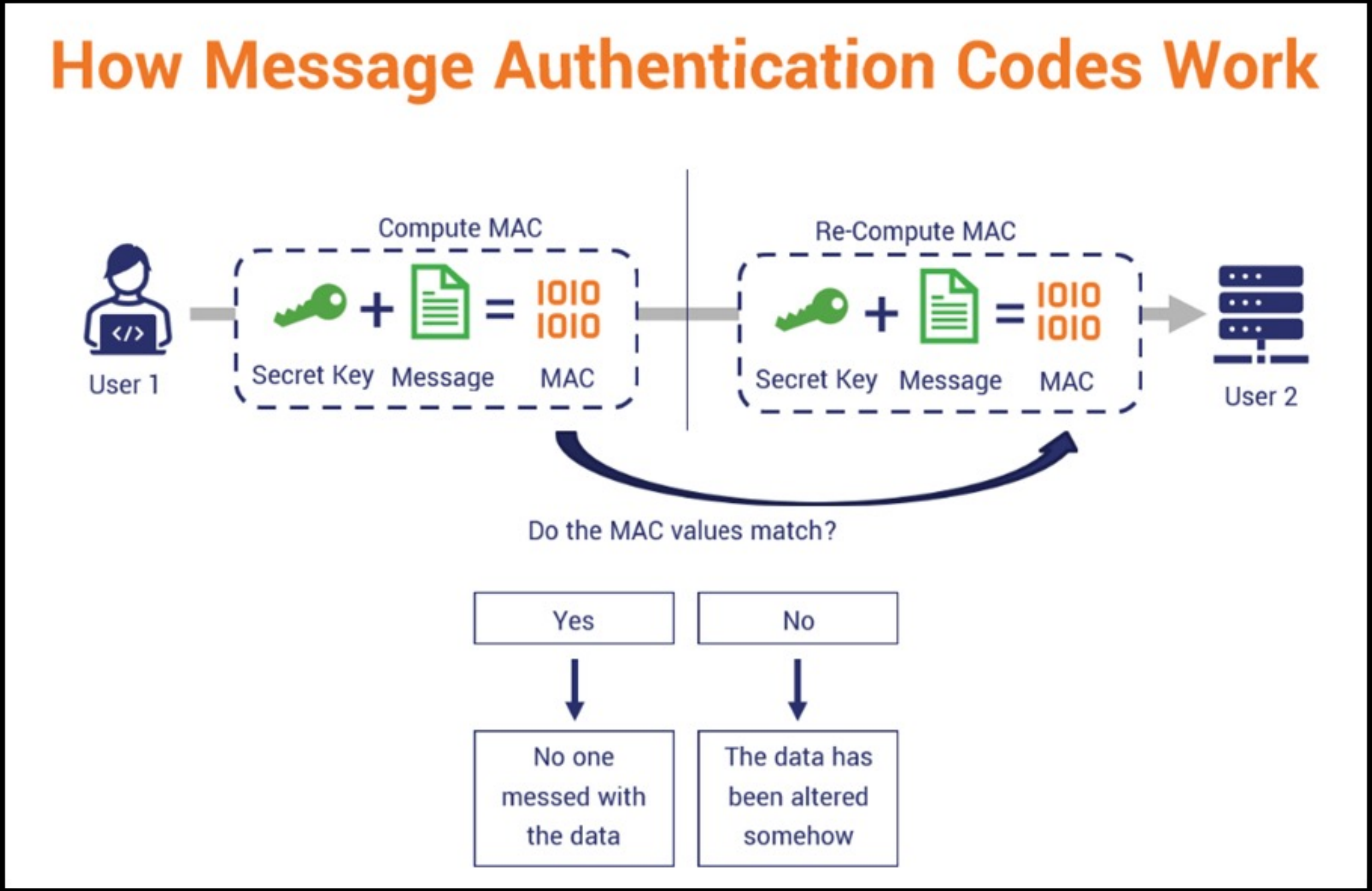
Message Authentication Codes (MACs)

- MACs protect:
  - **Integrity**: the message hasn't been altered
  - **Authenticity**: receiving the message from the intended sender

- $T = MAC(K, M)$ → hash the message M using the key K to produce a tag T



Message Authentication Codes (MACs)



Message Authentication Codes (MACs)

- Cipher + MAC are used in secure communication systems.
- Cipher + MAC → confidentiality, integrity, authenticity.

