ECE 382N-Sec (FA25):

L6: TEE Overview and Attestation

Neil Zhao

neil.zhao@utexas.edu

Before We Start

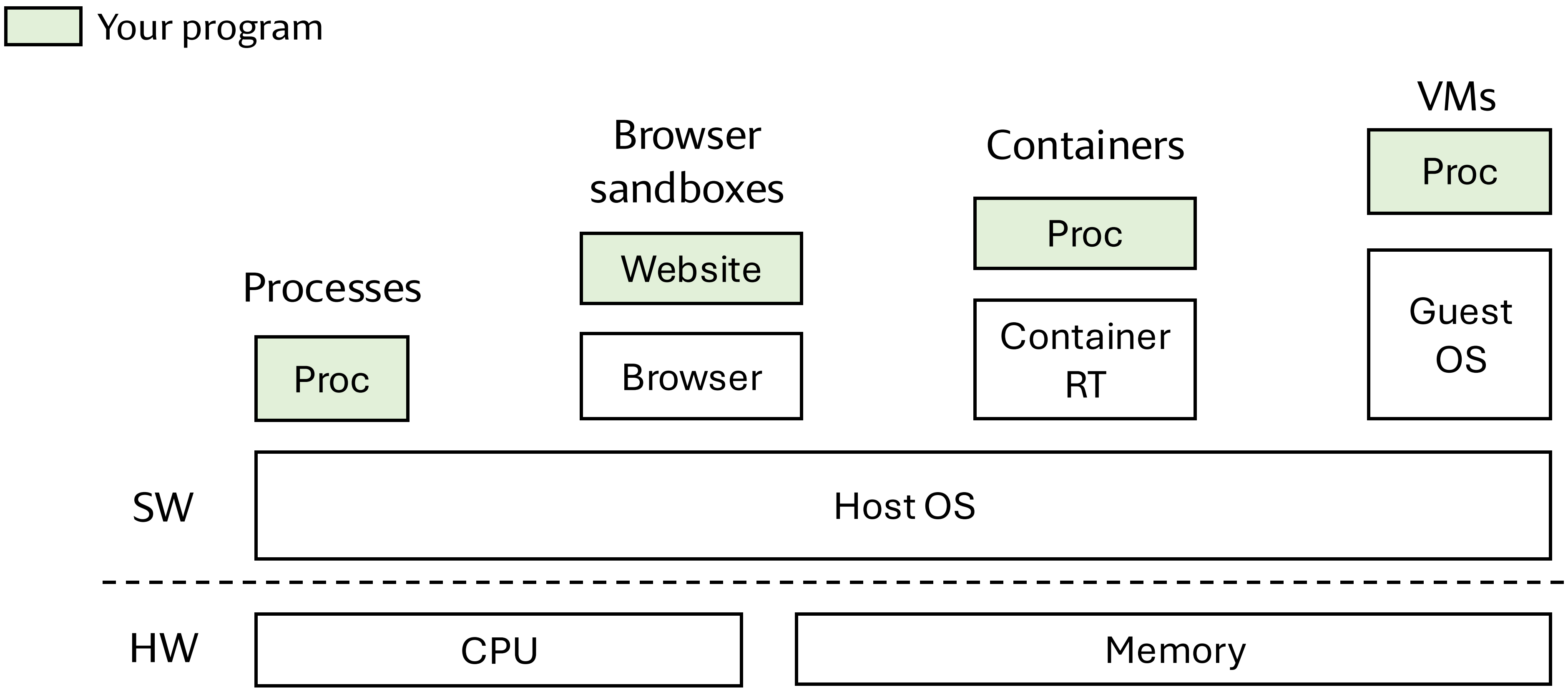
- Building Trusted-Execution Environments often involves various crypto tools

- This course focuses on general crypto primitives instead of specific algorithms and their implementations
    - These primitives are nice “hammers” to system builders
    - How these hammers are built is fascinating, but it’s out-of-scope for this course

- Our discussion simplifies certain aspects of these crypto primitives. It is good for building an intuitive understanding, but please do consult and follow various crypto standards for anything serious. Don’t re-invent the hammer!

- **A good reference:** “Serious Cryptography: A Practical Introduction to Modern Encryption” by Jean-Philippe Aumasson

