

# CS 453/698: Software and Systems Security

## **Module: Usable Security**

Lecture: Authentication and attestation

Adam Caulfield

*University of Waterloo*

Spring 2025

# Reminders & Recap

## Reminders:

- [A3 is released](#)

## Recap – last time we covered:

Access control

- Policies
- Models
- Mechanisms

# Today

## **Authentication**

- Definitions and factors
- What is the adversary model?
- Password protocols
- Alternative methods

## **Attestation**

- Key differences from authentication
- Requirements
- Attestation protocols

# Authentication

**Definition:** *the process or action or proving something to be genuine, true, or valid. In computing, it refers to the process or action of verifying the identity of a user or process.*

**Interaction between two entities → Verifier and Prover**

## **Factors:**

- A Prover secret / identity
  - Password, PIN, answer to secret question, ....
  - Biometrics
- A component in use by the Prover
  - ATM card, badge, browser cookie, phone, ....

# Authentication

## System & Adversary Model



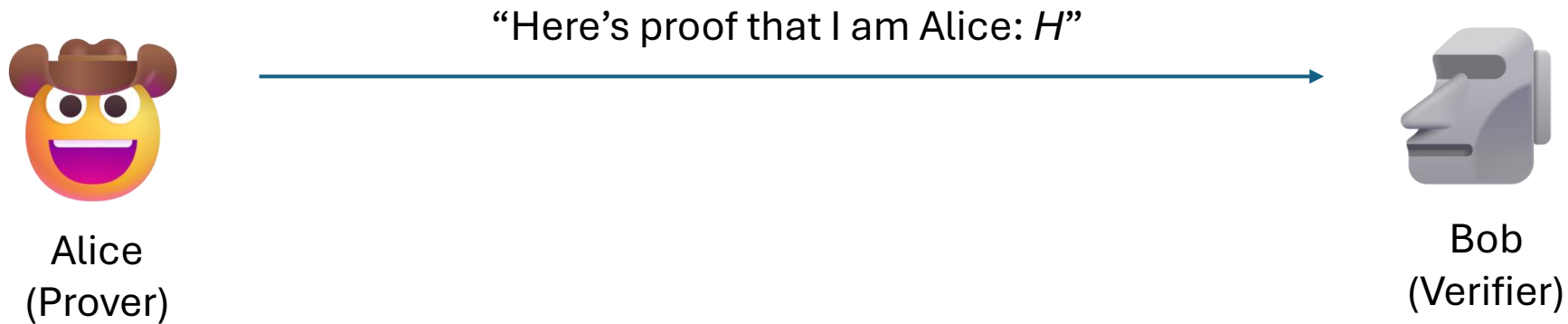
Alice



Bob

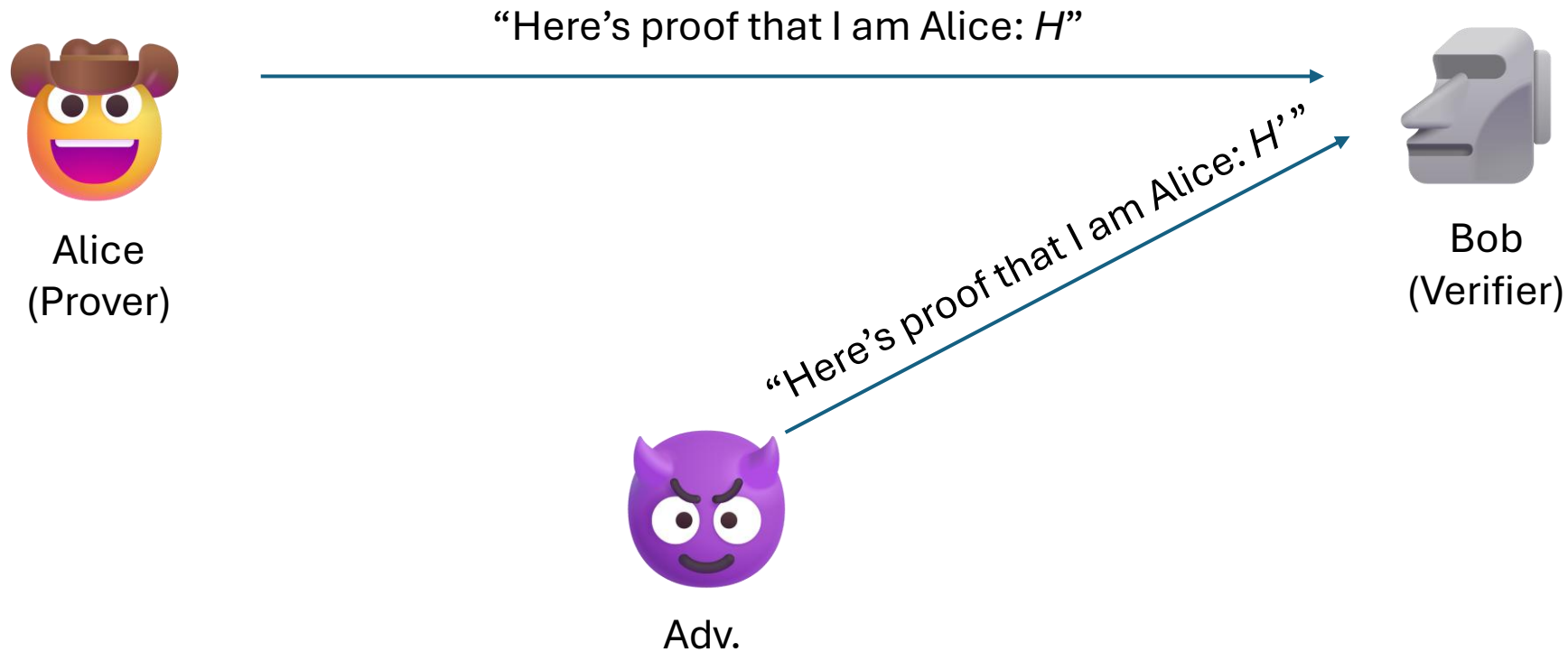
# Authentication

## System & Adversary Model



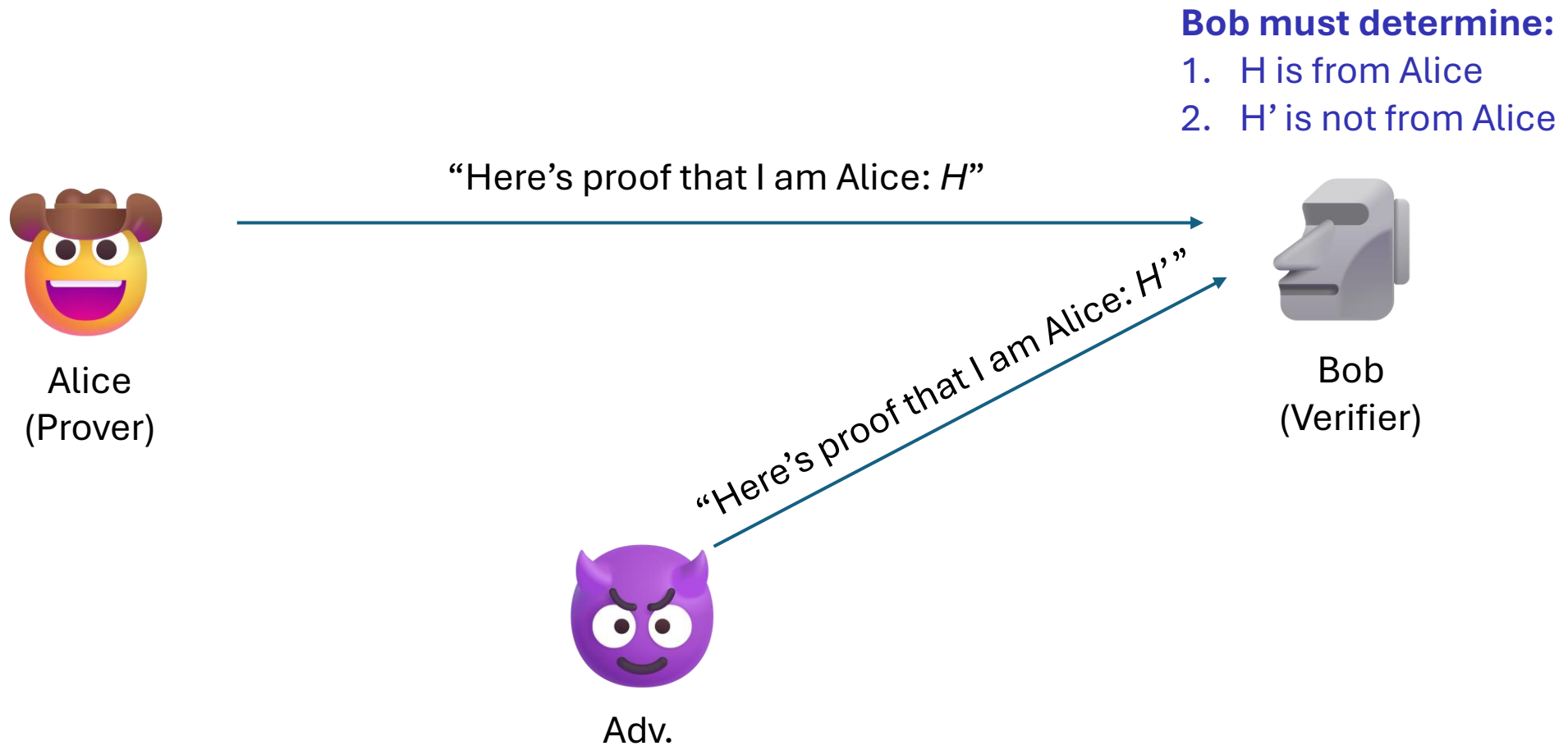
# Authentication

## System & Adversary Model



# Authentication

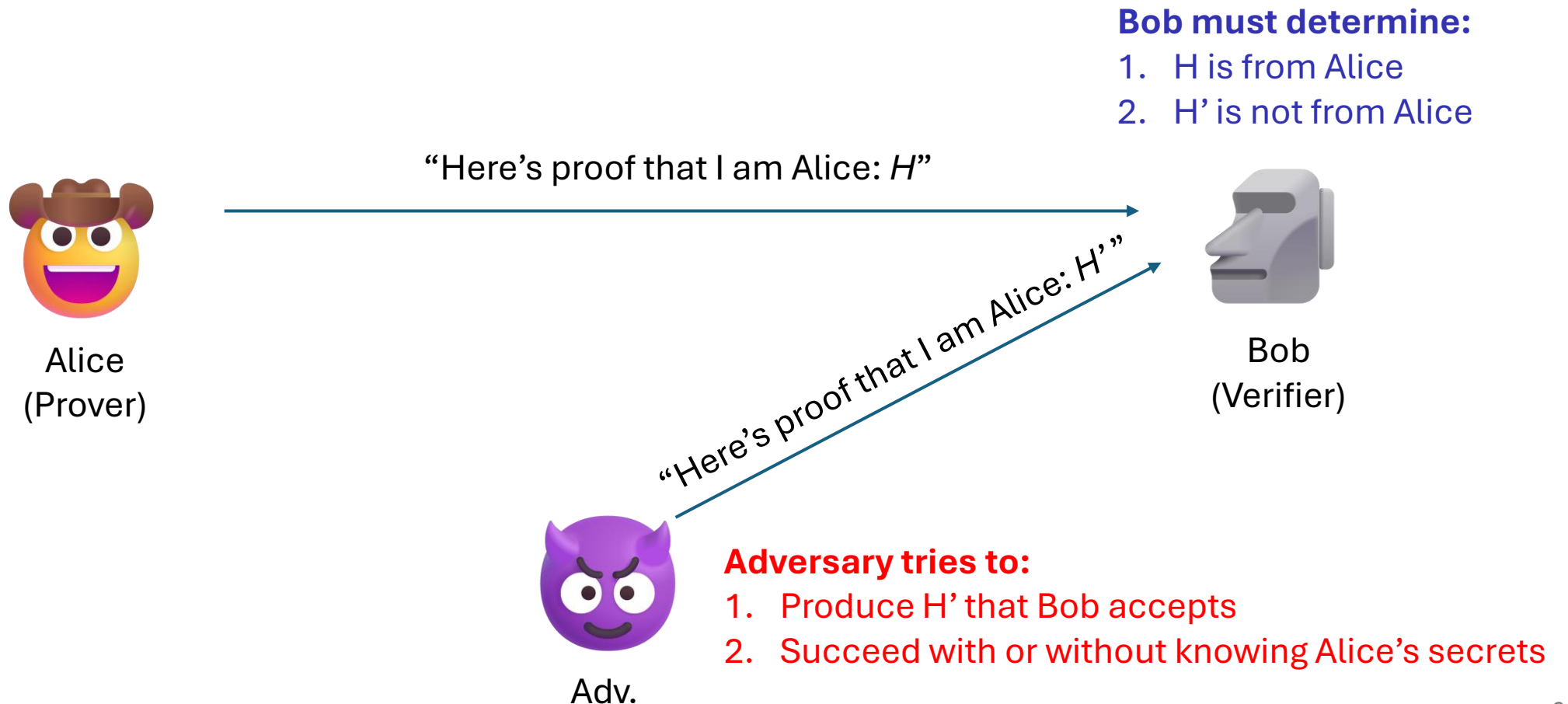
## System & Adversary Model





# Authentication

## System & Adversary Model



# Authentication

## System & Adversary Model

### System:

- Alice and Bob (and their devices) are honest
- Network availability (e.g., no denial of service)

### Adversary:

- Knows the algorithm used for producing and verifying proof
- Sits on the network
- Aims to forge proofs and claim false identity

# Authentication

## System & Adversary Model

### System:

- Alice and Bob (and their devices) are honest
- Network availability (e.g., no denial of service)

### Adversary:

- Knows the algorithm used for producing and verifying proof
- Sits on the network
- Aims to forge proofs and claim false identity

**Approach:** Passwords, public-key infrastructure (PKI), biometrics, etc.

# Authentication via Passwords

## Passwords:

- One of the oldest authentication mechanisms used in computer systems
- Origins: 1960s → [MIT's Compatible Time-Sharing System \(CTSS\)](#)

## Password guessing attacks:

- Brute force: [test  \$95^8\$  passwords in 5.5 hours using 25 GPUs](#)
- Enough to brute force every possible 8-character password containing:
  - Upper-case letters
  - Lower-case letters
  - Digits
  - Symbols

# Authentication via Passwords

Difficulty increase exponentially with length


But:

- Exhaustive search assumes people chose passwords randomly

People tend to create a structured password:

- Root: a core word
- Appendage: prefix or suffix on the core word
- Ex: “abc123” and “123abc”
- Ex: “rootGMail123” and “rootLinkedIn123”

June 2012 LinkedIn leak: [~6.5 million passwords leaked](#)

 KEEPER

How Long Would It Take To Crack Your Password?

	Lowercase Letters Only	At Least 1 Uppercase Letter	At Least 1 Uppercase Letter + Number	At Least 1 Uppercase Letter + Number + Symbol
1	Instantly	Instantly	-	-
2	Instantly	Instantly	Instantly	-
3	Instantly	Instantly	Instantly	Instantly
4	Instantly	Instantly	Instantly	Instantly
5	Instantly	Instantly	Instantly	Instantly
6	Instantly	Instantly	Instantly	Instantly
7	Instantly	Instantly	1 Minute	6 Minutes
8	Instantly	22 Minutes	1 Hour	8 Hours
9	2 Minutes	19 Hours	3 Days	3 Weeks
10	1 Hour	1 Month	7 Months	5 Years
11	1 Day	5 Years	41 Years	400 Years
12	3 Weeks	300 Years	2,000 Years	34,000 Years
13	1 Year	16,000 Years	100,000 Years	2 Million Years
14	51 Years	800,000 Years	9 Million Years	200 Million Years
15	1,000 Years	43 Million Years	600 Million Years	15 Billion Years
16	34,000 Years	2 Billion Years	37 Billion Years	1 Trillion Years

Source: security.org

[Source](#)

# Authentication via Passwords

## **NIST Guidelines for passwords (related to the phrase itself)**

- 15-64 characters
- Accept all printable ASCII characters
  - Including spacebar
- Require users change passwords if compromise
- Verifiers can compare passwords to blocklist containing:
  - Passwords obtained from previous breach corpus
  - Dictionary words
  - Context-specific words (e.g., name of the service)
- [And more from NIST...](#)

Testing strength of your password (via [security.org](#))

- Even better... ([The Password Game](#)) 😊

# Password Protocols

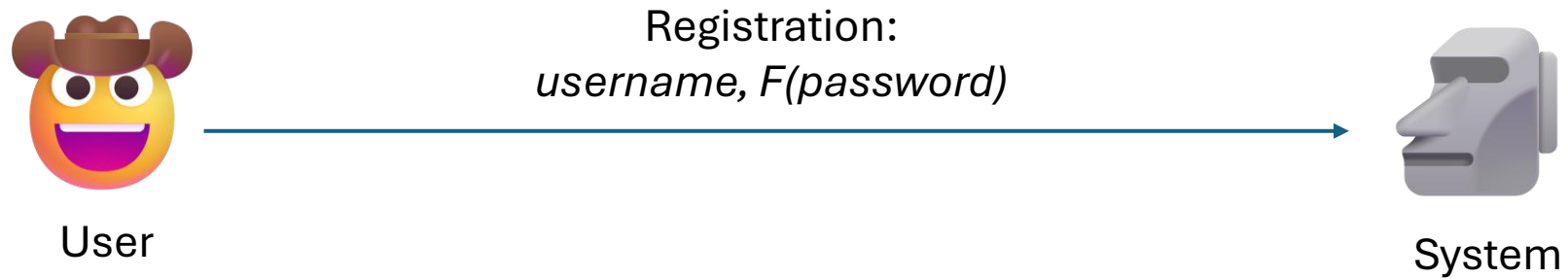
## **Formal modeling of passwords:**

Useful for examining the pros and cons of several password-based authentication protocols

# Password Protocols

## Formal modeling of passwords:

Useful for examining the pros and cons of several password-based authentication protocols