- Example protocols – generate a MAC for each packet:
    - IPSec
    - SSH
    - TLS

Message Authentication Codes (MACs)

- Not all communication systems use MACs.

- An authentication tag adds overhead to each packet
  - 64 to 128 bits

- 3G and 4G encrypt the voice packet without authentication
  - If an attacker damages an encrypted voice packet, it will decrypt to noise, which would sound like static.

Message Authentication Codes (MACs)

What does it mean for a MAC to be secure?

1. An attacker cannot create a tag without knowing the key
   1. **Forgery** is made up message/key pair

2. **Unforgeability**: an attacker cannot create fake message/key pair

3. Impossible to recover the secret key from a list of tags

Message Authentication Codes (MACs)

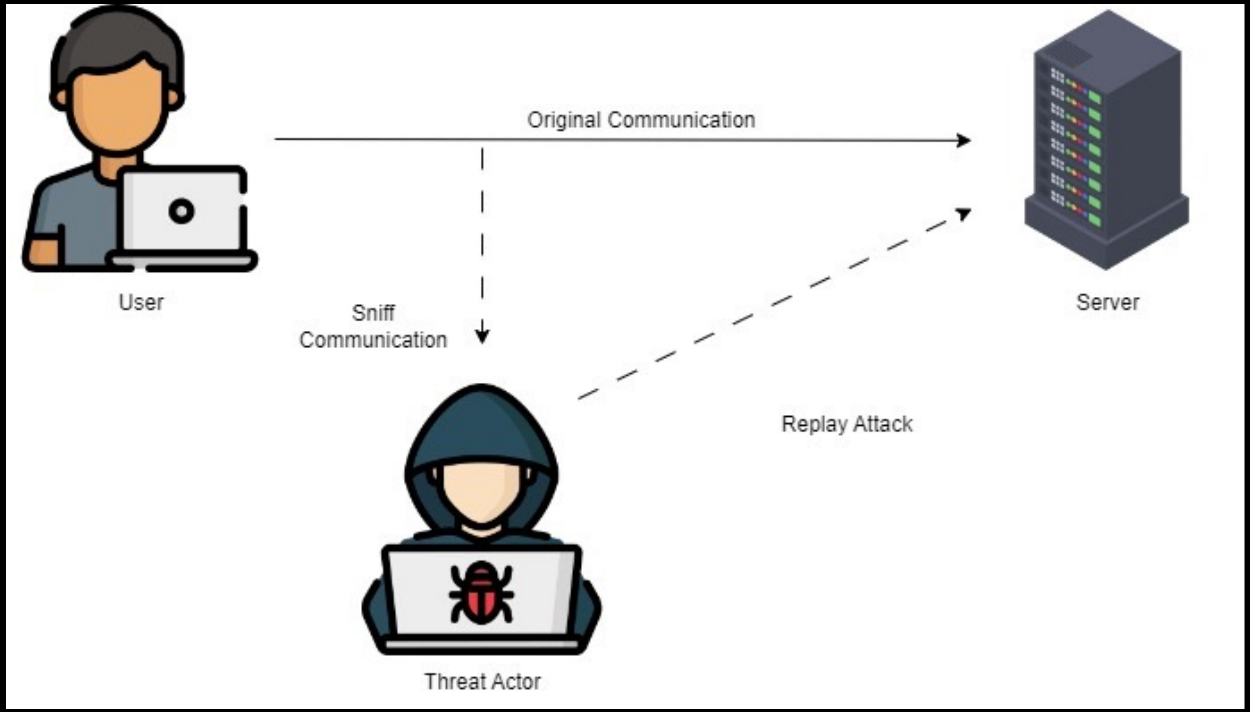
What can an attacker do to break a MAC? (attack model)

1. **Known-message attack** collects messages and their associated tags
   - Basic model

2. **Chosen-message attack** attackers get tags for messages of their choice
   - Standard model