Different Isolation Techniques



🟩 Your program

Processes

Proc

Browser sandboxes

Website

Browser

Containers

Proc

Container RT

VMs

Proc

Guest OS

SW

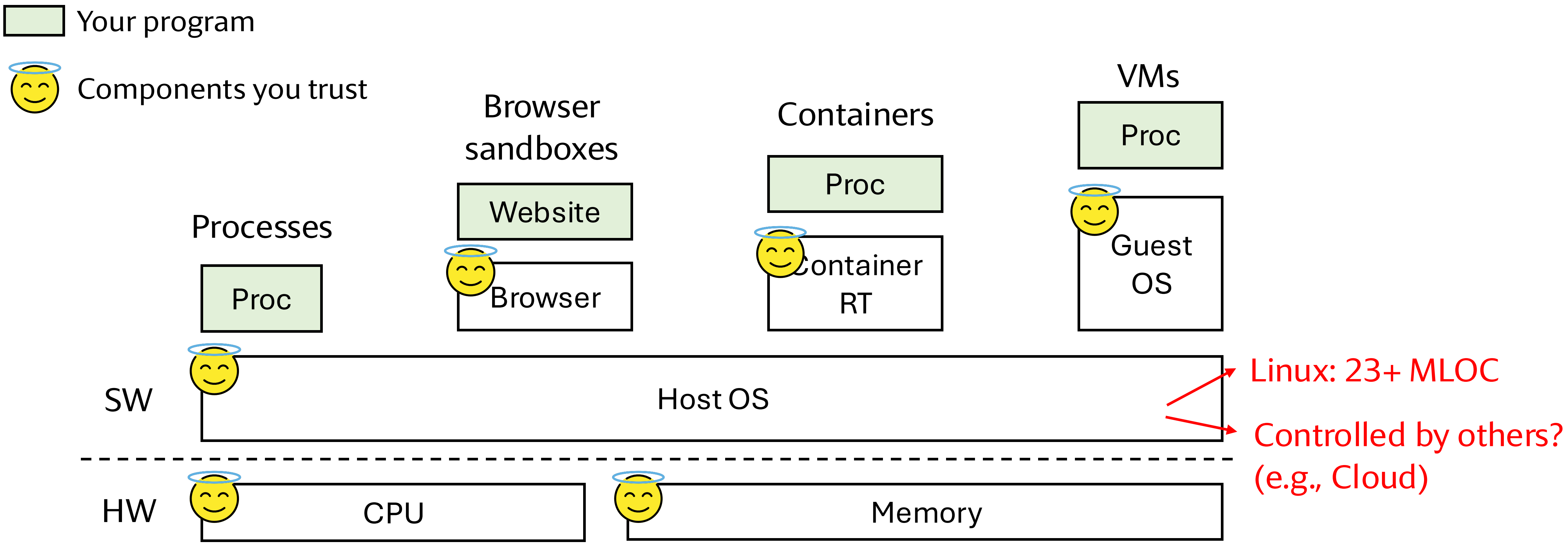
Host OS

HW

CPU

Memory

Trusted Computing Base (TCB)