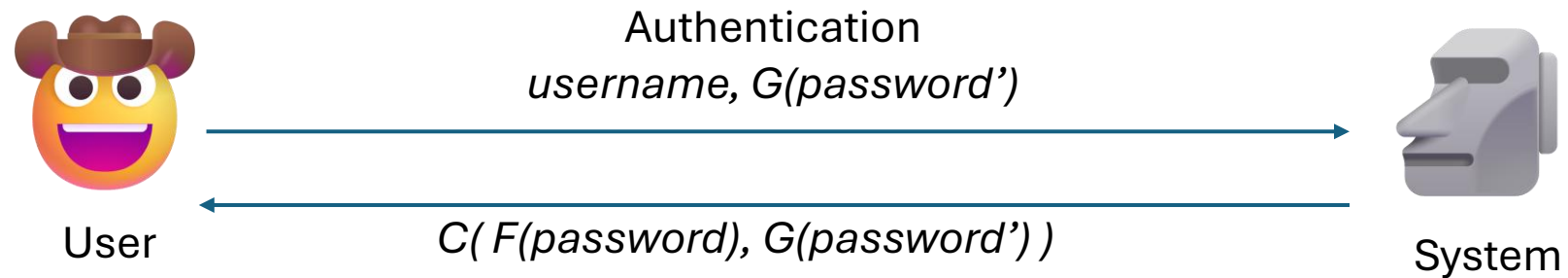
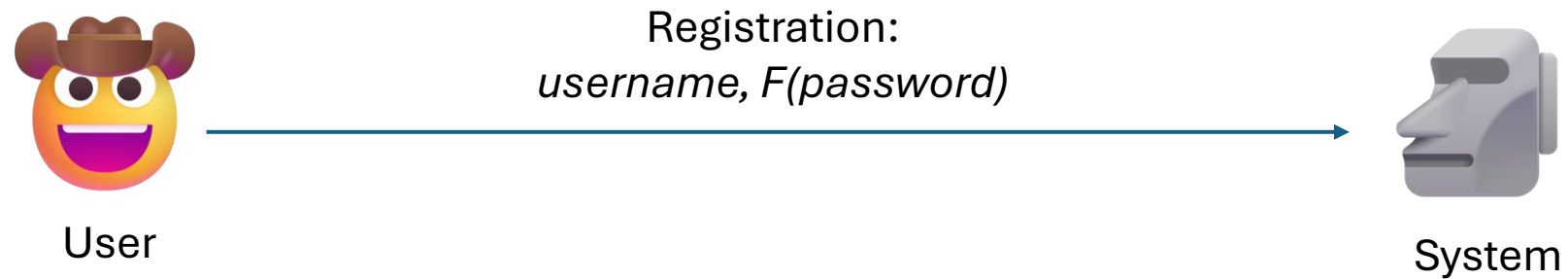




# Password Protocols

## Formal modeling of passwords:

Useful for examining the pros and cons of several password-based authentication protocols



# Password Protocols

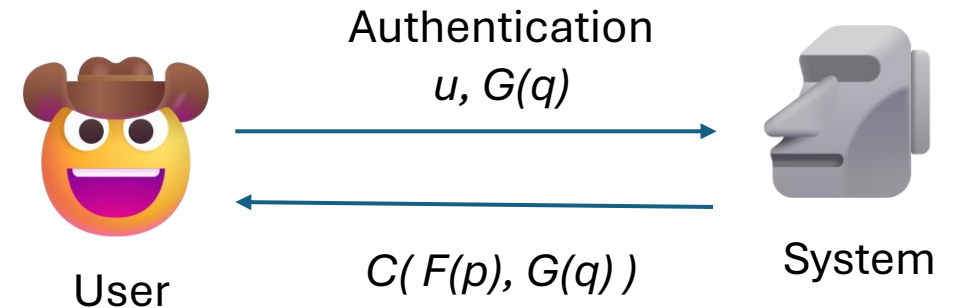
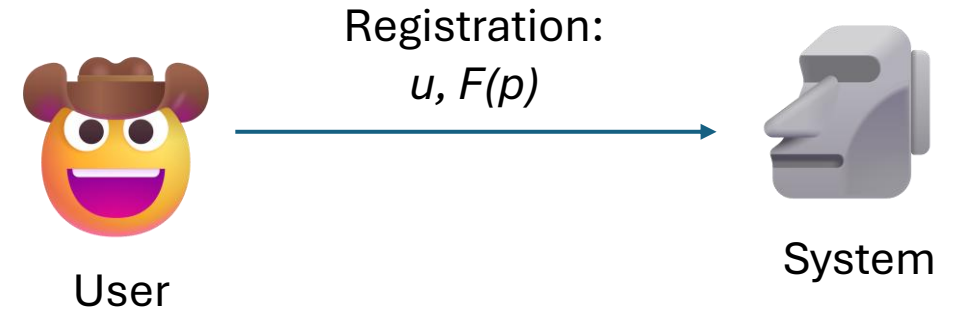
## Formal modeling of passwords:

$u$ : User identifier

$(p, q)$ : Passphrase at Reg./Auth. time

$(G, F)$ : mapping of phrase to token at Reg./Auth. Time

$C$ : function to evaluate correctness



# Password Protocols

## Formal modeling of passwords:

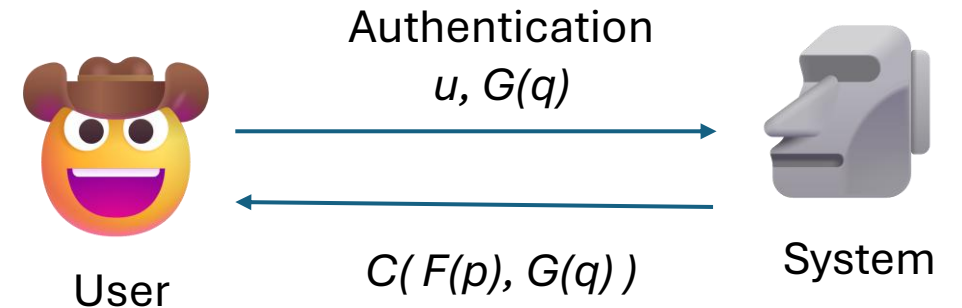
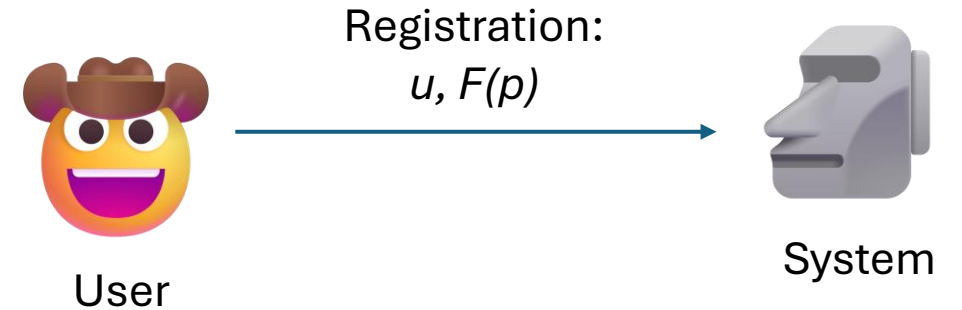
$u$ : User identifier

$(p, q)$ : Passphrase at Reg./Auth. time

$(G, F)$ : mapping of phrase to token at Reg./Auth. Time

$C$ : function to evaluate correctness

## What is the correctness requirement?



# Password Protocols

## Formal modeling of passwords:

$u$ : User identifier

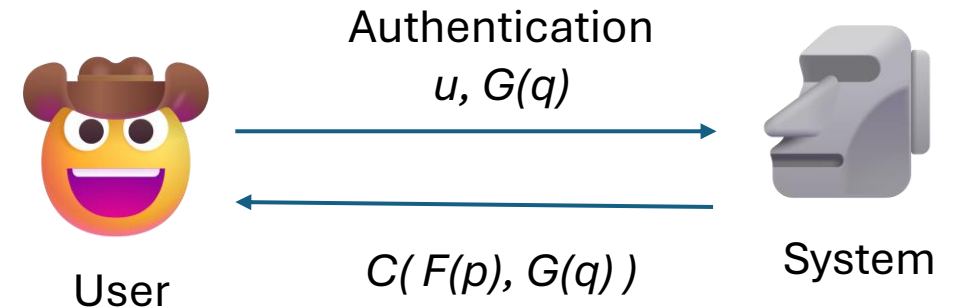
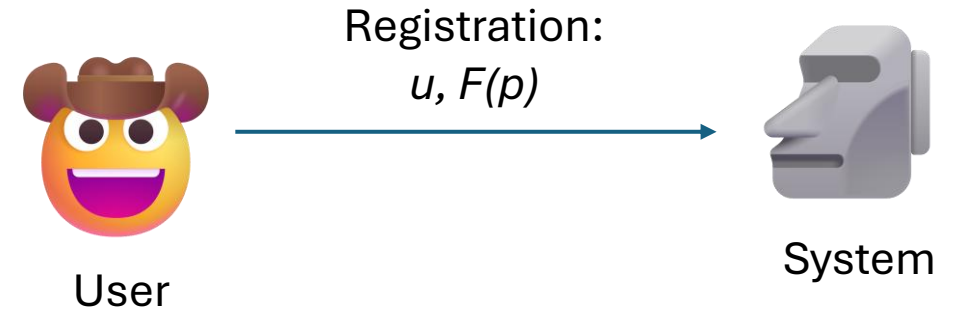
$(p, q)$ : Passphrase at Reg./Auth. time

$(G, F)$ : mapping of phrase to token at Reg./Auth. Time

$C$ : function to evaluate correctness

## What is the correctness requirement?

- $(p = q) \rightarrow C( F(p), G(q) ) = \text{True}$
- $(p \neq q) \rightarrow C( F(p), G(q) ) = \text{False}$



# Password Protocols

## Formal modeling of passwords:

$u$ : User identifier

$(p, q)$ : Passphrase at Reg./Auth. time

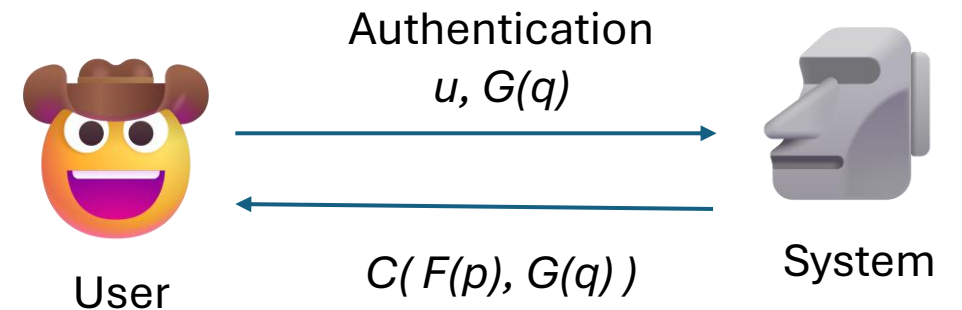
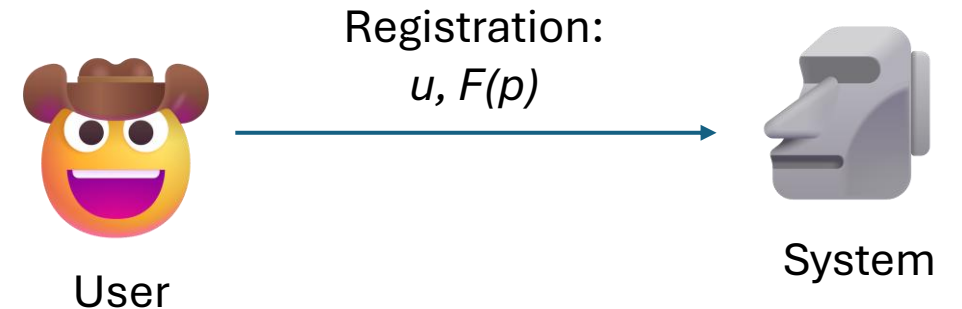
$(G, F)$ : mapping of phrase to token at Reg./Auth. Time

$C$ : function to evaluate correctness

## What is the correctness requirement?

- $(p = q) \rightarrow C( F(p), G(q) ) = \text{True}$
- $(p \neq q) \rightarrow C( F(p), G(q) ) = \text{False}$

Let's design a simple protocol to satisfy this requirement...



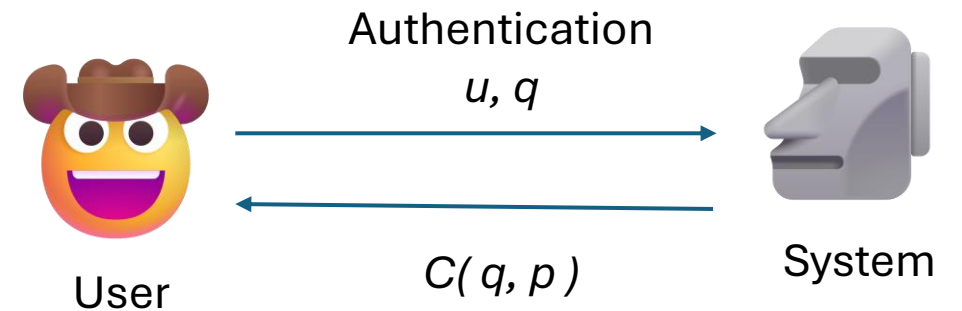
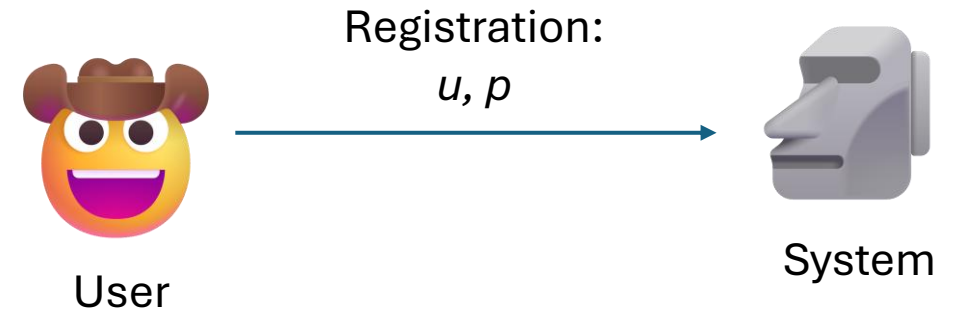
# Password Protocols

## Simple protocol --- Done!

$F(p) \rightarrow p$

$G(q) \rightarrow q$

$C(x, y) \rightarrow x \stackrel{?}{=} y$



# Password Protocols

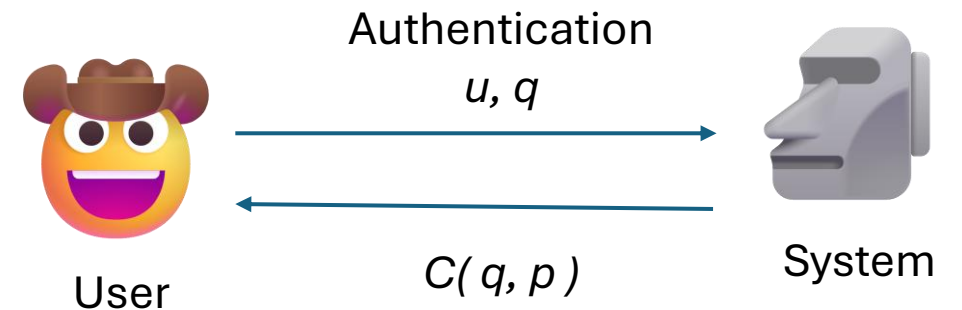
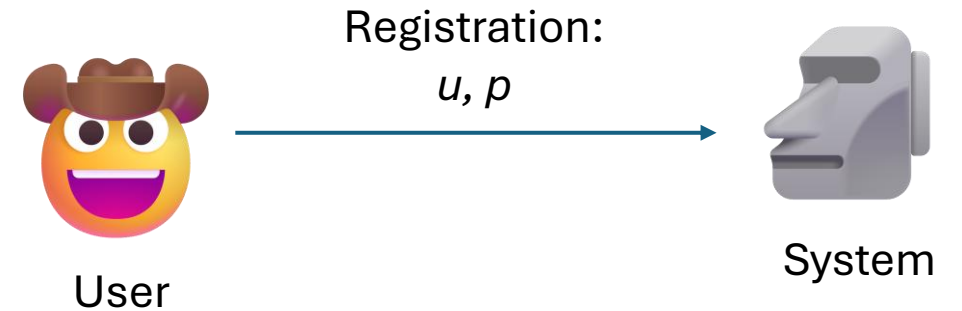
## Simple protocol --- Done!