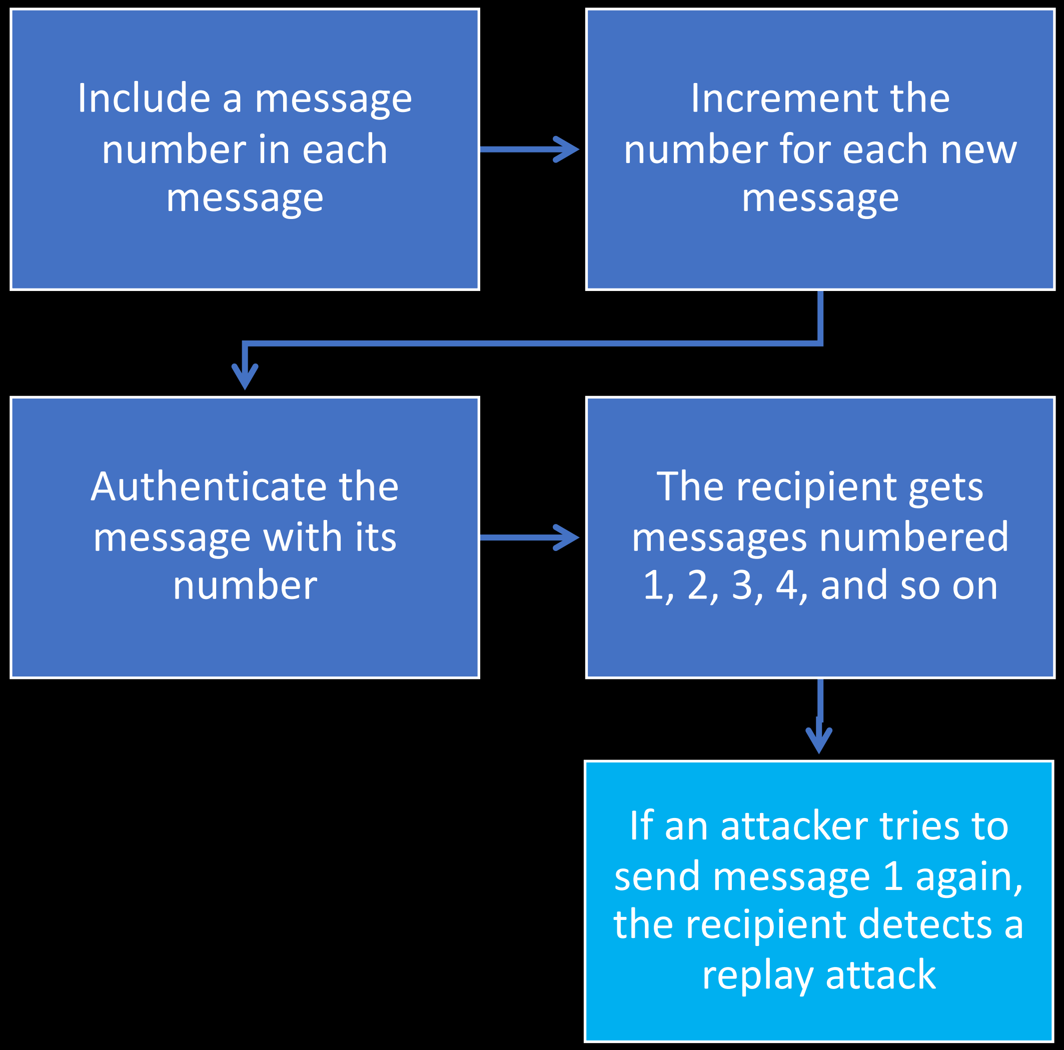
Message Authentication Codes (MACs)

- MACs are susceptible to **replay attacks**



Message Authentication Codes (MACs)

To prevent replay attacks



Content

Content
Introduction
Message Authentication Codes (MACs)
Pseudorandom Functions (PRFs)
Creating Keyed Hashes from Unkeyed Hashes
Creating Keyed Hashes from Block Ciphers: CMAC
Dedicated MAC Designs

Pseudorandom Functions (PRFs)

- $PRF(K, M)$ a function that takes a secret key and returns a random output

- PRFs are used as part of other algorithms and protocols:
  - Create block ciphers within the Feistel structure
  - Key derivation functions – generate cryptographic keys from a password
  - 4G protocol to authenticate SIM cards
  - TLS protocol to generate session-specific random values from a master key

Pseudorandom Functions (PRFs)

- PRFs security → <u>indistinguishability</u> from a random function
  - They do not have patterns → the output is pseudorandom but not truly random