🟩 Your program

😇 Components you trust

Processes

Proc

Browser sandboxes

Website

Browser

Containers

Proc

Container RT

VMs

Proc

Guest OS

SW

Host OS

HW

CPU

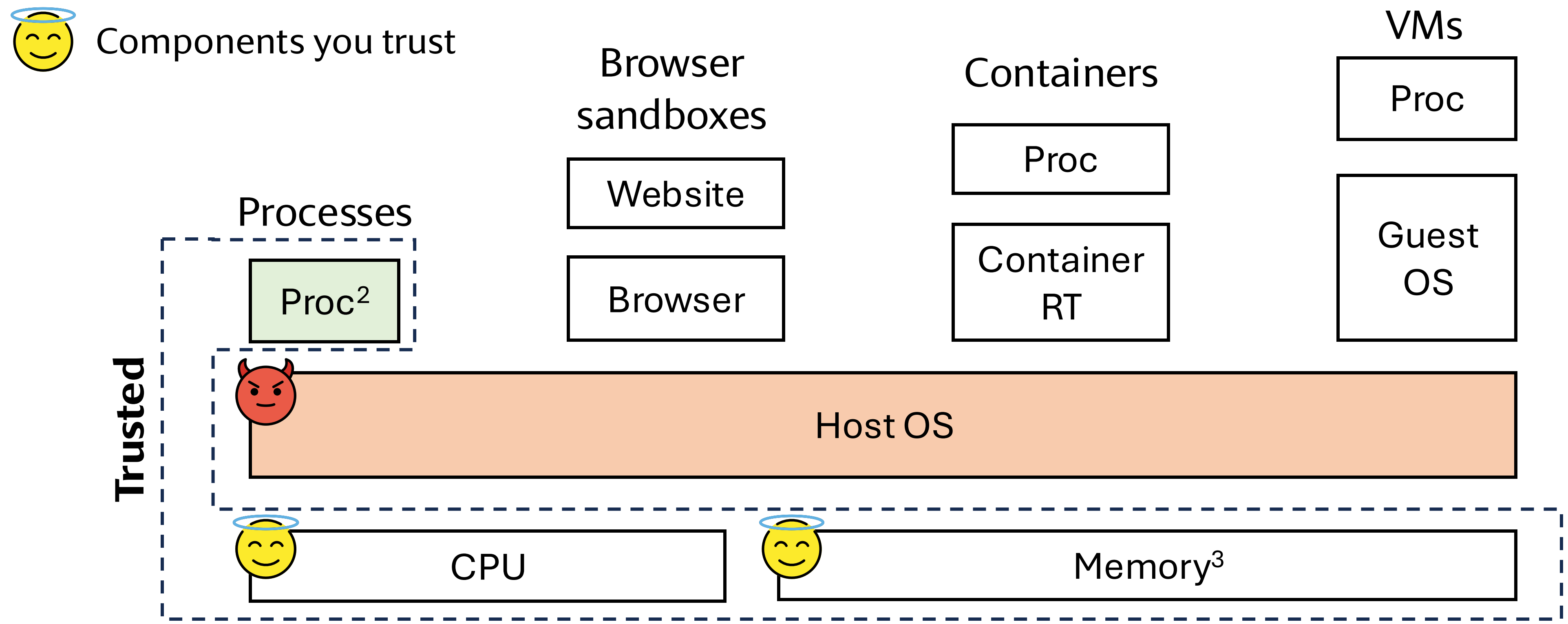
Memory

Linux: 23+ MLOC

Controlled by others? (e.g., Cloud)

The only components that can betray us, are the ones we trust

Trusted-Execution Environments (TEE)[1]



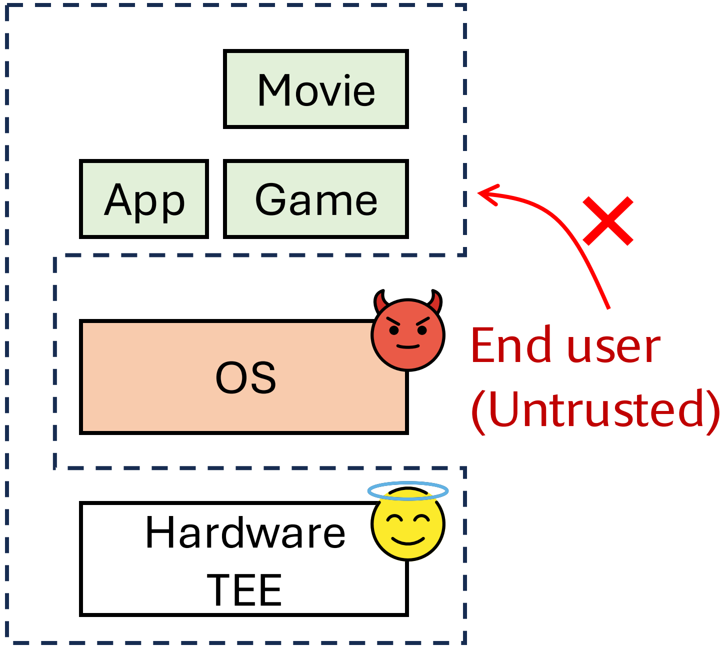
[1] TEE is a somewhat overloaded term. We focus on hardware-based TEEs

[2] The process may be divided into trusted and untrusted parts

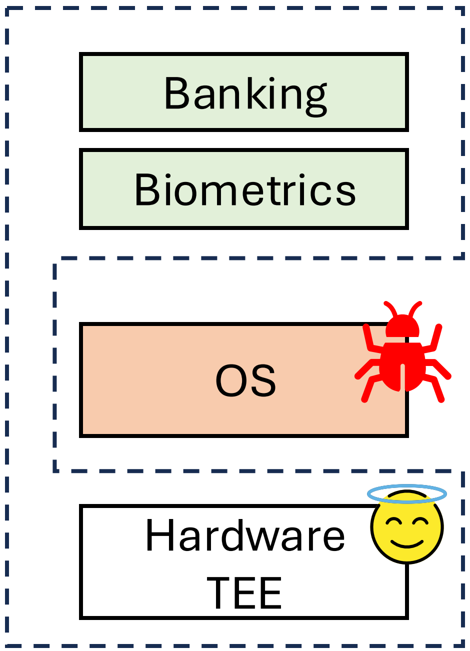
[3] Depending on the memory type and threat model, it may or may not be trusted