$F(p) \rightarrow p$

$G(q) \rightarrow q$

$C(x, y) \rightarrow x \stackrel{?}{=} y$

## What's wrong with this scheme?



# Password Protocols

## Simple protocol --- Done!

$F(p) \rightarrow p$

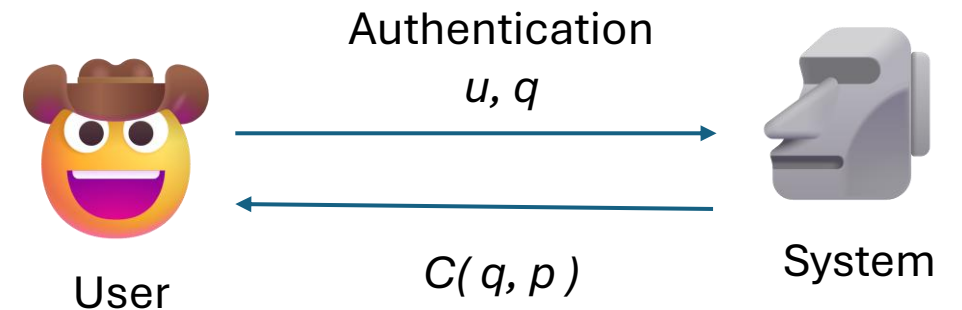
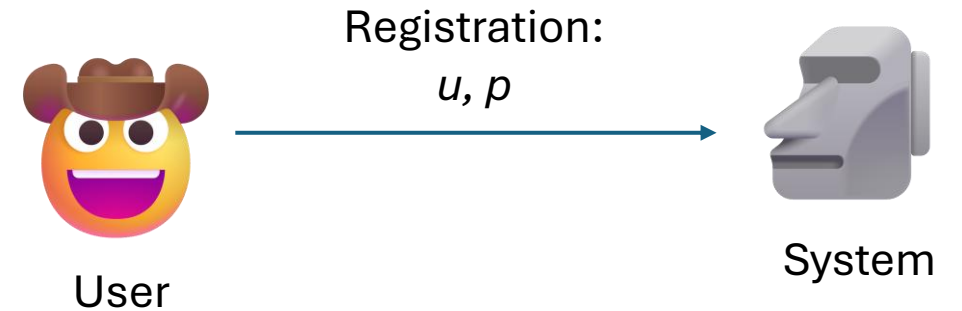
$G(q) \rightarrow q$

$C(x, y) \rightarrow x \stackrel{?}{=} y$

## What's wrong with this scheme?

Stores passwords in plaintext

- System might have another vulnerability
- Leaked plaintext password breaks the protocol





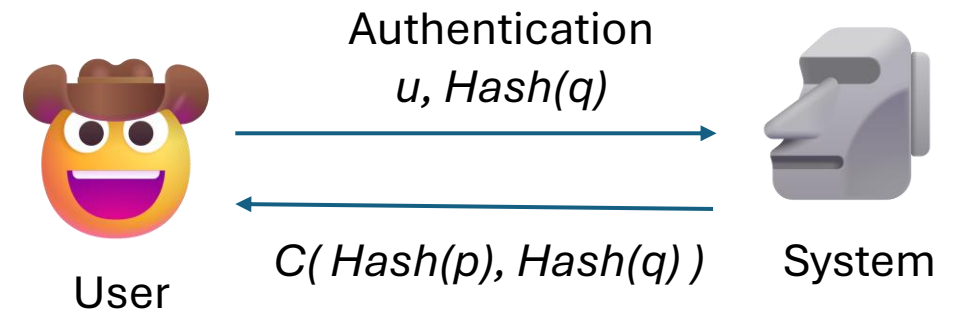
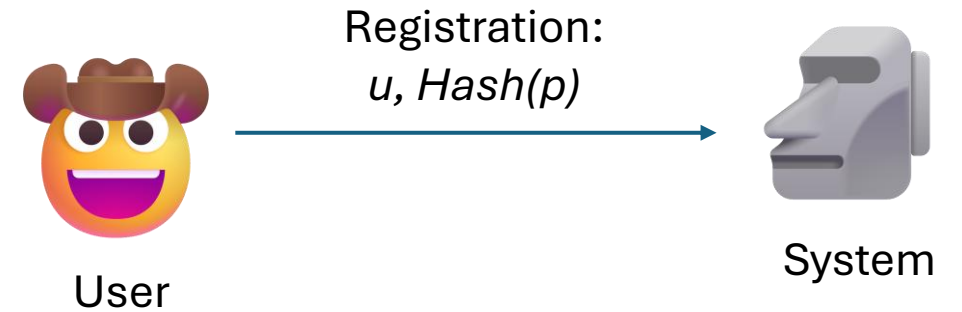
# Password Protocols

## Alternative approach... use a hash

$F(p) \rightarrow \text{Hash}(p)$

$G(q) \rightarrow \text{Hash}(q)$

$C(x, y) \rightarrow x \stackrel{?}{=} y$



# Password Protocols

## Alternative approach... use a hash

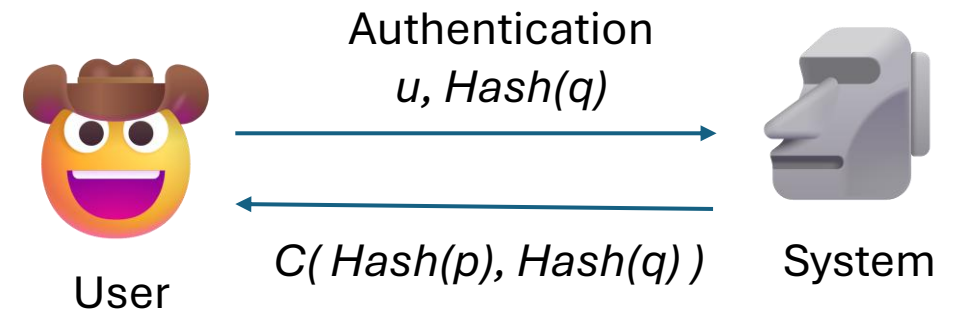
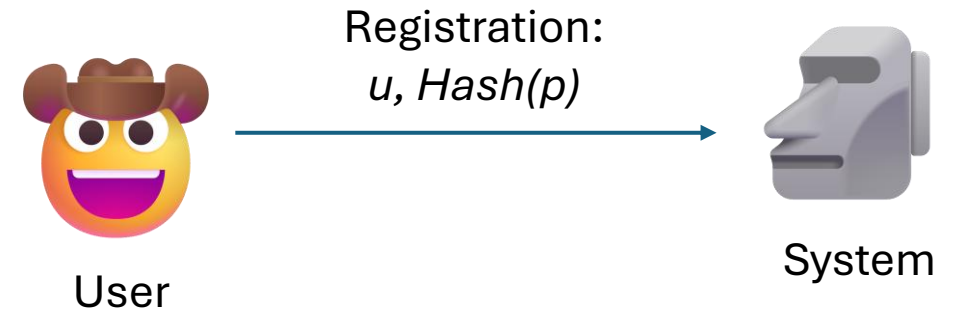
$F(p) \rightarrow \text{Hash}(p)$

$G(q) \rightarrow \text{Hash}(q)$

$C(x, y) \rightarrow x \stackrel{?}{=} y$

## What is a hash function?

- Takes arbitrary length string  $x$
- Computes fixed-length digest:  $y = \text{Hash}(x)$
- Deterministic:  $H(x)$  always produces a single  $y$
- Examples: MD5, SHA1, SHA2, SHA3



# Password Protocols

## Alternative approach... use a hash

$F(p) \rightarrow \text{Hash}(p)$

$G(q) \rightarrow \text{Hash}(q)$

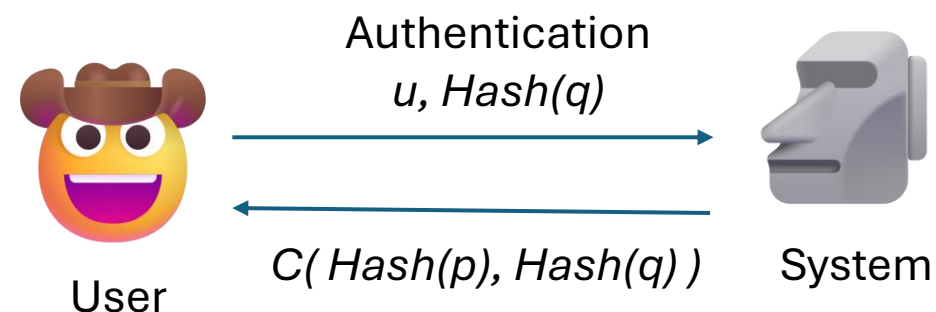
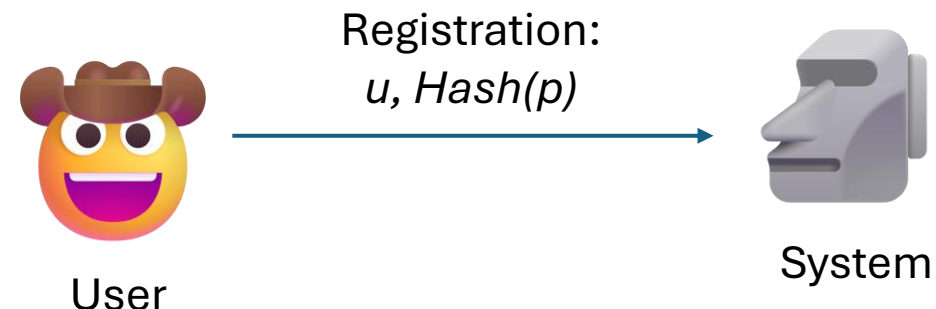
$C(x, y) \rightarrow x \stackrel{?}{=} y$

## What is a hash function?

- Takes arbitrary length string  $x$
- Computes fixed-length digest:  $y = \text{Hash}(x)$
- Deterministic:  $H(x)$  always produces a single  $y$
- Examples: MD5, SHA1, SHA2, SHA3

A hash function is cryptographically secure if it has...

- Pre-image resistance:
  - Given  $y$ , it is hard to find  $x$  s.t.  $\text{Hash}(x) = y$
- Second preimage resistance:
  - Given  $x$ , it is hard to find  $x' \neq x$  and  $h(x) = h(x')$
- Collision-resistance
  - It is hard to find two values  $(x, x')$  such that  $h(x) = h(x')$



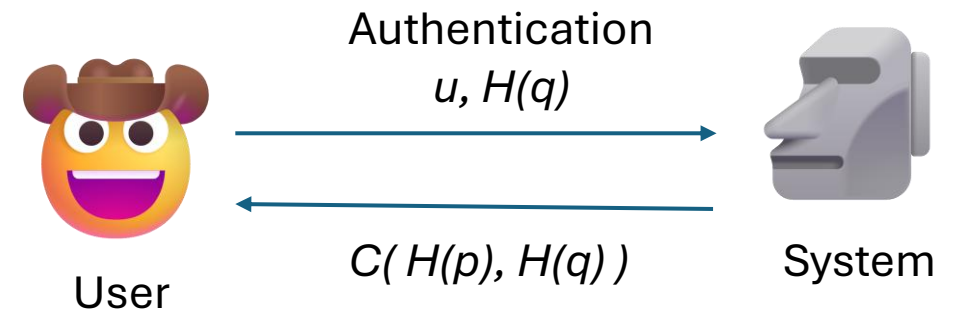
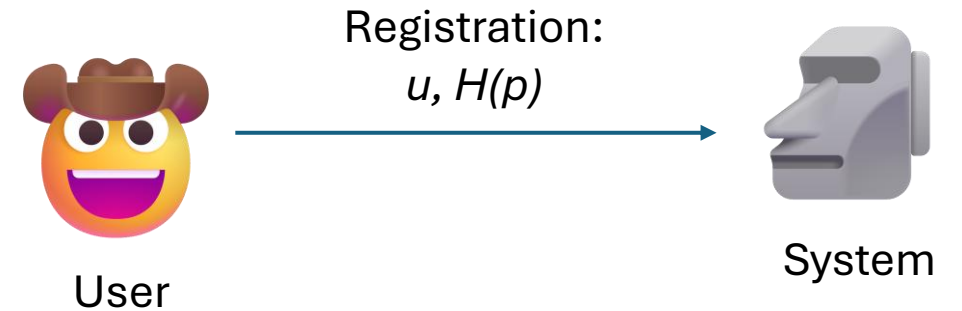
# Password Protocols

**Let's use a cryptographic hash function  $H(\cdot)$**

Does this satisfy the correctness requirement?

The first half:

$$p = q \rightarrow C( H(p), H(q) ) = ( H(p) \stackrel{?}{=} H(q) ) = \text{True}$$



# Password Protocols

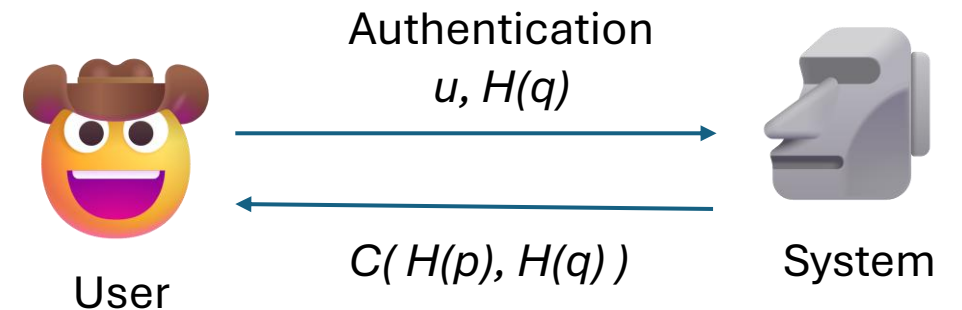
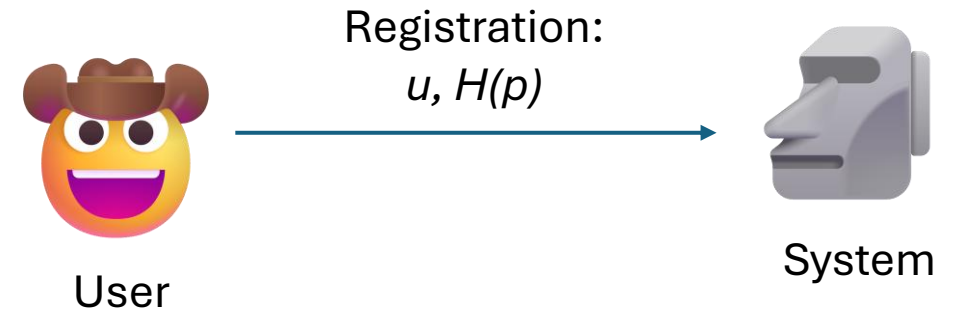
**Let's use a cryptographic hash function  $H(\cdot)$**

Does this satisfy the correctness requirement?

The first half:

$p = q \rightarrow C( H(p), H(q) ) = ( H(p) \text{ ?} = H(q) ) = \text{True}$

- Yes:  $H(p)$  is deterministic



# Password Protocols

## Let's use a cryptographic hash function $H(.)$

Does this satisfy the correctness requirement?

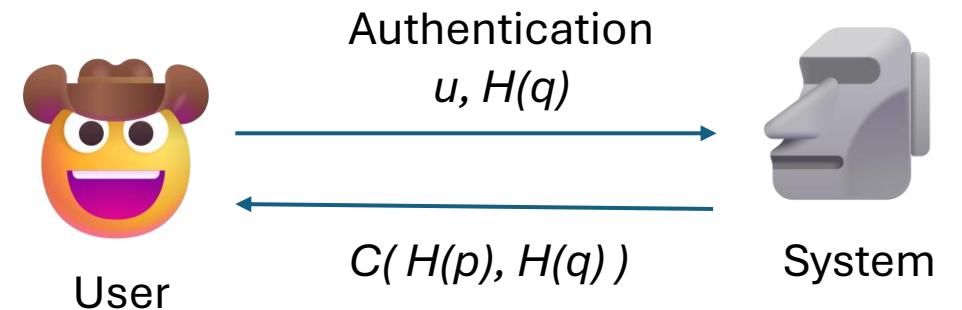
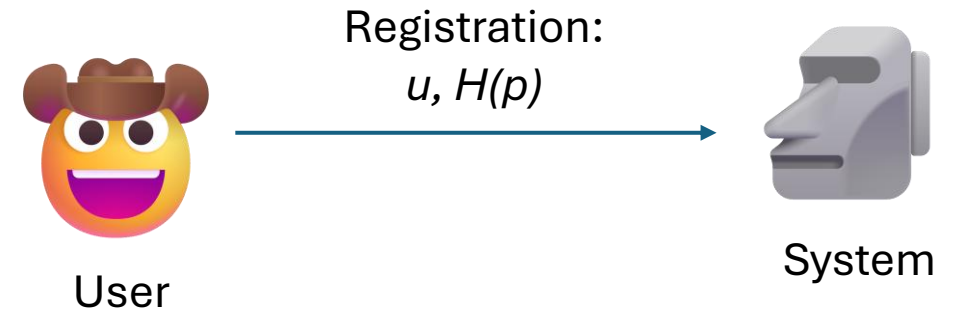
The first half:

$$p = q \rightarrow C( H(p), H(q) ) = ( H(p) \text{ ?} = H(q) ) = \text{True}$$

- Yes:  $H(p)$  is deterministic

The second half:

$$p \neq q \rightarrow C( H(p), H(q) ) = ( H(p) \text{ ?} = H(q) ) = \text{False}$$



# Password Protocols

## Let's use a cryptographic hash function $H(.)$

Does this satisfy the correctness requirement?

The first half:

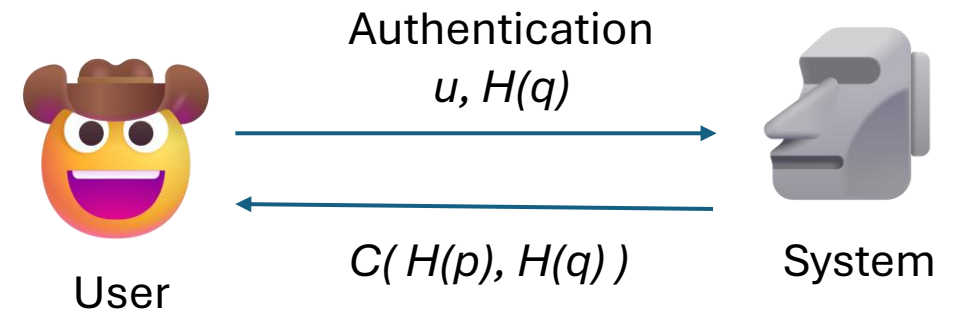
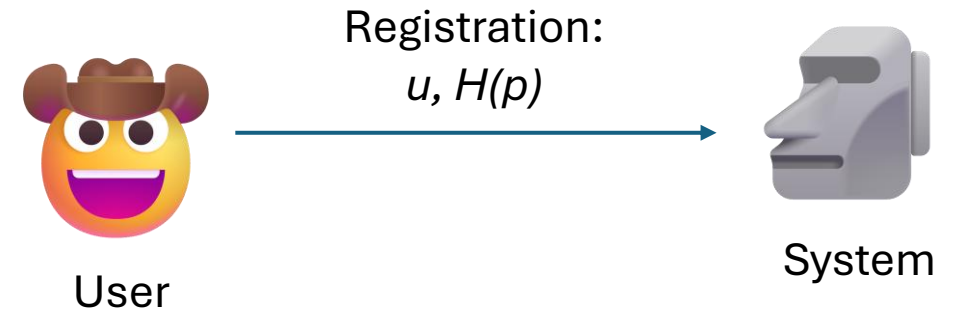
$p = q \rightarrow C( H(p), H(q) ) = ( H(p) ?= H(q) ) = \text{True}$

- Yes:  $H(p)$  is deterministic

The second half:

$p \neq q \rightarrow C( H(p), H(q) ) = ( H(p) ?= H(q) ) = \text{False}$

- $H()$  is collision resistant with high probability...



# Password Protocols

## Let's use a cryptographic hash function $H(.)$

Does this satisfy the correctness requirement?