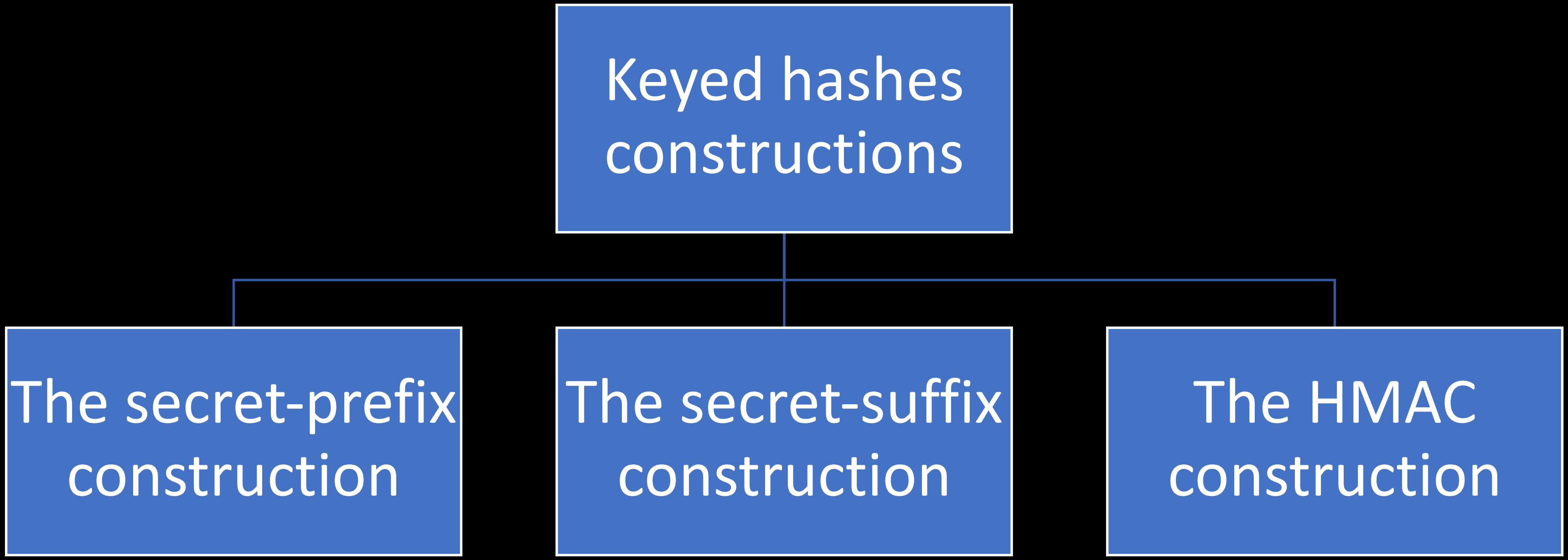
<table>
<tr><th>PRF</th><th>MAC</th></tr>
<tr><td colspan="2">Both are keyed hashes</td></tr>
<tr><td colspan="2">PRFs are stronger than MACs</td></tr>
<tr><td>Security: outputs are indistinguishable from random</td><td>Security: tags cannot be forged</td></tr>
<tr><td colspan="2">Secure PRF → Secure MAC
The converse is not true</td></tr>
</table>

Content

Content
Introduction
Message Authentication Codes (MACs)
Pseudorandom Functions (PRFs)
Creating Keyed Hashes from Unkeyed Hashes
Creating Keyed Hashes from Block Ciphers: CMAC
Dedicated MAC Designs