TEE Use Cases

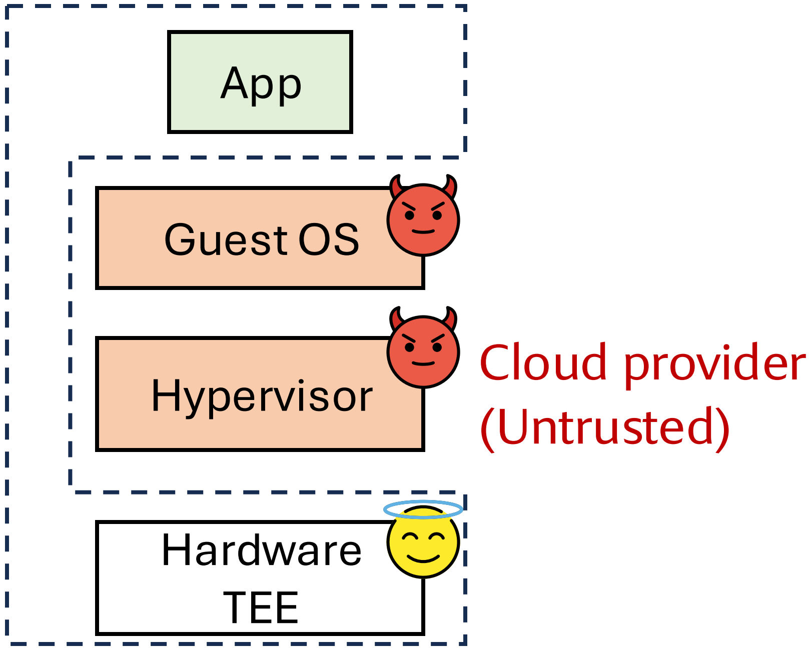
Copyright Protection



Minimizing TCB



Outsourcing Computation



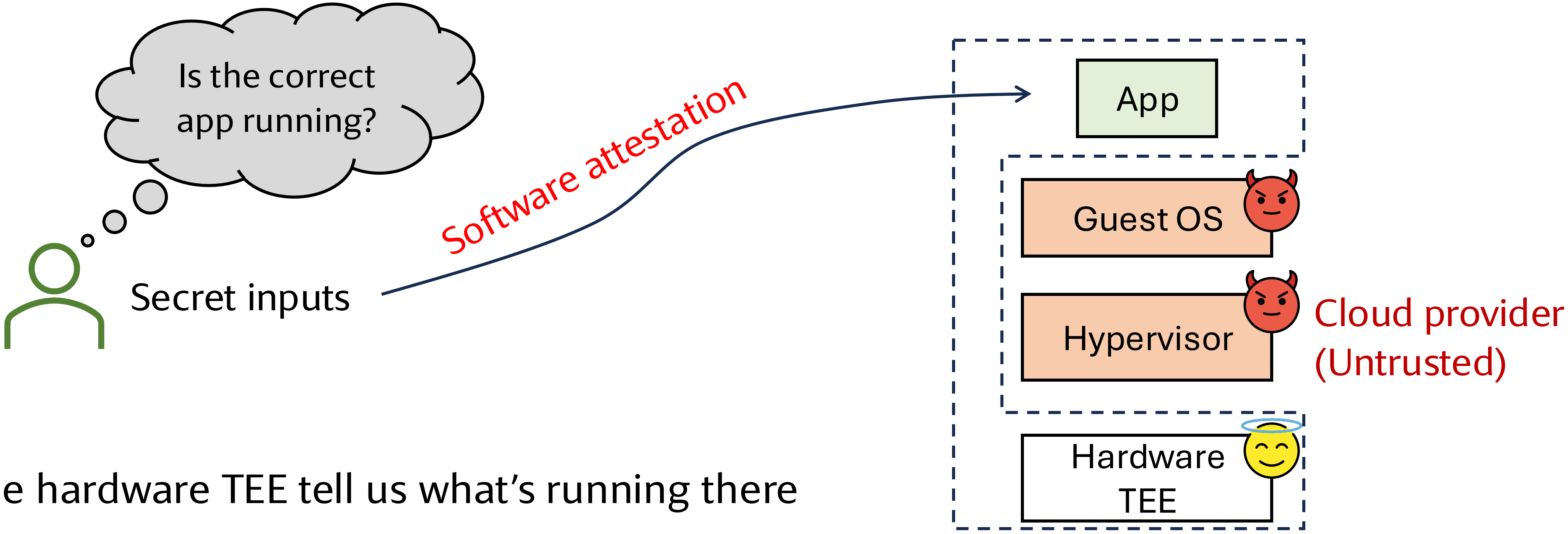
(Common*) Security Goals of TEEs

			Example Attacks	
			Software Attack	**Physical Attack**
✓	**Confidentiality**	Attacker cannot directly access my private program states (Side channel? Spectre?)	OS reads my pages	Bus snooping
✓	**Integrity**	Attacker cannot tamper with my program states (**Freshness:** Program state is up-to-date)	OS writes my pages	? Bus spoofing
✗	**Availability**	Attacker refuses to execute or give enough resources to my program	OS allocates no CPU time	Pull the plug

Note: Availability of the host is protected from my program in TEE---the OS can always terminate my program without my cooperation

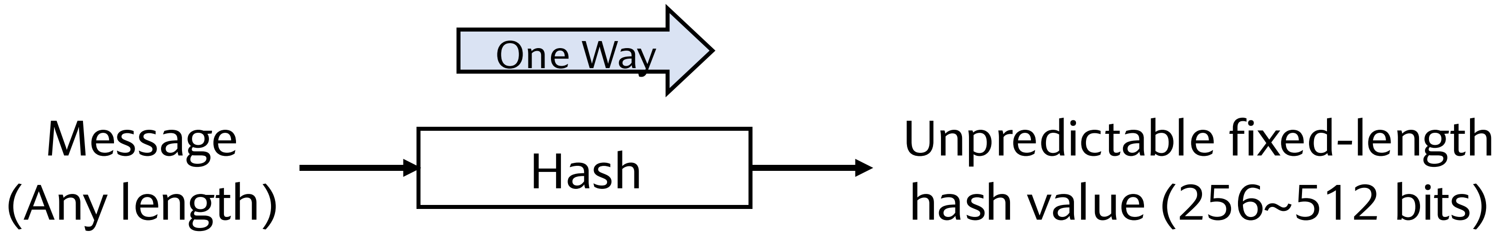
*Many variants exist

Establishing Trust?