The first half:

$$p = q \rightarrow C( H(p), H(q) ) = ( H(p) \text{ ?} = H(q) ) = \text{True}$$

- Yes:  $H(p)$  is deterministic

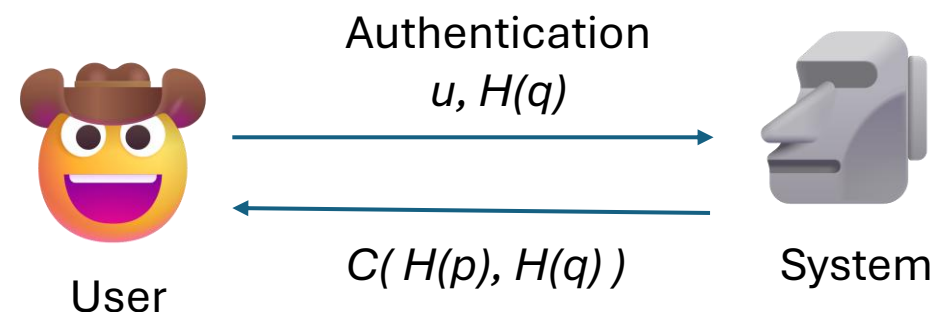
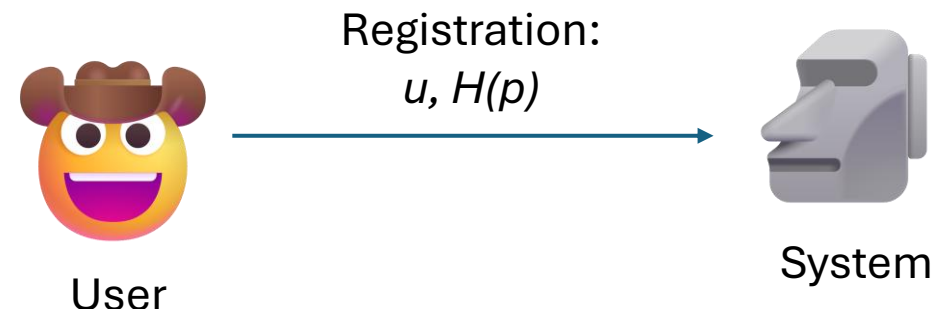
The second half:

$$p \neq q \rightarrow C( H(p), H(q) ) = ( H(p) \text{ ?} = H(q) ) = \text{False}$$

- $H()$  is collision resistant with high probability...

$$p \neq q \rightarrow \mathbf{Pr}[C( H(p), H(q) ) = ( H(p) \text{ ?} = H(q) ) = \text{True}] < e$$

- Correct with a minimal error ( $e$ )

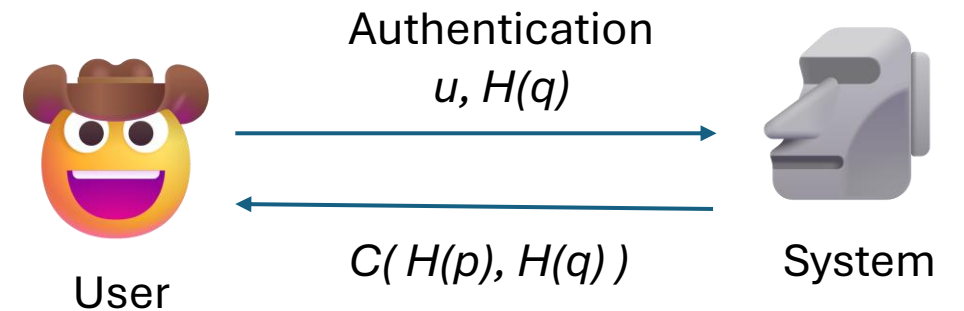
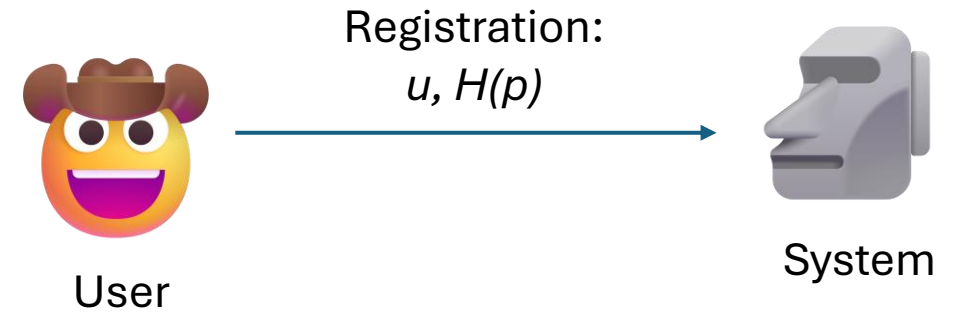




# Password Protocols

## Weaknesses of hash-based protocol:

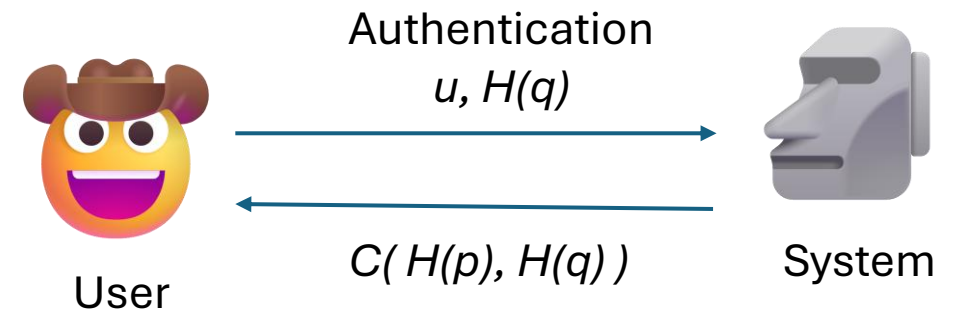
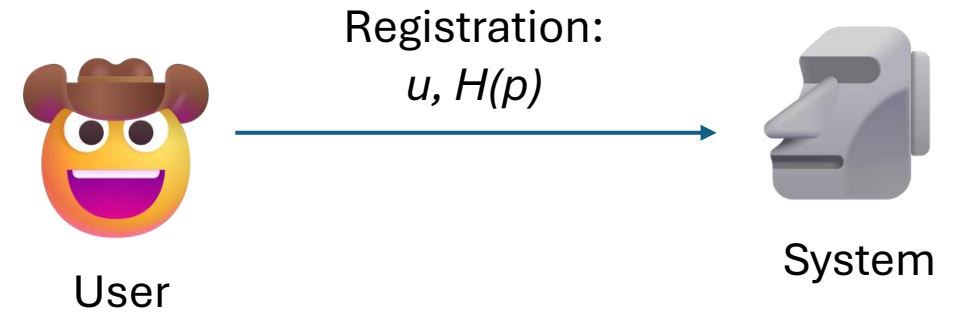
- Possible collisions with error (e)
- Anything else?



# Password Protocols

## Weaknesses of hash-based protocol:

- Possible collisions with error (e)
- Anything else?
  - Same password, same digest

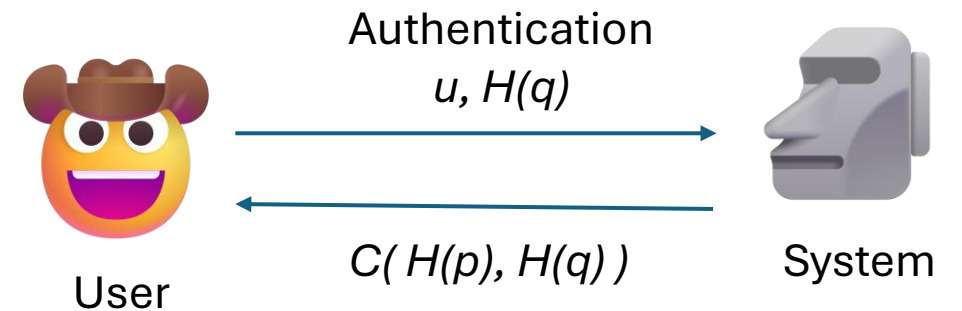
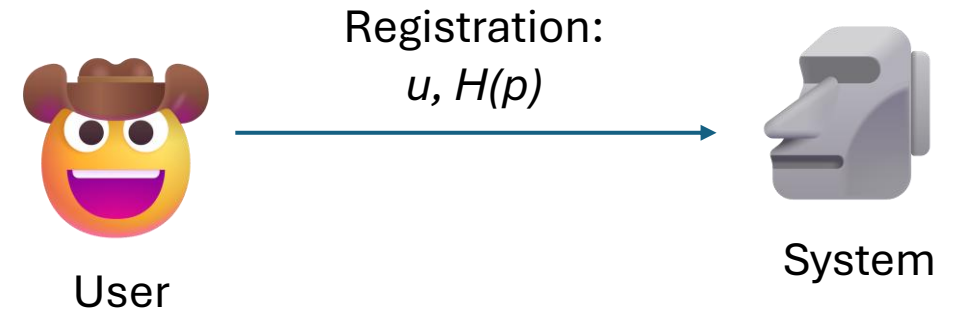


# Password Protocols

## Weaknesses of hash-based protocol:

- Possible collisions with error (e)
- Anything else?
  - Same password, same digest

**How to mitigate?** → Salted passwords



# Password Protocols

## Salted password protocol (v1):

$F(p) \rightarrow H(p \mid s)$

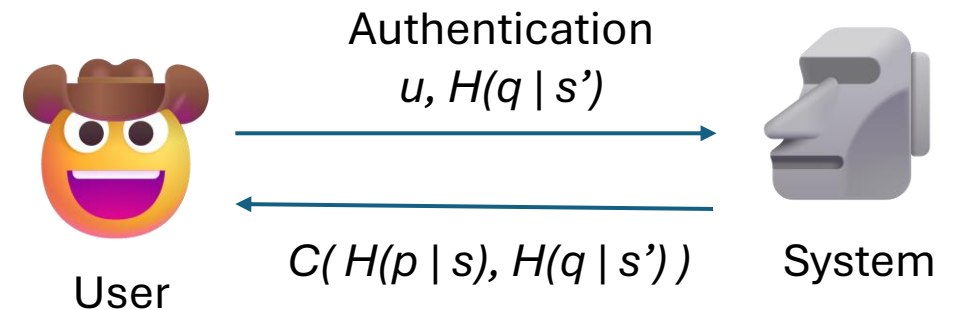
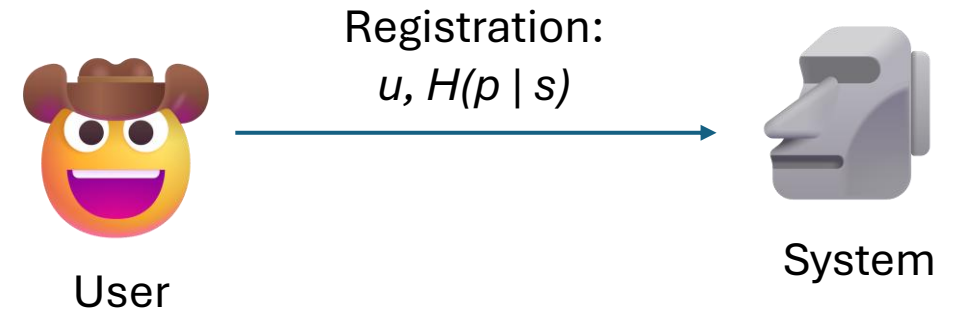
$G(q) \rightarrow H(q \mid s)$

$C(x,y) \rightarrow x \stackrel{?}{=} y$

The digest of the password is not stored in plaintext

### Challenge:

- User is responsible for storing the salt in use
- Can still be an inconvenience



# Password Protocols

## Salted password protocol (v2):

- User sends (s) during registration
- System stores the salt

## Same functions:

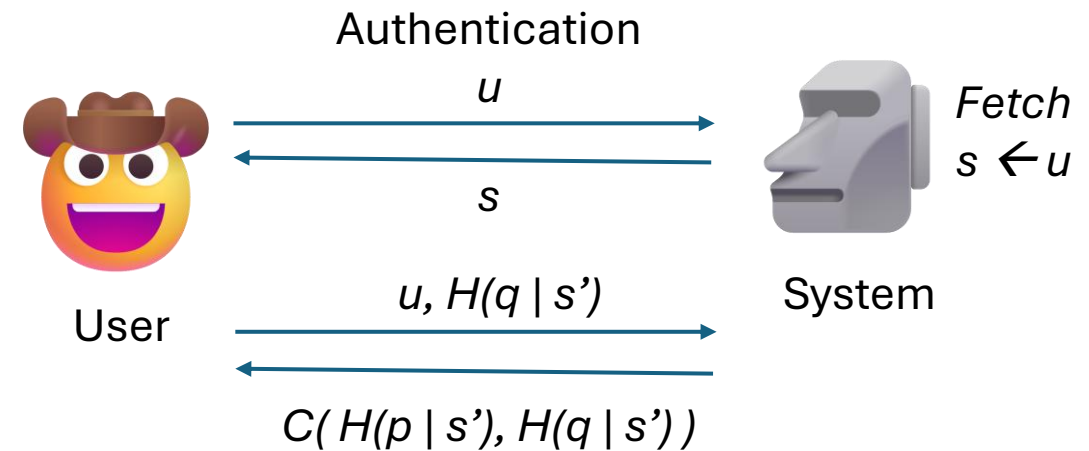
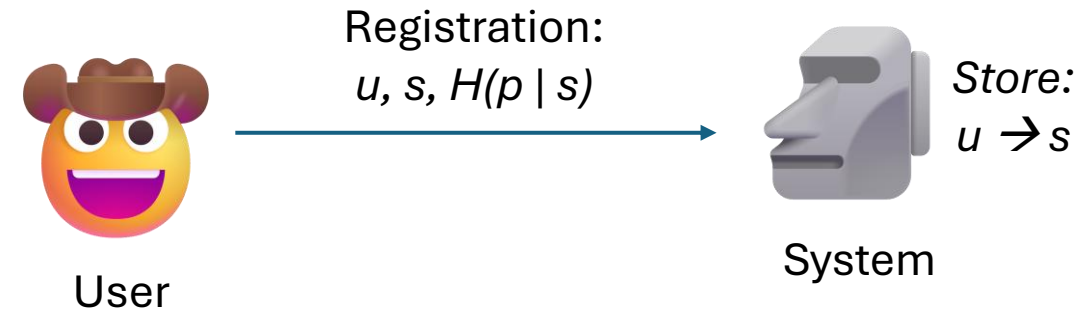
$F(p) \rightarrow H(p \mid s)$

$G(q) \rightarrow H(q \mid s)$

$C(x,y) \rightarrow x \stackrel{?}{=} y$

## Downside:

- Requires an extra roundtrip into the authentication
- Potentially enables user-probing attack
  - Request salts for user id's
  - If System returns back a salt, can use to guess p again



# Password Protocols

## Salted password protocol (v3):

- Salt is assigned by the System and is oblivious to the user

## Salt applied at comparison:

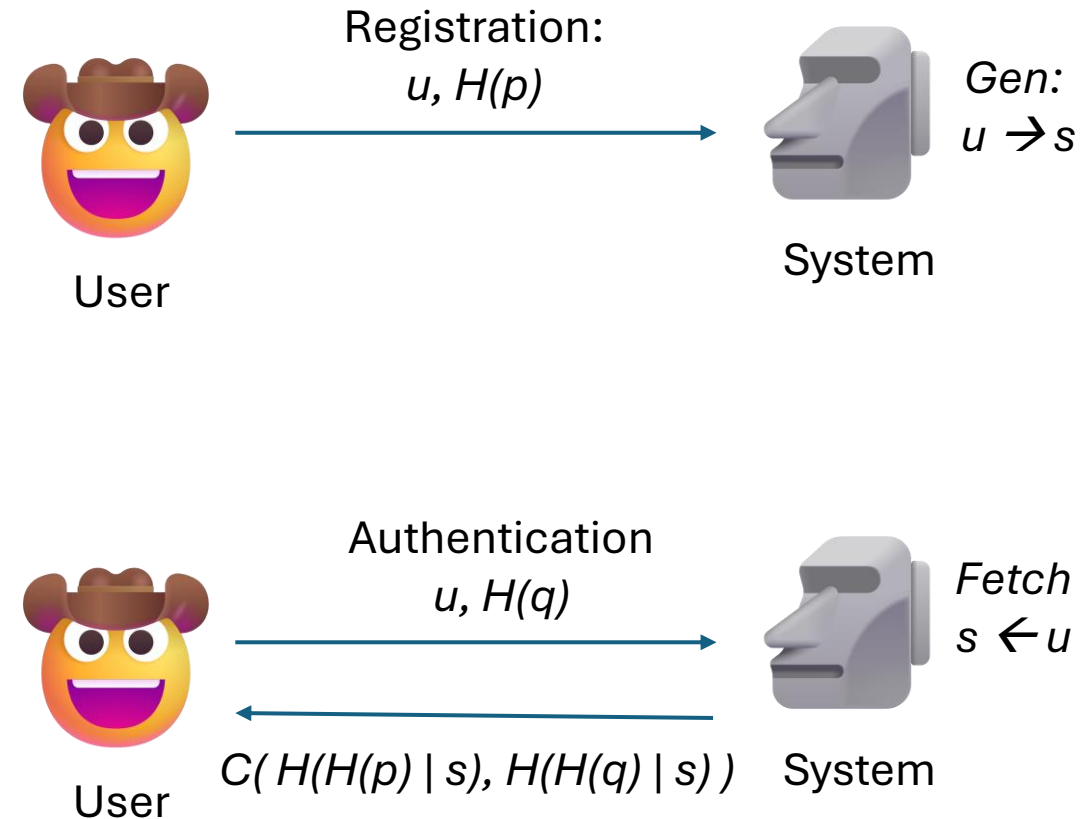
$F(p) \rightarrow H(p)$

$G(q) \rightarrow H(q)$

$C(x, y) \rightarrow H(x \parallel s) \stackrel{?}{=} H(y \parallel s)$

## Result:

- Stronger: adversary must determine both  $p$  and  $s$
- But still (potentially) possible to brute force
- Example:
  - $p$  is not a strong password (e.g., previously leaked, in dictionary)
  - Reduced to brute-forcing  $s$



# Password Protocols

## **The problem: inputs are either**

- Sent over the network
- Potentially leaked due to use in other systems
- Possible to brute force