Keyed Hashes from Unkeyed Hashes

- PRFs and MACs are designed from existing hash functions or block ciphers



Keyed Hashes from Unkeyed Hashes

The secret-prefix construction

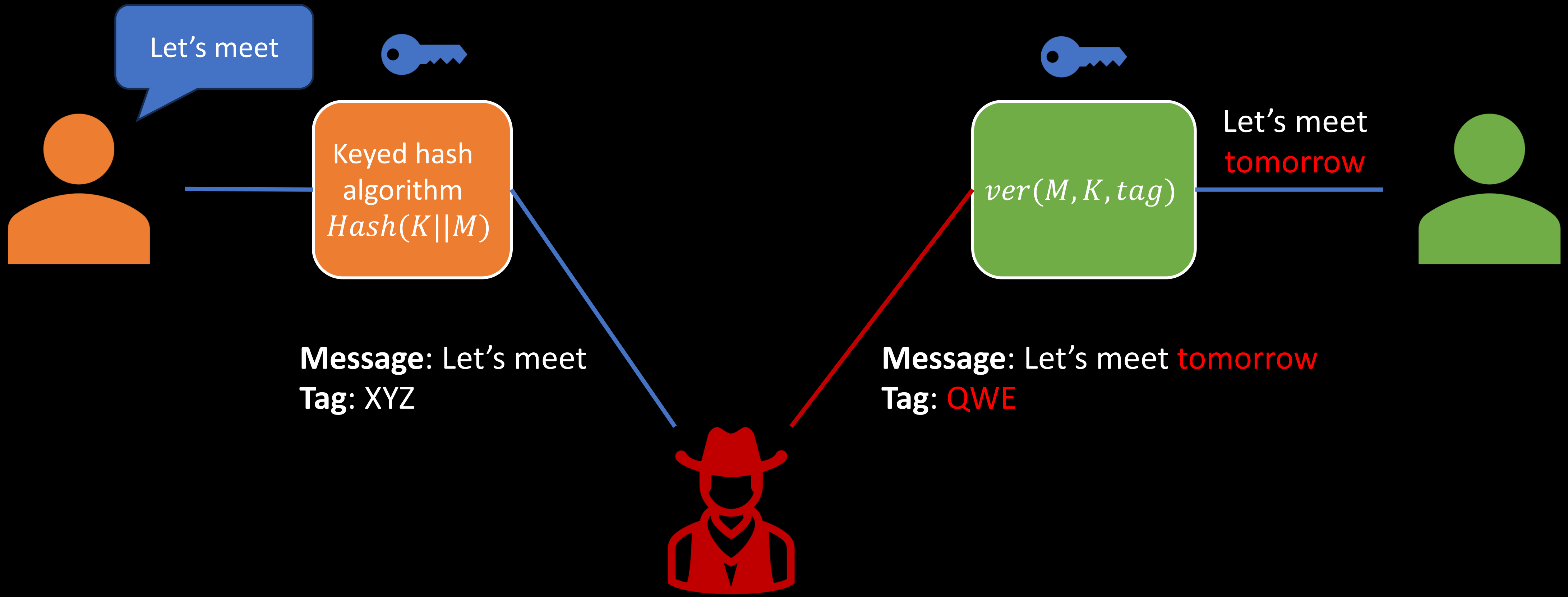
- Prepends a key to the message and compute $Hash(K||M)$
- Insecure against length-extension attacks
  - Attackers compute $Hash(K||M_1||M_2)$ given only $Hash(K||M_1)$ without M_1 nor K
- Insecure when the hash support keys with different length
  - You get the same hash for different messages by manipulating the key

Keyed Hashes from Unkeyed Hashes

The secret-prefix construction – <u>normal scenario</u>

Keyed Hashes from Unkeyed Hashes

The secret-prefix construction – length extension attack



Keyed Hashes from Unkeyed Hashes

What hash functions that are vulnerable to the length-extension attack?