Idea: Let the hardware TEE tell us what’s running there

Hammer 1: Cryptographic Hash Functions



One Way

Message (Any length)

Hash

Unpredictable fixed-length hash value (256~512 bits)

Avalanche effect (example uses SHA-256):

“44 students are registered for ECE-382N” → “`f1246dddd902aa8de27ddce24bd24031`…”

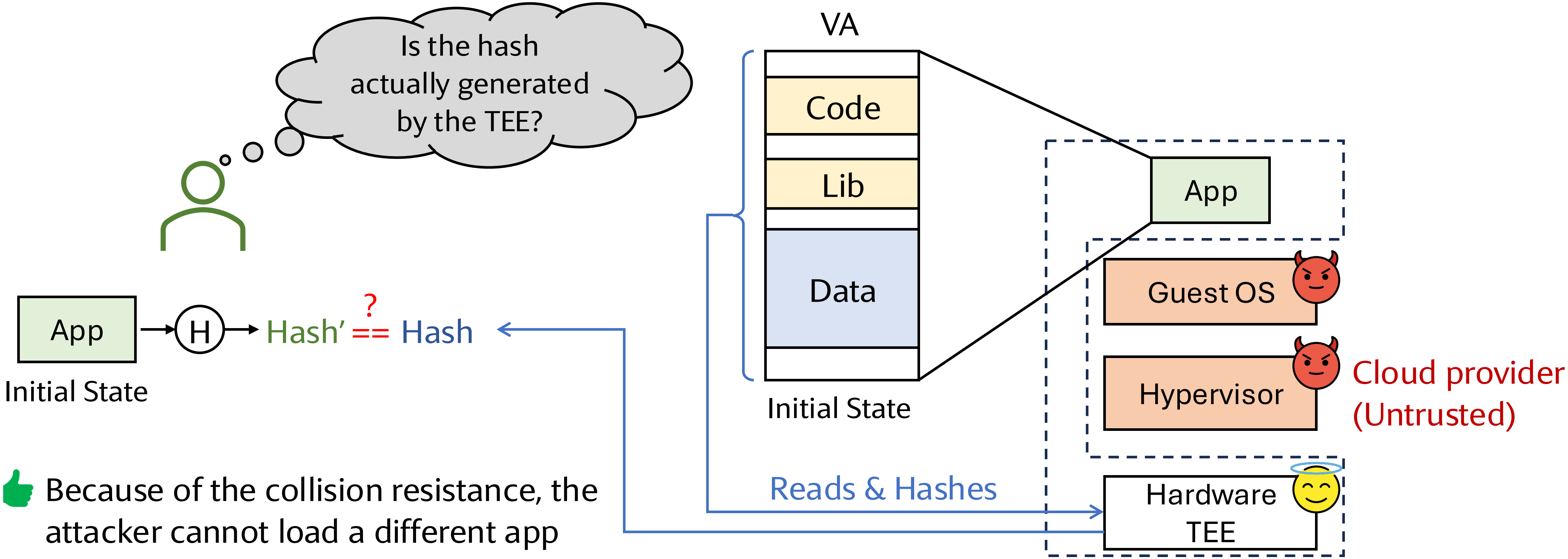
“43 students are registered for ECE-382N” → “`4bb1db3619a8442661fc9107b1767483`…”

Security properties:

- Preimage resistance: For any random hash value h, it’s practically impossible to find a message M such that Hash(M) = h in practice
- Collision resistance: Despite the inevitability of collision, it’s practically impossible find two distinct messages that hash to the same value

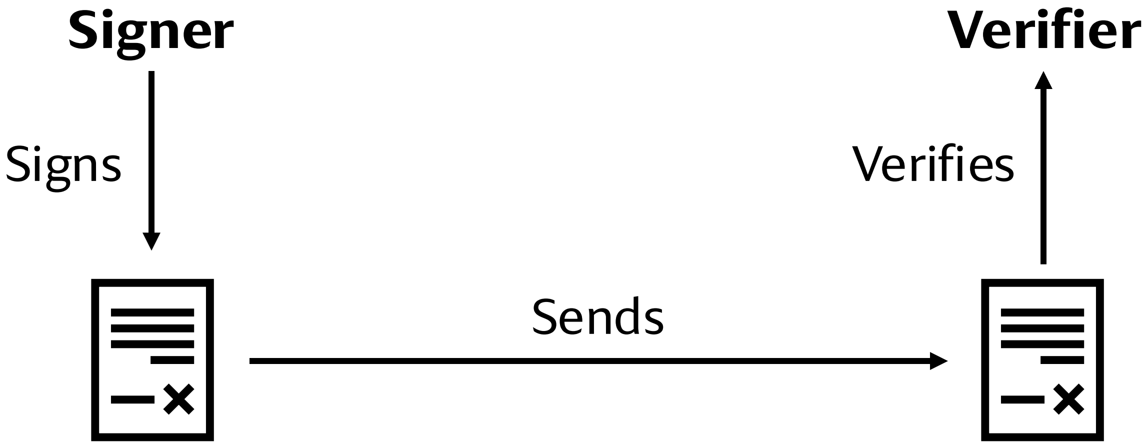
Note: Not all hash functions are cryptographic hash functions! E.g., Cyclic redundancy checks (CRCs) are not cryptographic hashes