## **So we need something that is...**

- A unique secret per system
- Challenging to brute force

## **PKI saves the day!**

# Password Protocols

## PKI password protocol:

- At registration, store user's verification key

## Define functions based on PKI:

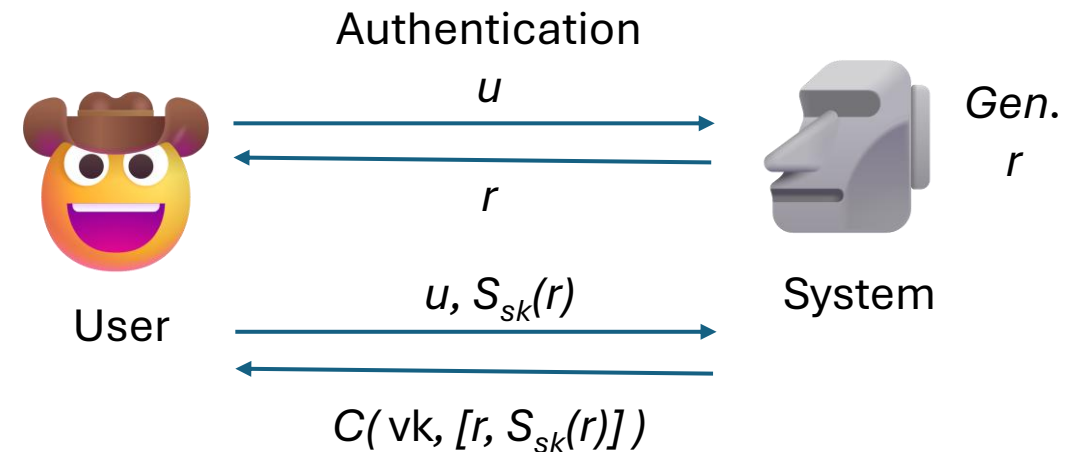
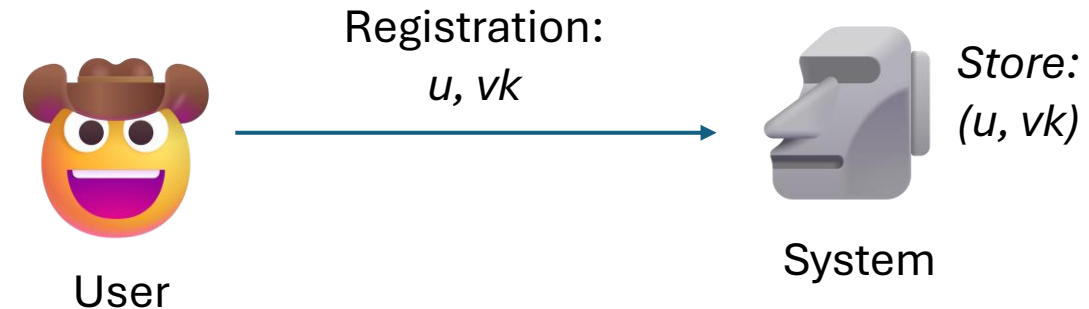
$F(u) \rightarrow vk$  : map user ( $u$ ) to verification/public key ( $vk$ )

$G(r) \rightarrow S_{sk}(r)$  : signature over  $r$  using secret/private key ( $sk$ )

$C(vk, r, S_{sk}(r)) \rightarrow V_{vk}(r, S_{sk}(r))$  : signature verification

## Result:

- $r \rightarrow$  nonce
- Relying on signature scheme's hardness assumption
- Requires/assumes secret key management by User
- Example: passkey, passwordless ssh





# Alternative methods

Many alternatives to text-based passwords

- Unlock patterns
- Geographical based passwords

## Biometrics

- Fingerprints, handwriting, typing patterns
- If observed trait is **sufficiently close** to the previously stored trait, accept the user
- Since observed fingerprint will never be completely identical to a previously stored fingerprint of the same user

Other challenges with biometrics:

- Privacy/secretcy concerns
- Accuracy
- Legal/ethical concerns

# Attestation

From authentication to attestation....

## **Recall: Authentication assumes**

- Devices (aka Alice and Bob) are honest
- The Adversary controls the network
  - Messages can be intercepted, reads, modified, or replayed by the adversary

# Attestation

From authentication to attestation....

## Recall: Authentication assumes

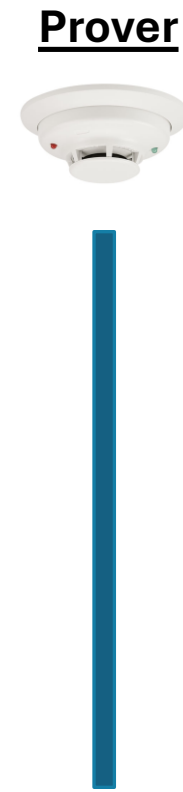
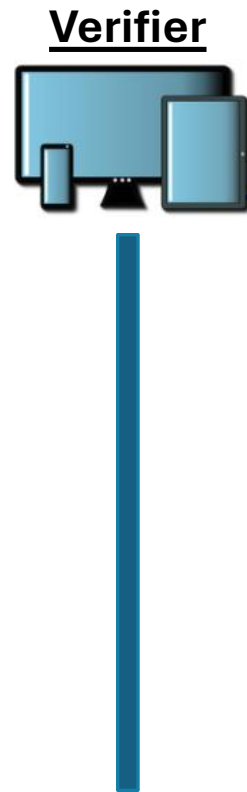
- Devices (aka Alice and Bob) are honest
- The Adversary controls the network
  - Messages can be intercepted, reads, modified, or replayed by the adversary

## What about a slightly stronger adversary?

- Both the network and a prover's software might be under control by an Adversary
  - Less trust → stronger Adversary