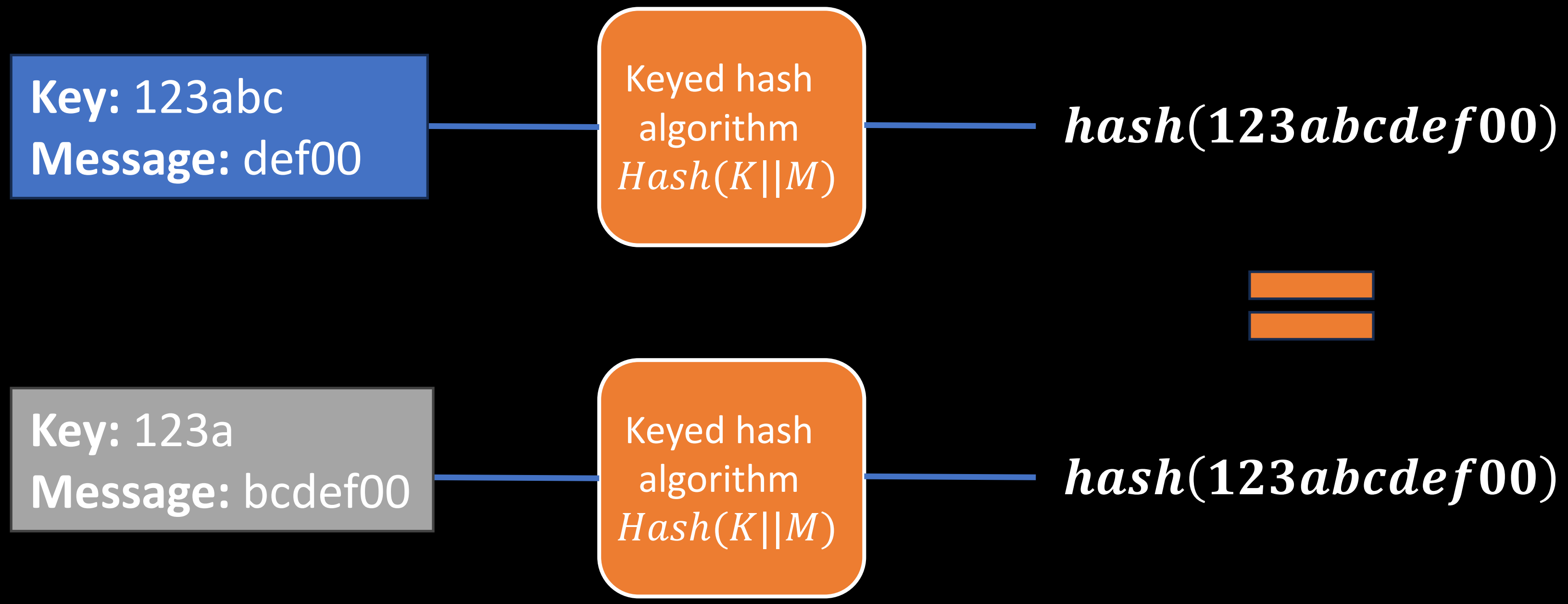
Keyed Hashes from Unkeyed Hashes

What hash functions that are vulnerable to the length-extension attack?

- All hash functions that are based on the Merkle–Damgård construction
- This includes SHA2 family: SHA256 and SHA512
- SHA3 and BLAKE2 are secure

Keyed Hashes from Unkeyed Hashes

The secret-prefix construction – <u>Insecurity with Different Key Lengths</u>



Keyed Hashes from Unkeyed Hashes

The secret-suffix construction

- Appends a key to the message and compute $Hash(M||K)$
  - Secure against length-extension attack

- Insecure when the hash function is vulnerable to collisions:
  - If $Hash(M_1) = Hash(M_2) \rightarrow Hash(M_1||K) = Hash(M_2||K)$

Keyed Hashes from Unkeyed Hashes

The Hash-based MAC (HMAC) construction

- More secure than the previous constructions

- Used in the IPSec, SSH, and TLS protocols

- $Hash((K \oplus opad)||Hash((K \oplus ipad)||M))$
  - $opad$: outer padding is a string $(5c5c5c \ldots)$ that as long as $\mathrm{H}ash$’s block size
  - $ipad$: inner padding is a string $(3636 \ldots)$ that is as long as the $Hash$’s block size