Idea: “Measure” the Program State Using Crypto Hash



👍 Because of the collision resistance, the attacker cannot load a different app without user’s detection

Hammer 2: Digital Signature



Signer

Signs

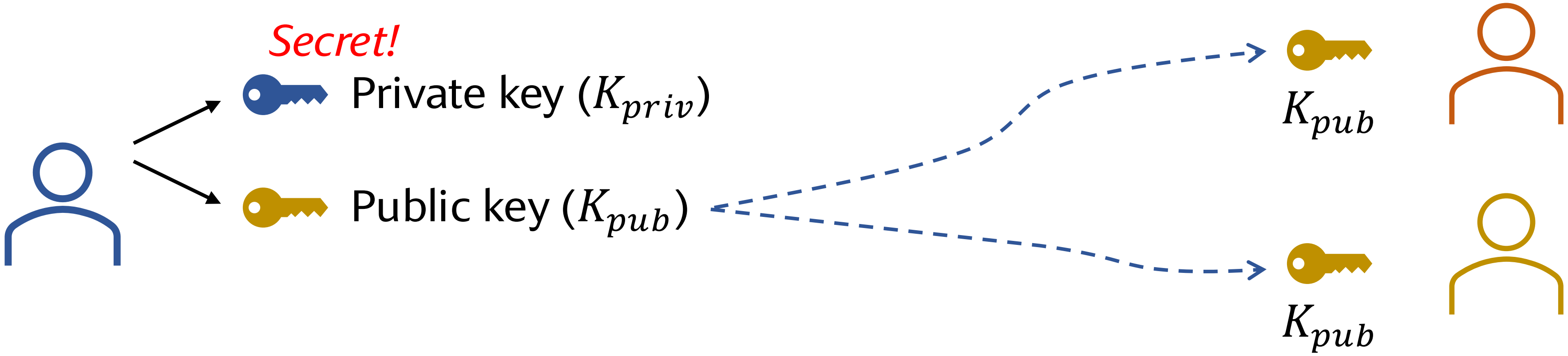
Sends

Verifier

Verifies

How to achieve this?

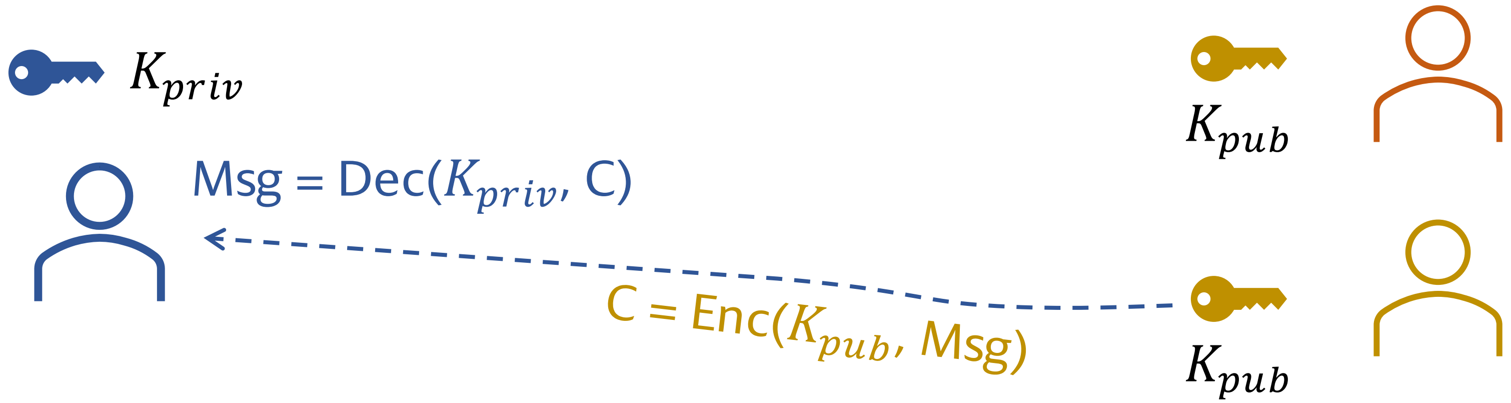
Hammer 3: Asymmetric Crypto



Examples: RSA, elliptic-curve cryptography, …

Property: Messages encrypted with K_{priv} can only be decrypted with K_{pub} and vice versa