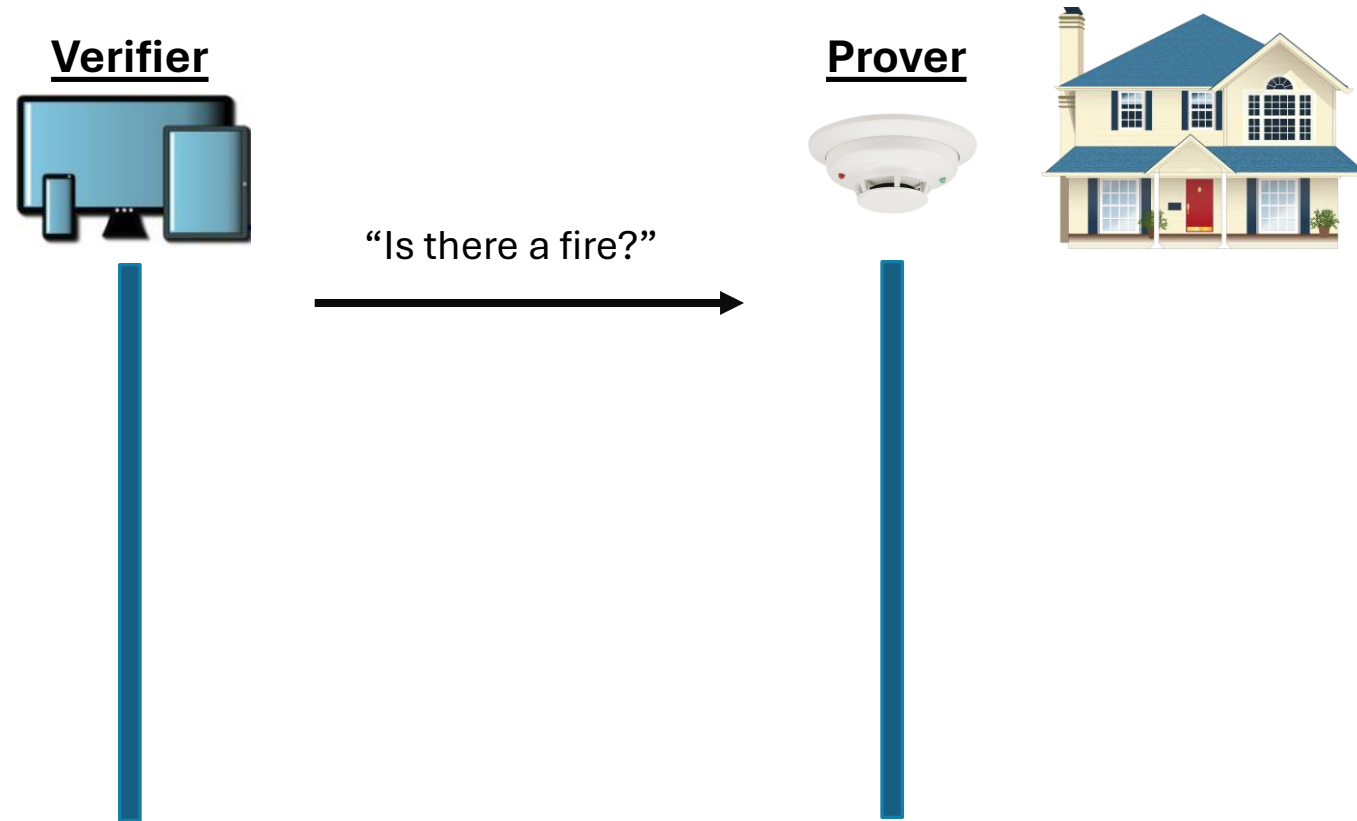
# Attestation

Example use case: remotely-operated sensor



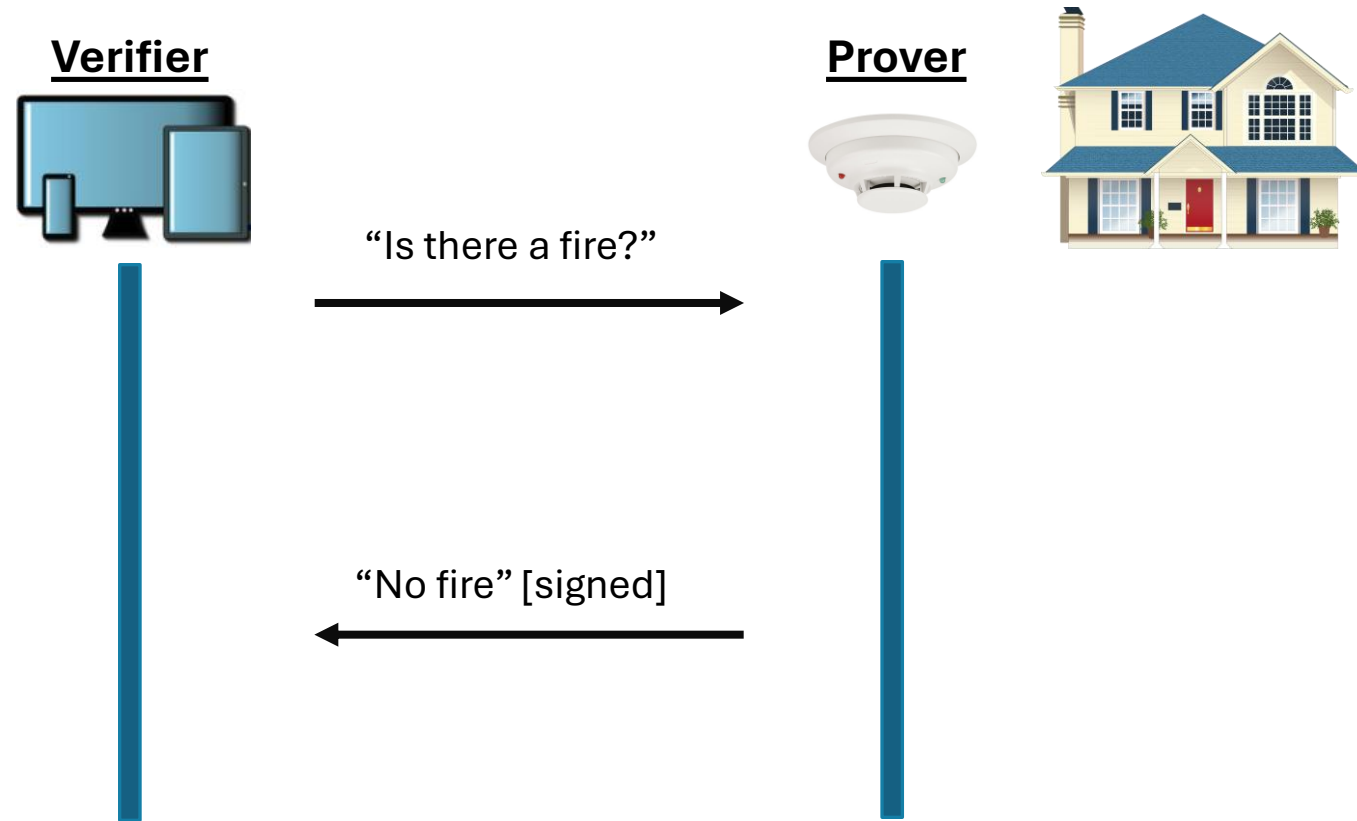
# Attestation

Example use case: remotely-operated sensor



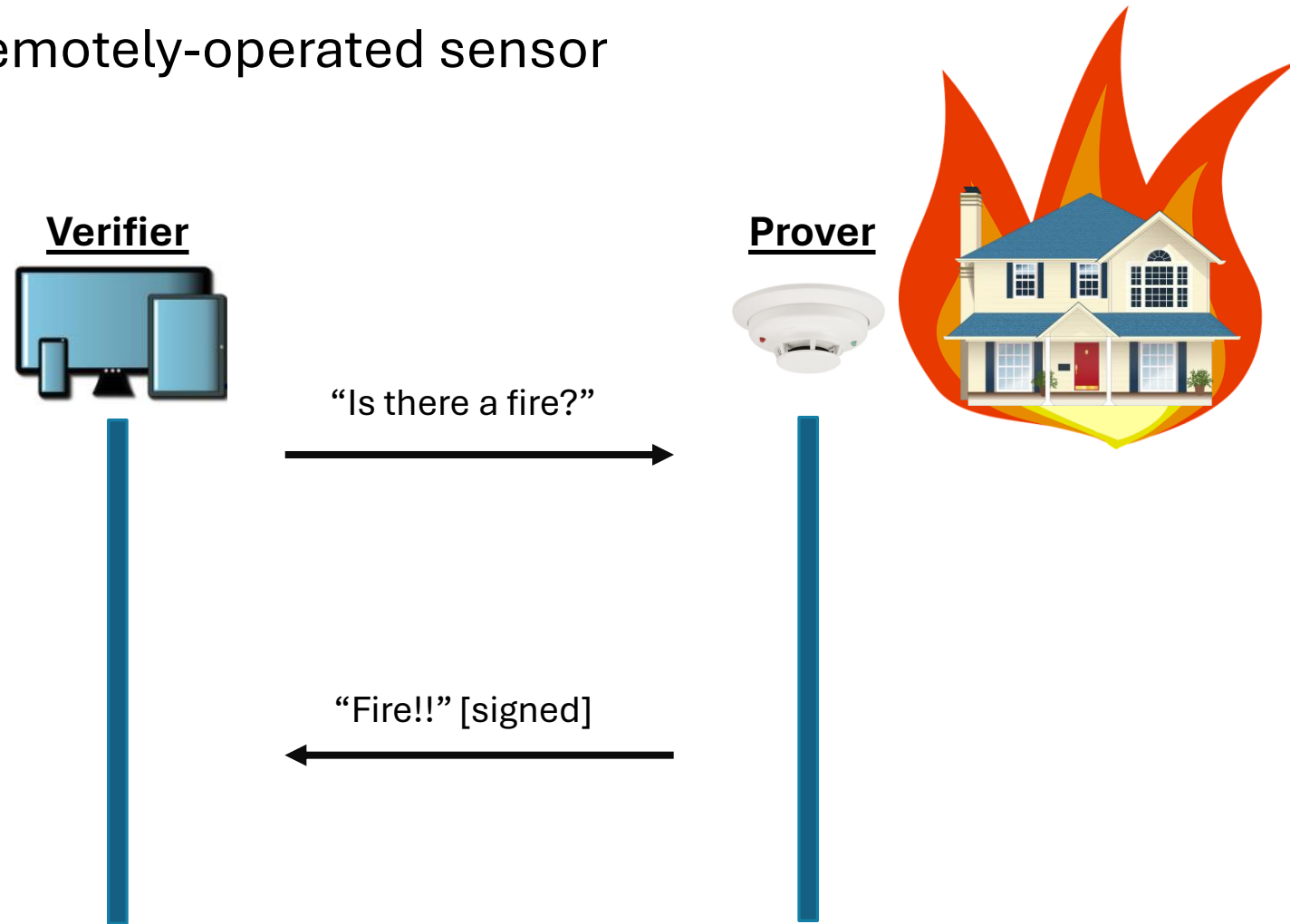
# Attestation

Example use case: remotely-operated sensor



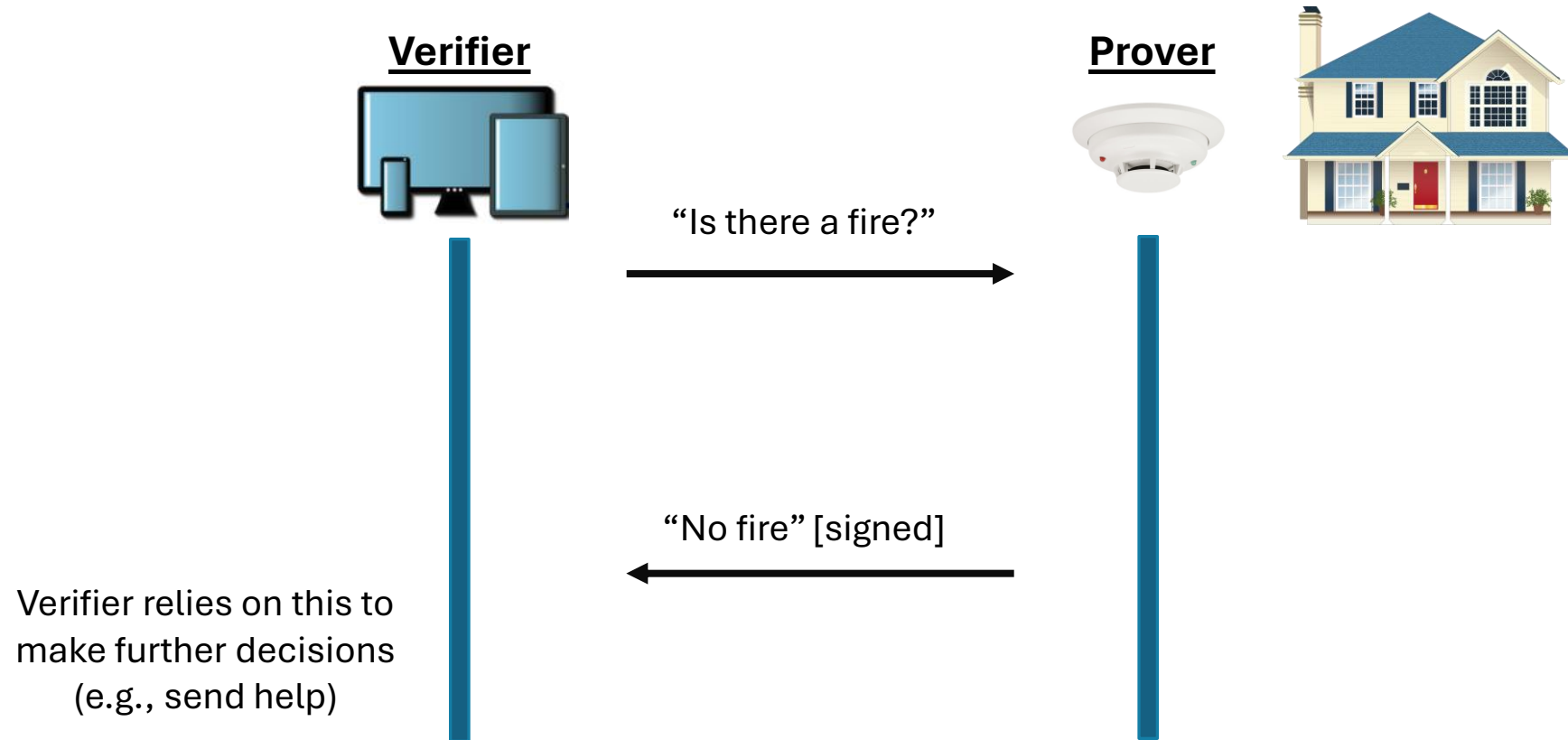
# Attestation

Example use case: remotely-operated sensor



# Attestation

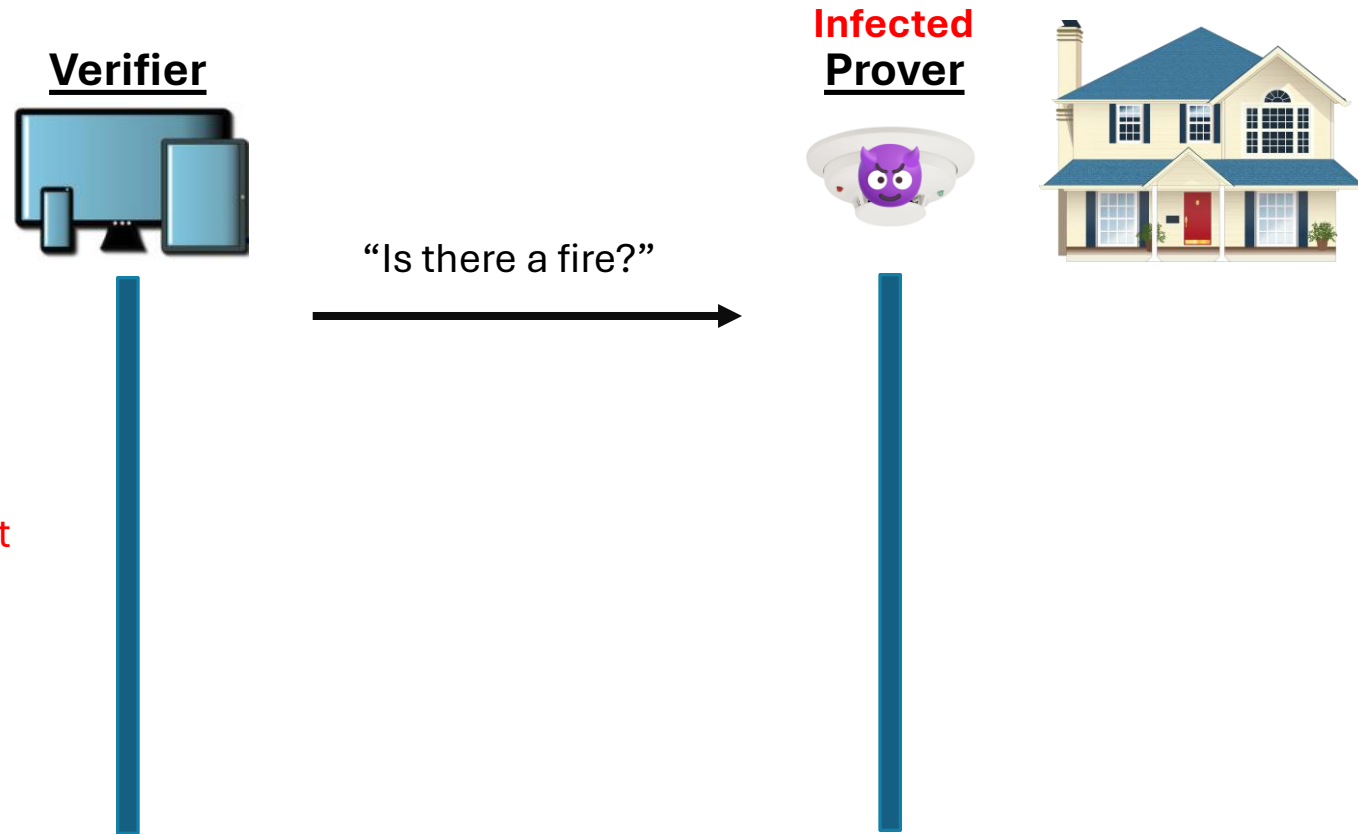
Example use case: remotely-operated sensor





# Attestation

Example use case: remotely-operated sensor

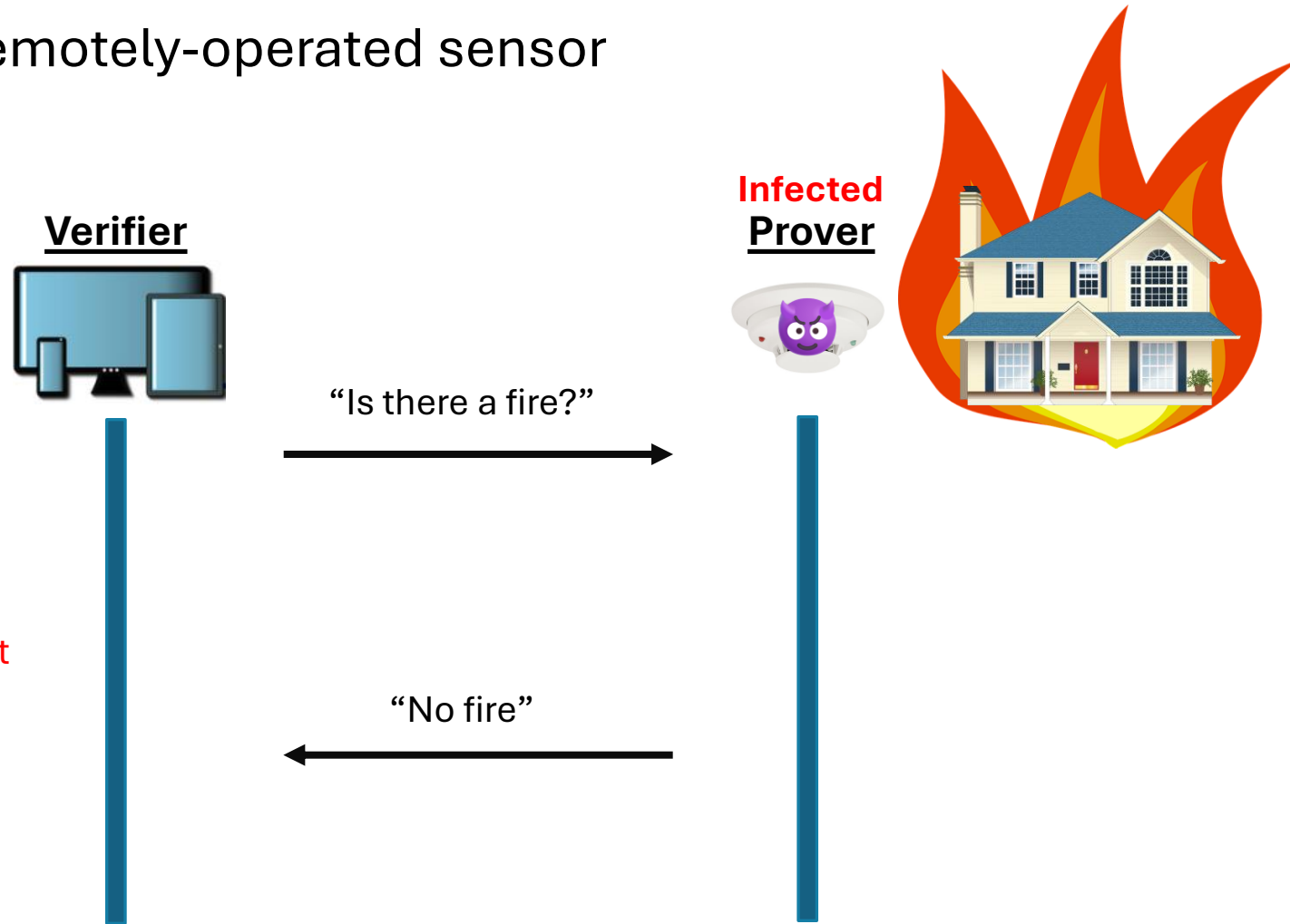


**Problem:**

- Compromised software might spoof results

# Attestation

Example use case: remotely-operated sensor

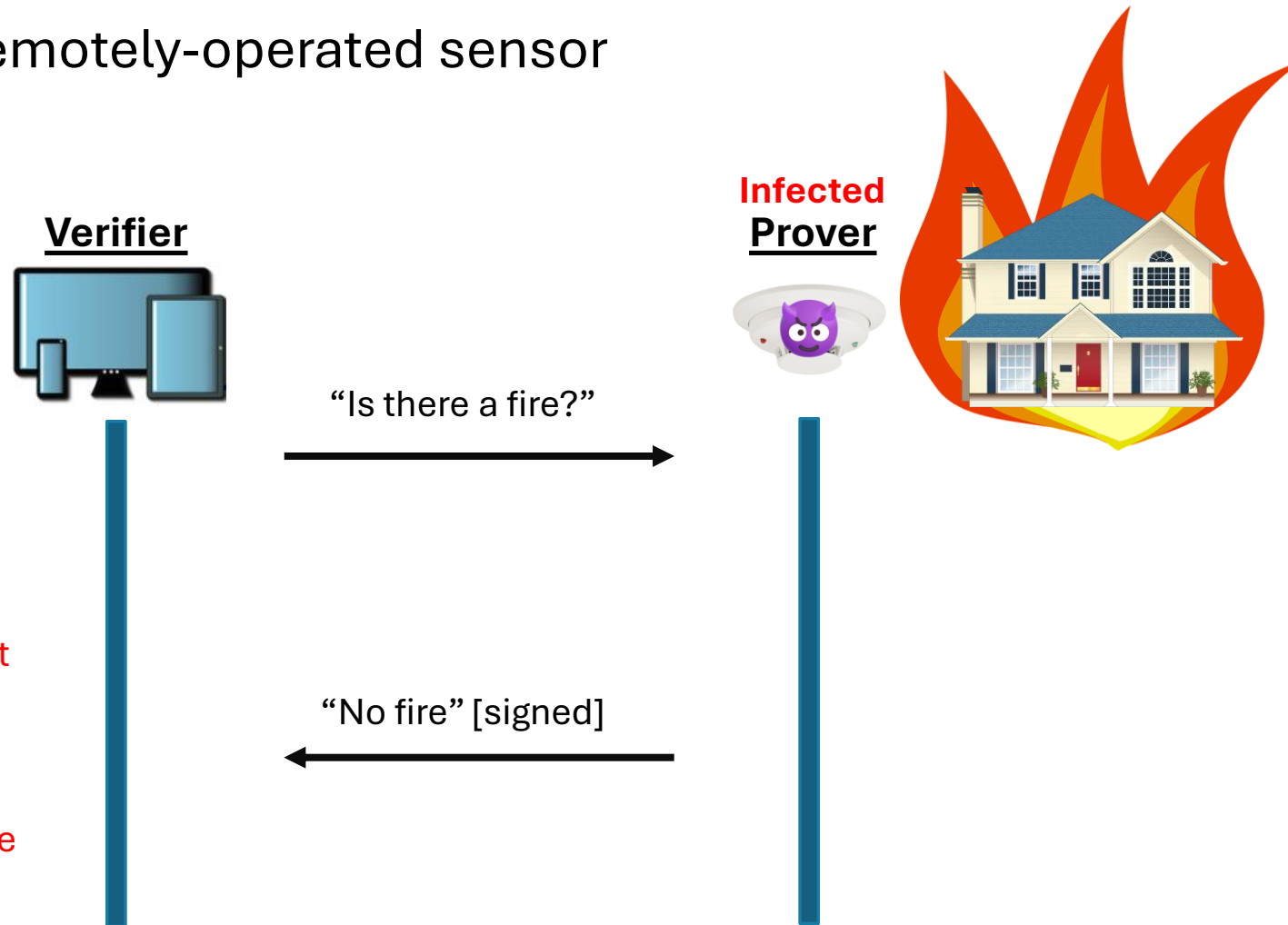


**Problem:**

- Compromised software might spoof results

# Attestation

Example use case: remotely-operated sensor



**Problem:**

- Compromised software might spoof results

**Authentication isn't enough:**

- Tell's Verifier that the message came from the Prover
- Doesn't tell Verifier if it is trustworthy

# Attestation

## What is Attestation?

**Definition:** *a protocol/method in which a Prover authenticates its hardware and software configuration to a remote Verifier with the goal of enabling a the Verifier to determine the level of trust in the integrity of Prover.*

**Remote Attestation** → Prover and Verifier are connected over the network

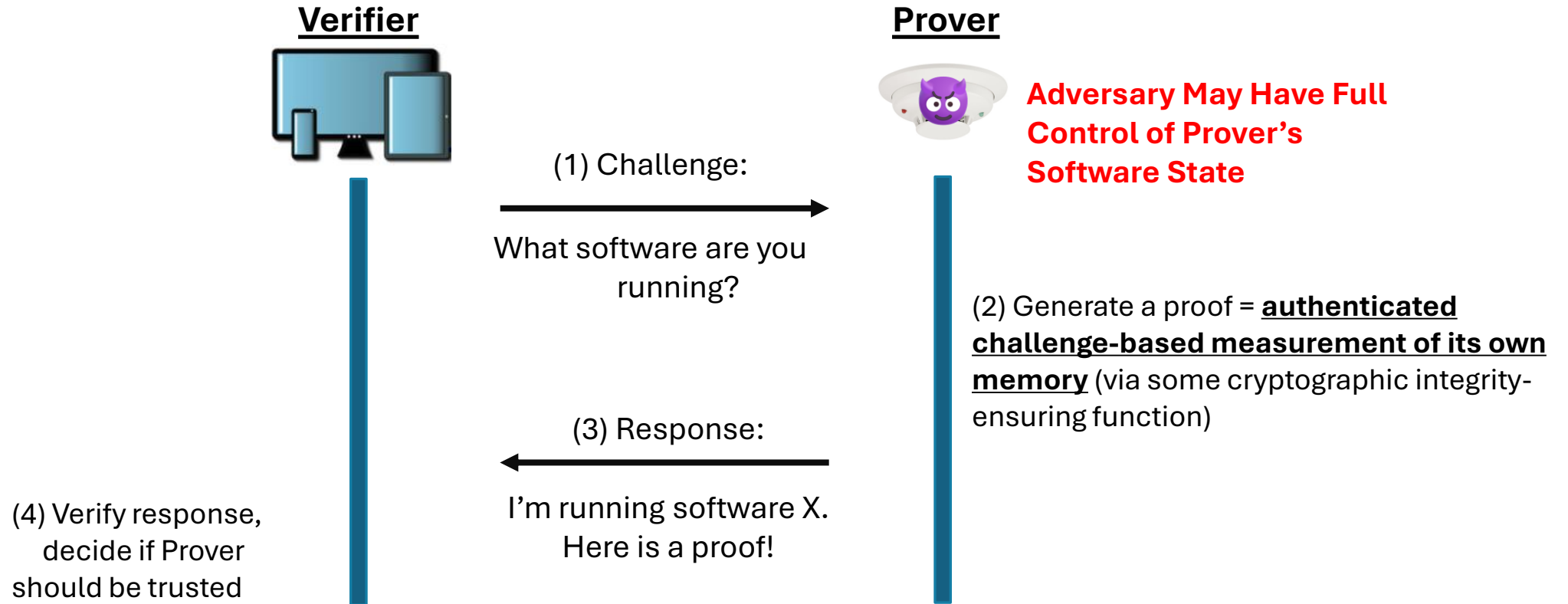
Slightly different than Authentication, but sounds similar:

- Unforgeable evidence
- Verifier/Challenger & Prover

**Key difference:** Prover's software is untrusted

# Attestation

Example use case: remotely-operated sensor



Let's look into two instantiations...

# Remote Attestation Protocols

**But first, clarify the assumptions:**

**Adversary:**

- Has control over the network (same as for Authentication Protocols)
- Has control over any software in Prover that is not explicitly protected
  - Read, write, execute

# Remote Attestation Protocols

**But first, clarify the assumptions:**

## **Adversary:**

- Has control over the network (same as for Authentication Protocols)
- Has control over any software in Prover that is not explicitly protected
  - Read, write, execute

**For now, make one more assumption...**

- Prover has a **Root of Trust (RoT)** that can
  - Securely store keys
  - Can compute cryptographic functions without leaking keys
  - Guarantees are upheld even when all software has been compromised/modified
- How? Coming up in later lectures....