Keyed Hashes from Unkeyed Hashes

The Hash-based MAC (HMAC) construction

- $Hash((K \oplus opad)||Hash((K \oplus ipad)||M))$

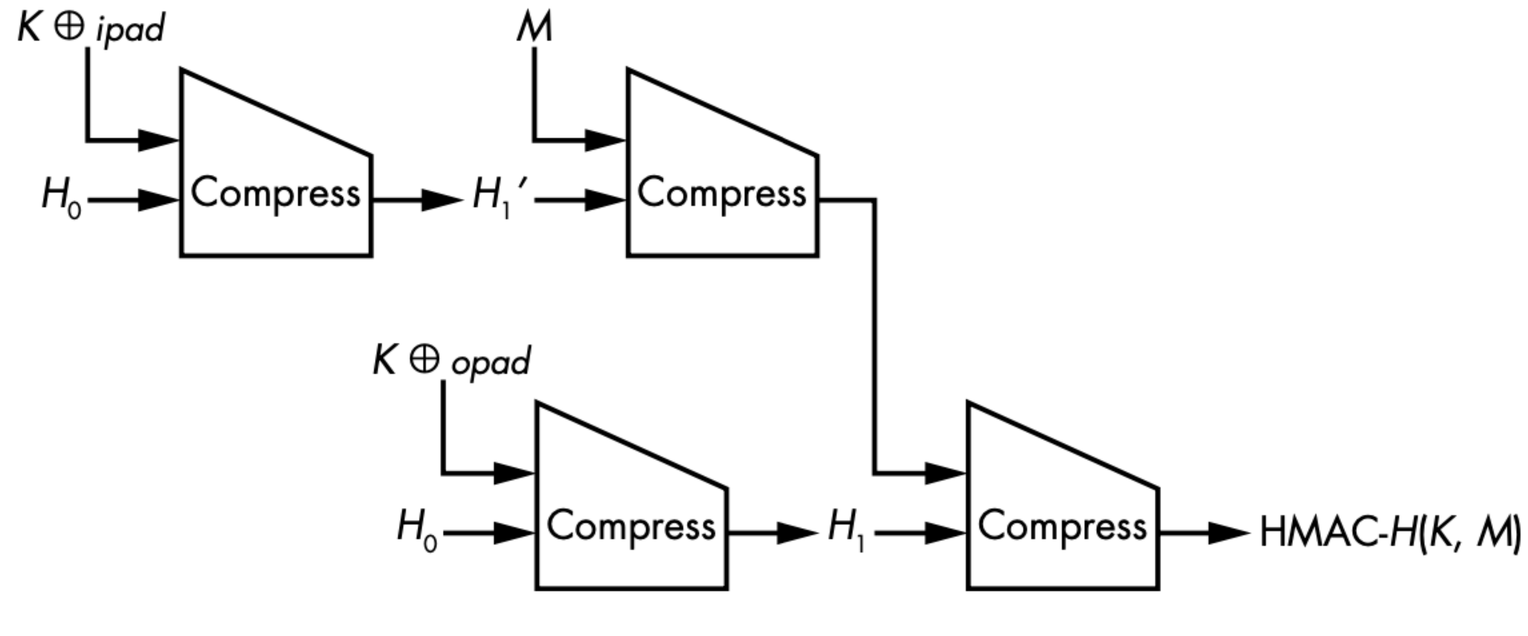


Figure 7-1: The HMAC hash-based MAC construction

Content

Content
Introduction
Message Authentication Codes (MACs)
Pseudorandom Functions (PRFs)
Creating Keyed Hashes from Unkeyed Hashes
Creating Keyed Hashes from Block Ciphers: CMAC
Dedicated MAC Designs

Keyed Hashes from Block Ciphers: CMAC

- Instead of using hash functions, use block ciphers

- CMAC is used in the internet key exchange protocol (IKE)
  - IKE is used to establish a secure, authenticated channels between two parties
  - IKE is part of the IPSec suite
  - AES-CMAC-PRF-128: https://www.rfc-editor.org/rfc/rfc4615

Content

Content
Introduction
Message Authentication Codes (MACs)
Pseudorandom Functions (PRFs)
Creating Keyed Hashes from Unkeyed Hashes
Creating Keyed Hashes from Block Ciphers: CMAC
Dedicated MAC Designs

Dedicated MAC Designs

- Instead of building PRFs/MACs based on hash functions/block ciphers
- Use specific (independent) designs for better efficiency
- The most common dedicated MAC designs:
  - Poly1305
  - SipHash

Dedicated MAC Designs

- Poly1305 is designed to be fast on modern CPUs
- Used by Google to secure HTTPs connections
- Used in the OpenSSH suite
- Poly1305 is built on two techniques:
  - Universal hash functions
  - Wegman-Carter construction.

Dedicated MAC Designs

- The universal hash (UH) is <u>weaker</u> than a crypto hash but <u>faster</u>
- The UH of Poly1305 is a <u>polynomial-evaluation</u> hash

$$\mathbf{UH}(R, K, M) = R + M_1K + M_2K^2 + M_3K^3 + \ \ldots \ + M_nK^n \mod p$$

  - p is a prime number used as a modulo
  - R and K are key values in the range $[1{:}\,p]$
  - M is the message consisting of n blocks
- The block size is 128 bits
- The prime number is slightly larger than 2^{128}, e.g., $2^{128} + 51$

Dedicated MAC Designs

- What is the issue of this universal hash function?

$$\mathbf{UH}(R, K, M) = R + M_1K + M_2K^2 + M_3K^3 + \ \ldots \ + M_nK^n \mod p$$

Dedicated MAC Designs

- What is the issue of this universal hash function?

$$\mathbf{UH}(R, K, M) = R + M_1K + M_2K^2 + M_3K^3 + \ \ldots \ + M_nK^n \bmod p$$

1. $M = (M_1, M_2, \ldots, M_n) = (0, 0, \ldots, 0)$ → leaks R
2. $M = (M_1, M_2, \ldots, M_n) = (1, 0, \ldots, 0)$ → leaks K by subtracting R from the tag
3. It can authenticate one message only

Dedicated MAC Designs

- The Wagman-Carter construction → UH authenticates multiple messages
- It uses the UH, a PRF, a nonce N, and two keys K_1 and K_2 to build a MAC

$$\mathbf{MAC}(K_1, K_2, N, M) = \mathbf{UH}(K_1, M) + \mathbf{PRF}(K_2, N)$$

  - N is a unique nonce for each K_2
  - The secure PRF hides the weakness of the UH
  - The output values of UH and PRF are long enough to ensure high security.