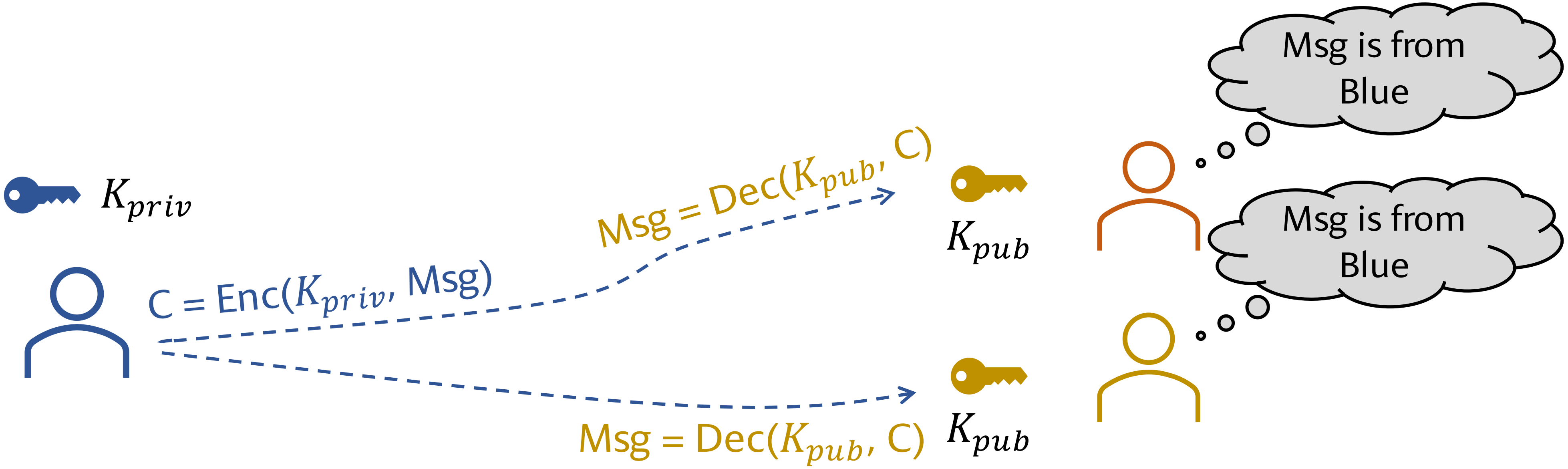
Hammer 3: Asymmetric Crypto



Examples: RSA, elliptic-curve cryptography, …

Property: Messages encrypted with K_{priv} can only be decrypted with K_{pub} and vice versa