# Remote Attestation Protocols

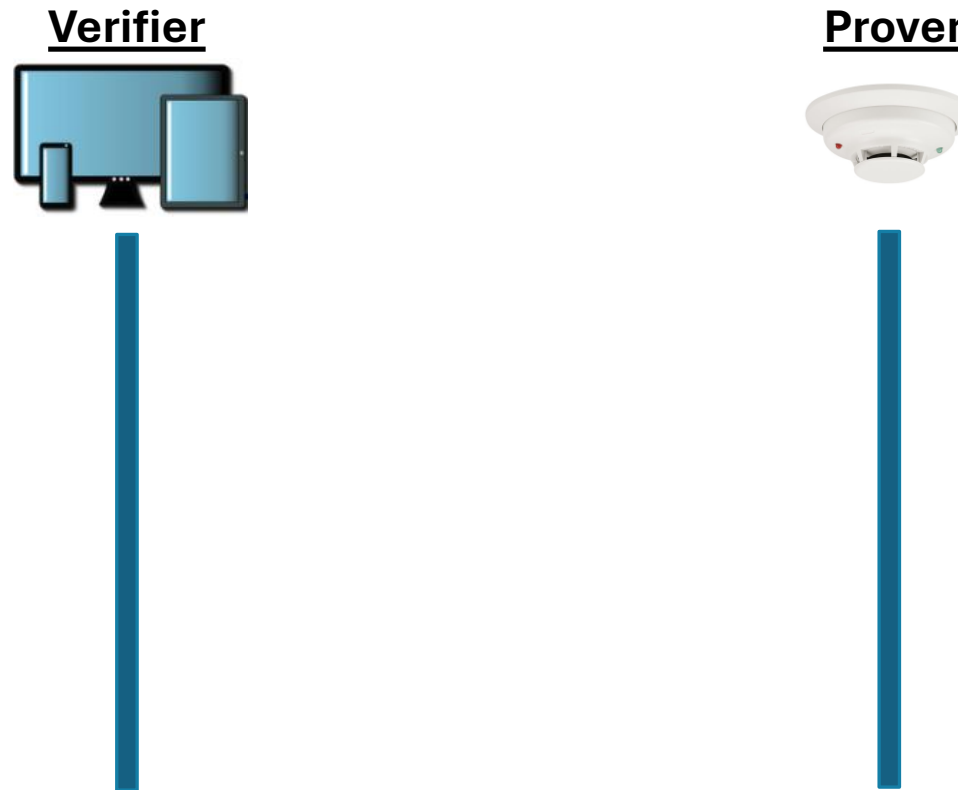
## Two Remote Attestation Protocols

- Using symmetric key:
  - Verifier and Prover's RoT share a secret key
- Using public key:
  - Verifier knows a public key corresponding to Prover's RoT's secret key