Dedicated MAC Designs

- An instantiation of the Wagman-Carter

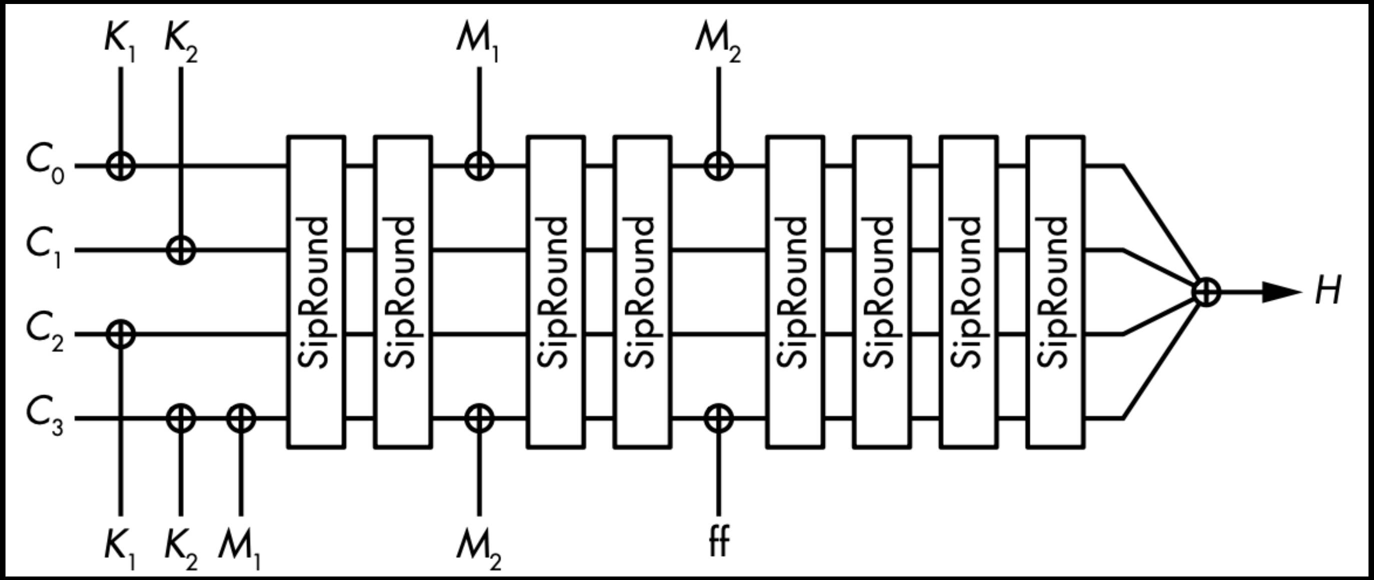
$$\mathbf{Poly1305}(K_1, M) + \mathbf{AES}(K_2, N) \bmod 2^{128}$$

  - AES is a PRP not a PRF
  - The construction works with either a PRP or a PRF

- The Poly1305 UH can be combined with other algorithms
  - e.g., ChaCha stream cipher
- Poly1305 is optimized for long messages, takes long time for short messages

Dedicated MAC Designs

- SipHash is designed to be efficient for short messages
  - The key is 128 bits, seen as two 64-bit words
  - (C_0, C_1, C_2, C_4) is 256-bit initial state, seen as four 64-bit words
  - SipRound is the core function
  - The output tag is 64-bit



TASK

Use $pycryptodome$ to implement:

- HMAC_SHA256 and HMAC_SHA512
- Poly1305_AES
- CMAC_AES_128
- SipHash_2_4 (# $pip\ install\ siphash$)
- Compare the running time of the previous constructions. Run each algorithm 10,000 times.
- Simulate a timing attack on SipHash, assuming the tag size is 4 bytes