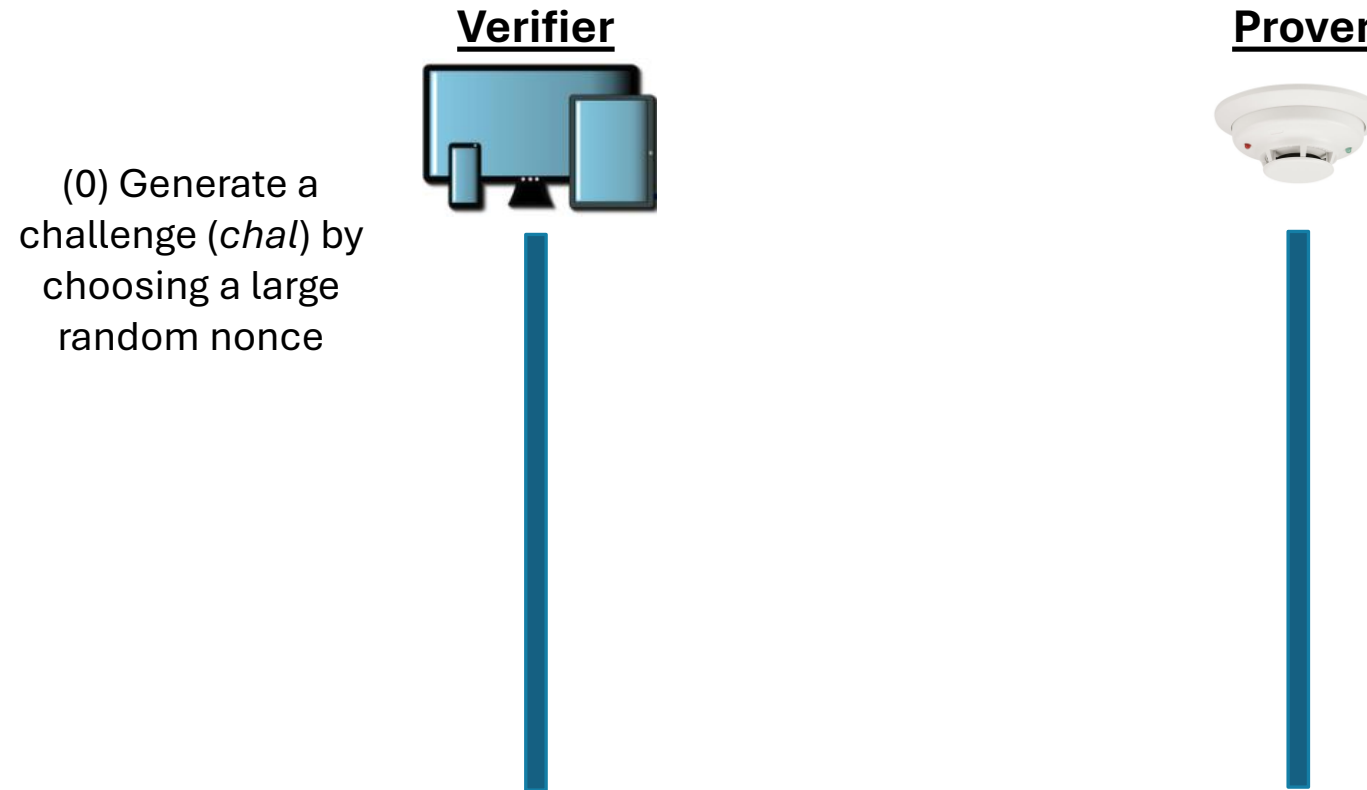
# Attestation

**Protocol 1:** Verifier and Prover's RoT share symmetric key ( $K$ )



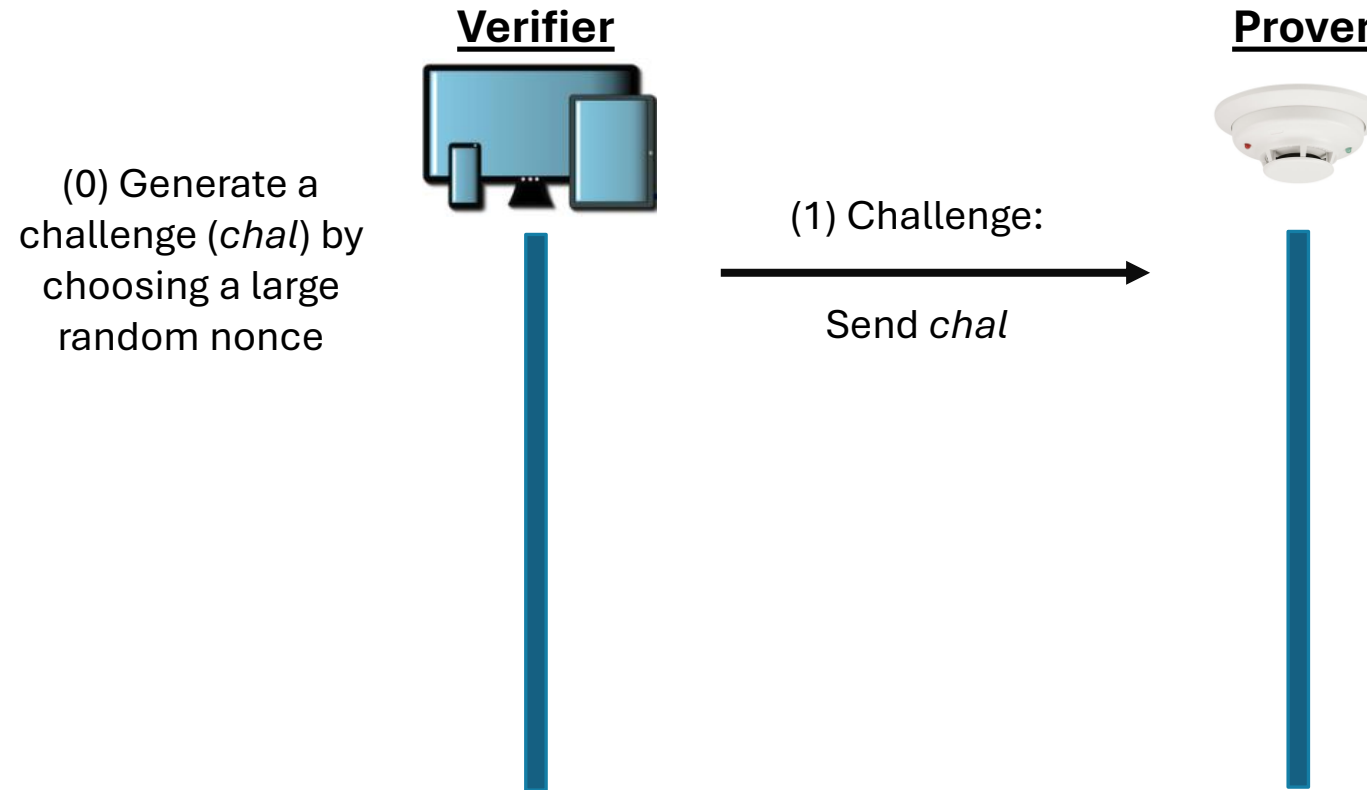
# Attestation

**Protocol 1:** Verifier and Prover's RoT share symmetric key ( $K$ )



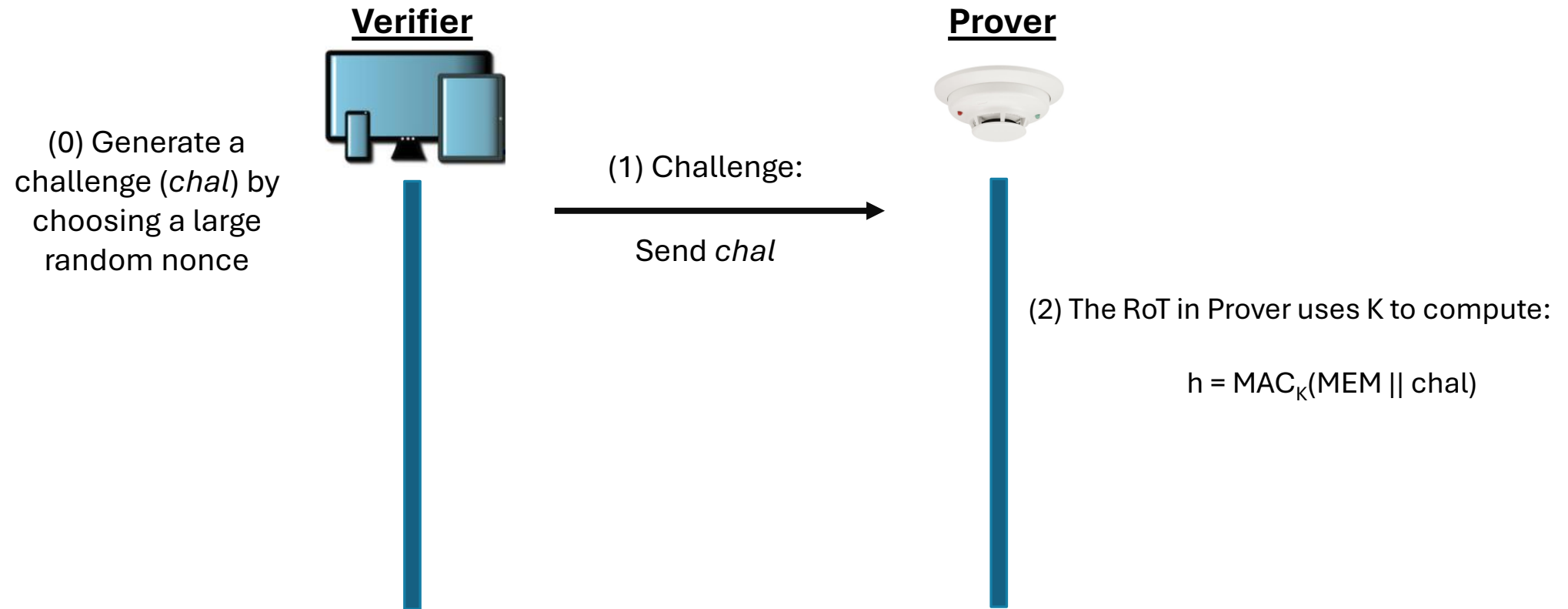
# Attestation

**Protocol 1:** Verifier and Prover's RoT share symmetric key ( $K$ )



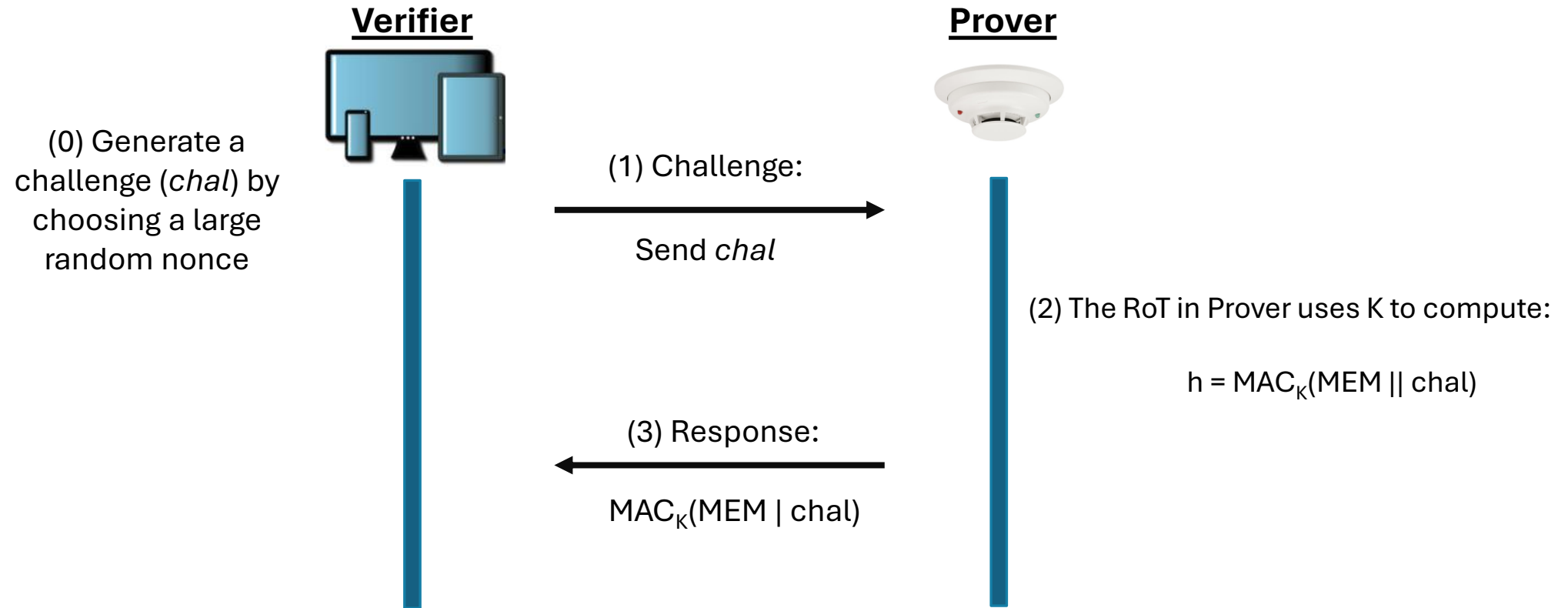
# Attestation

**Protocol 1:** Verifier and Prover's RoT share symmetric key (K)



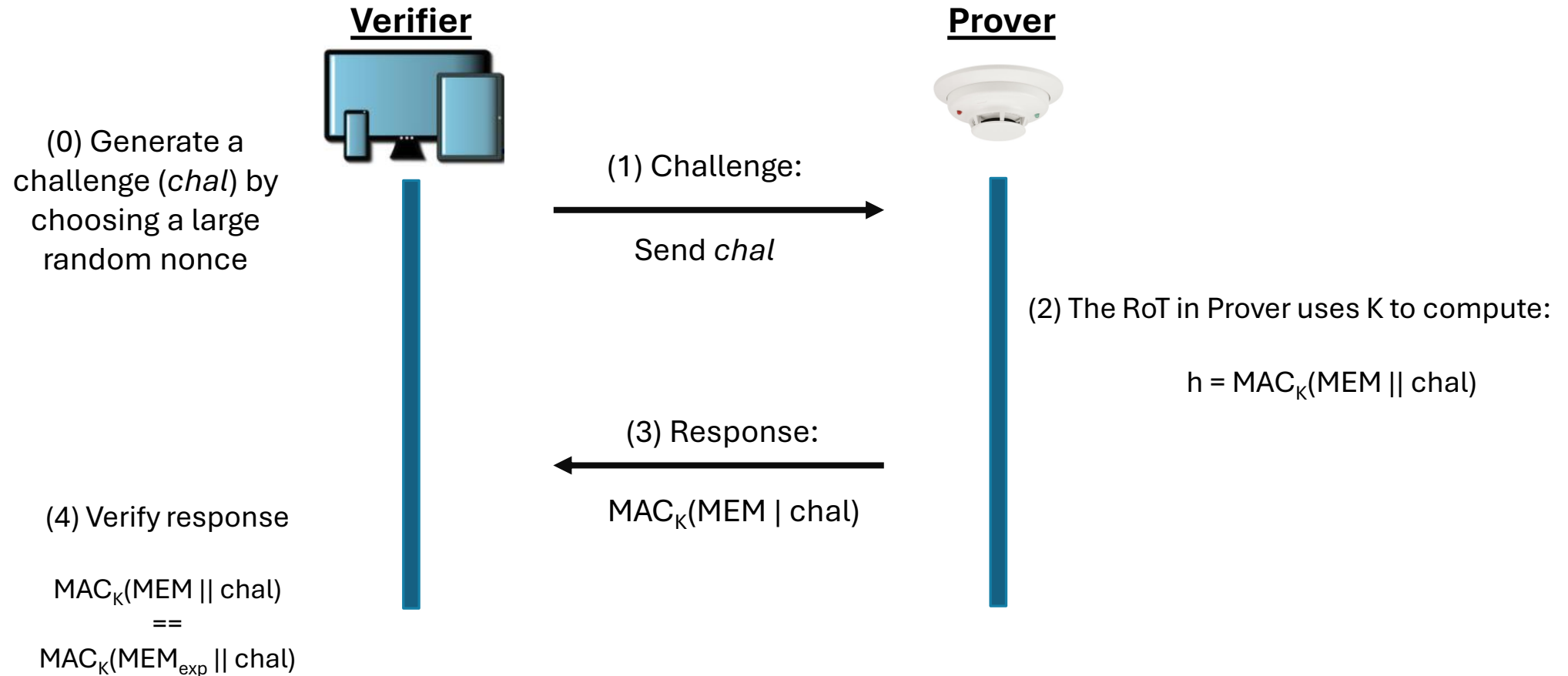
# Attestation

**Protocol 1:** Verifier and Prover's RoT share symmetric key ( $K$ )



# Attestation

**Protocol 1:** Verifier and Prover's RoT share symmetric key ( $K$ )



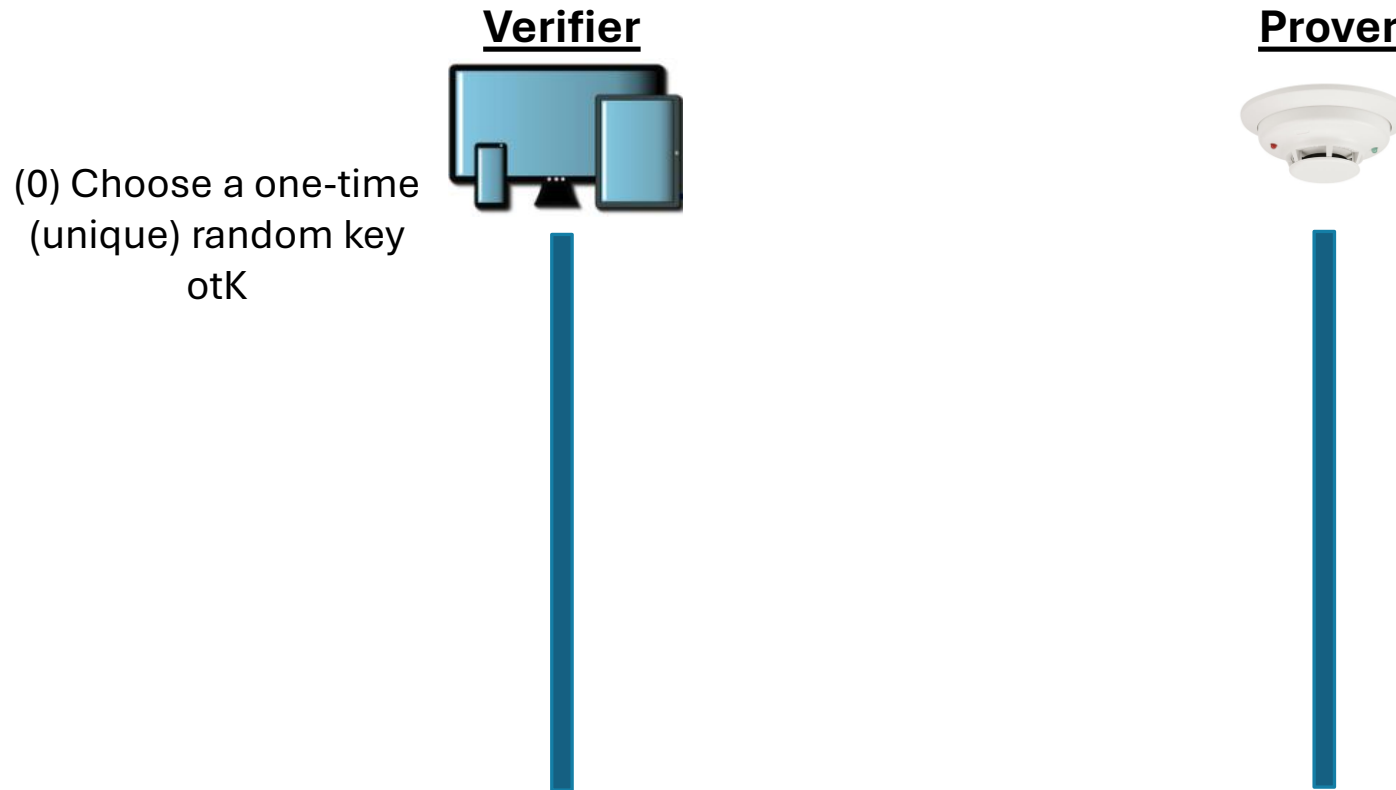
# Attestation

**In this protocol, Prover's RoT is trusted to:**

- Compute a MAC over the current *MEM* snapshot and *chal*
  - **Not** an older version of *MEM*
  - **Not** some other data or input
- Securely store and use *K* without ever leaking it
- To do so despite Prover's software attempting to potentially interfere

# Attestation

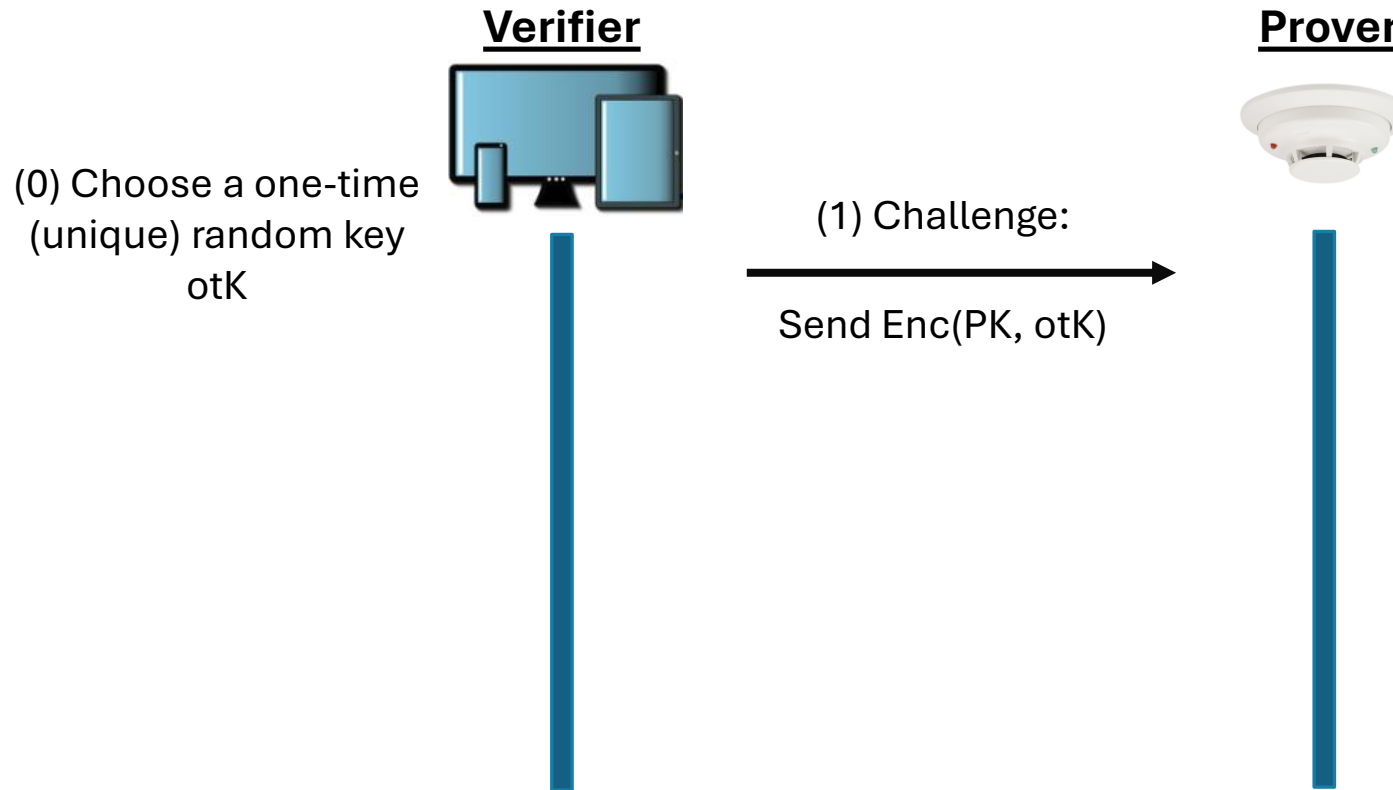
**Protocol 2:** Prover's RoT stores a secret key (SK); Verifier knows public key (PK)





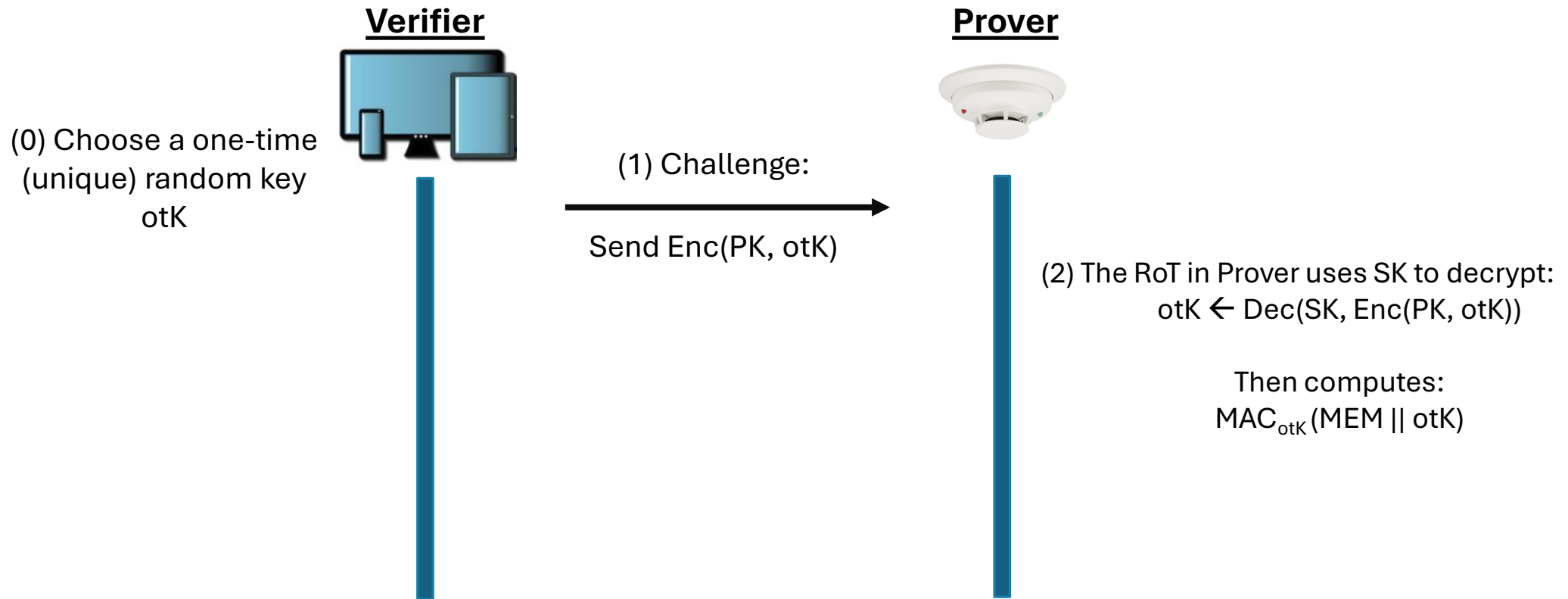
# Attestation

**Protocol 2:** Prover's RoT stores a secret key (SK); Verifier knows public key (PK)



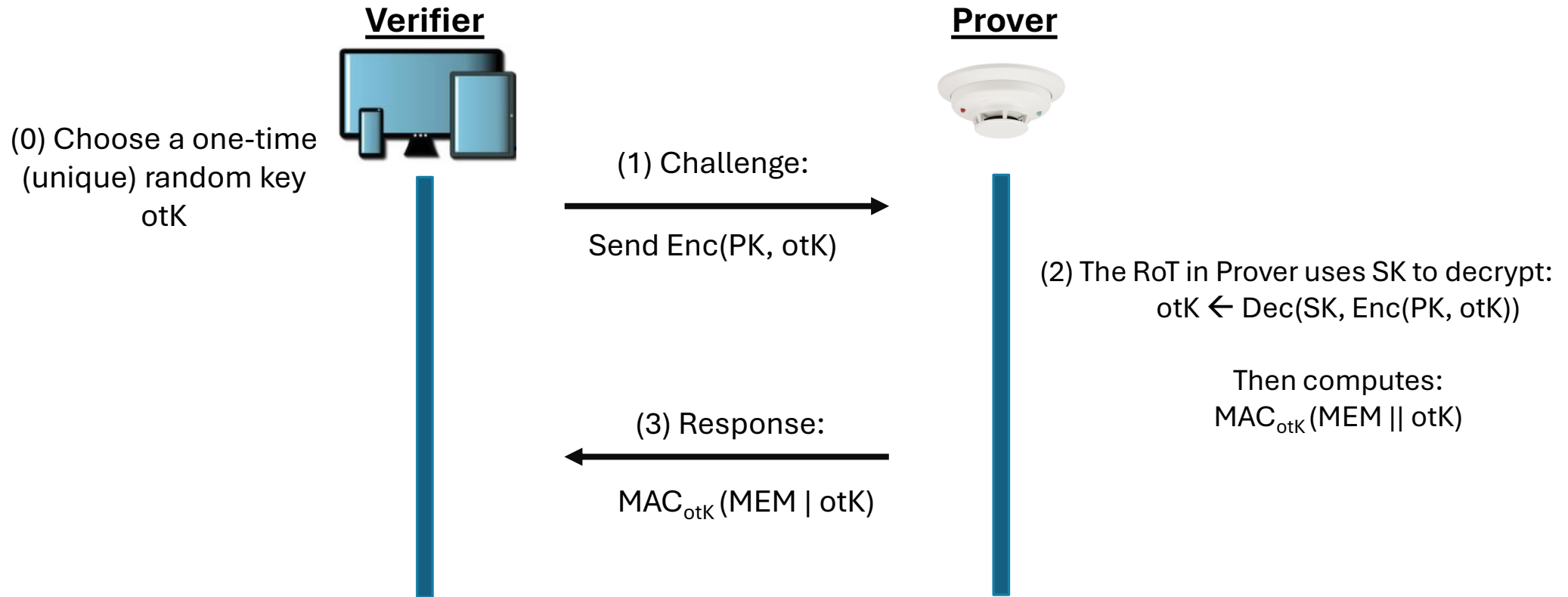
# Attestation

**Protocol 2:** Prover's RoT stores a secret key (SK); Verifier knows public key (PK)



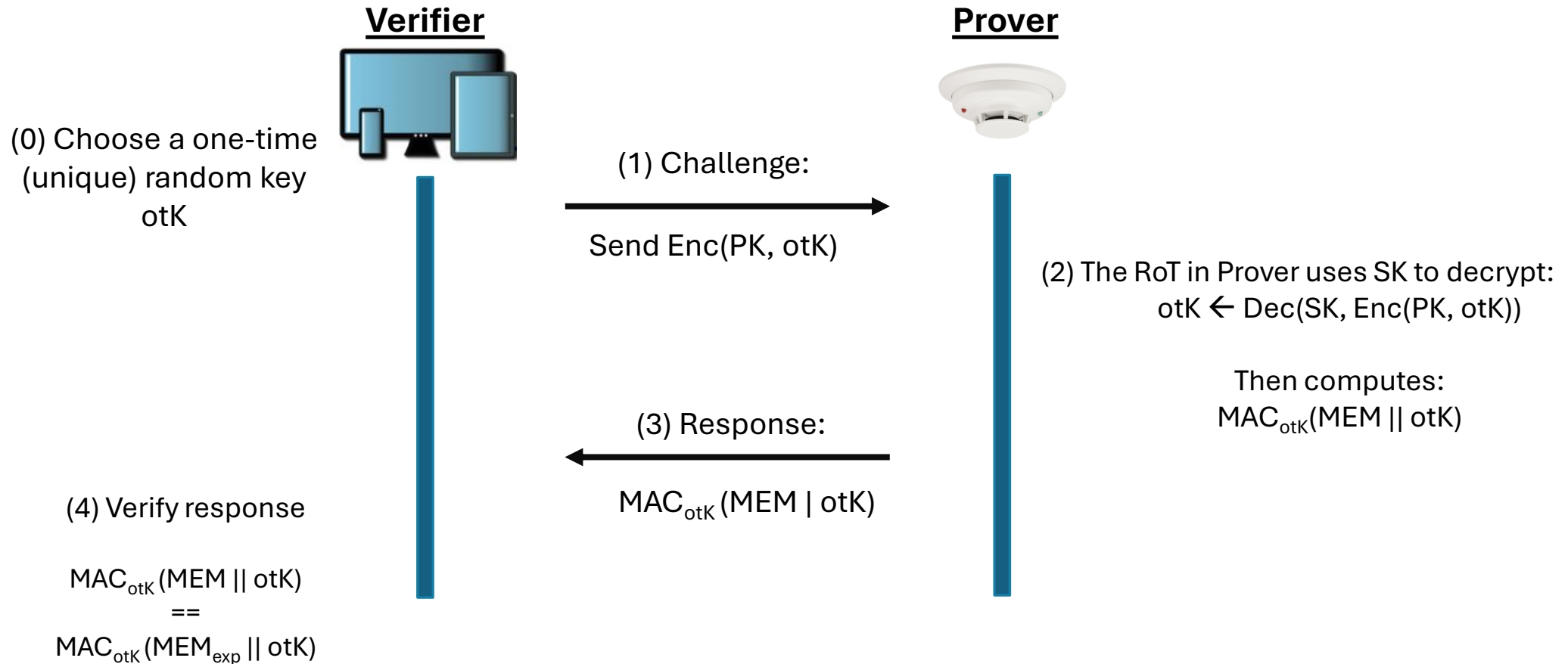
# Attestation

**Protocol 2:** Prover's RoT stores a secret key (SK); Verifier knows public key (PK)



# Attestation

**Protocol 2:** Prover's RoT stores a secret key (SK); Verifier knows public key (PK)



# Attestation

## **In this protocol, Prover's RoT is trusted to:**

- Securely store SK and otK
- Decrypt and compute MAC without leakage

## **More expensive:**

- One public key operation
- Good option for single-Prover & multiple-Verifier settings

## **Thought exercise:**

- How can we build an attestation protocol with only public key operations?

# That's all for today!

## Coming up....

- Next class:
  - Supply chain attacks & defenses (can attestation help?)
- After that: What can be used to obtain Prover RoT for Attestation?
  - Secure boot?
  - Something else? (... here comes Hardware & Mobile Security ...)

## Reminders:

- [A3 is due on July 11](#)
- No Class or instructor office hours next week
  - July 1 – it is Canada Day
  - July 3 – I will be away at a research conference