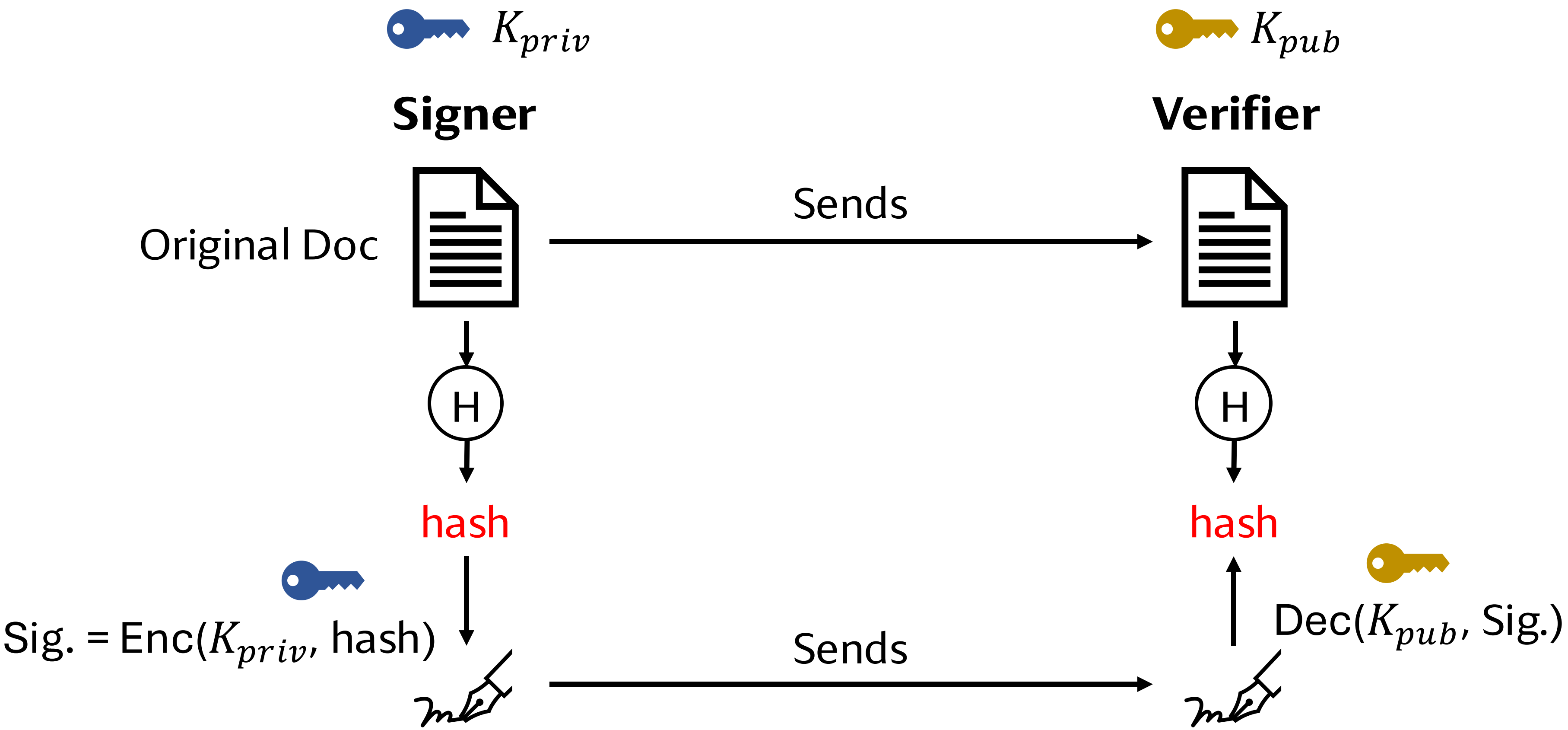
Hammer 3: Asymmetric Crypto



Examples: RSA, elliptic-curve cryptography, …

Property: Messages encrypted with K_{priv} can only be decrypted with K_{pub} and vice versa

Digital Signature



K_{priv}

Signer

Original Doc

H

hash

Sig. = Enc(K_{priv}, hash)

K_{pub}

Verifier

Sends

Sends

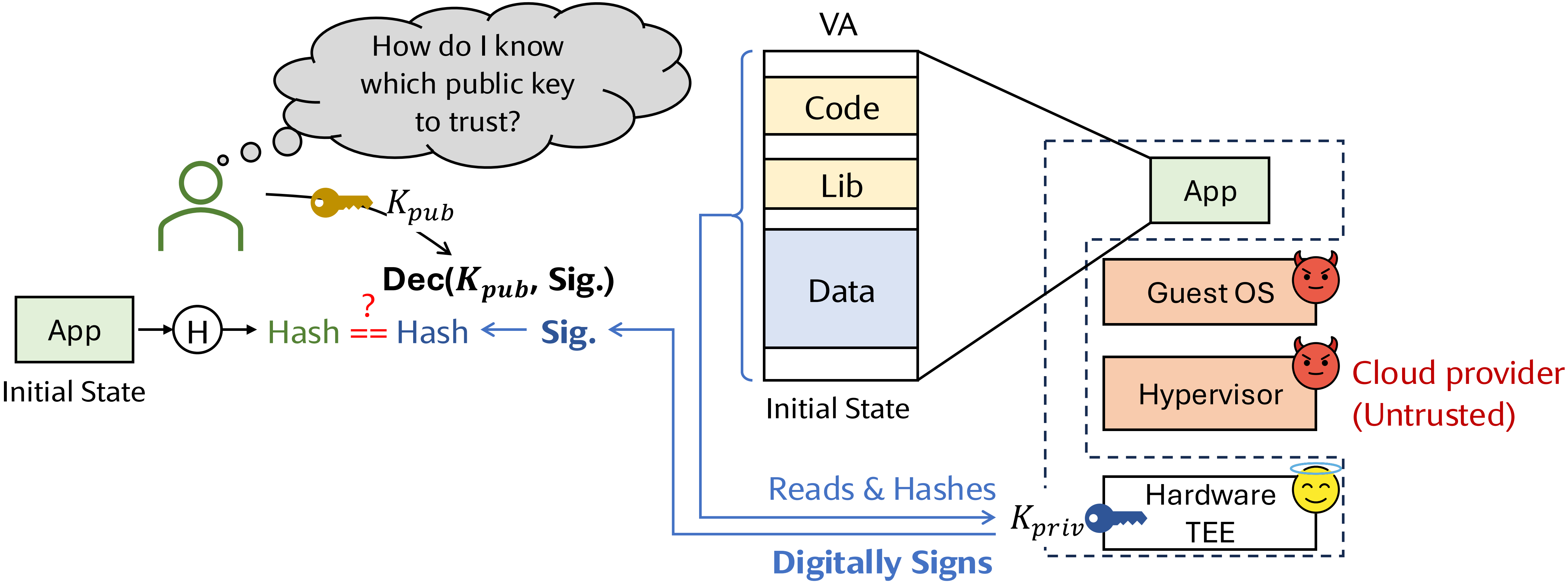
H

hash

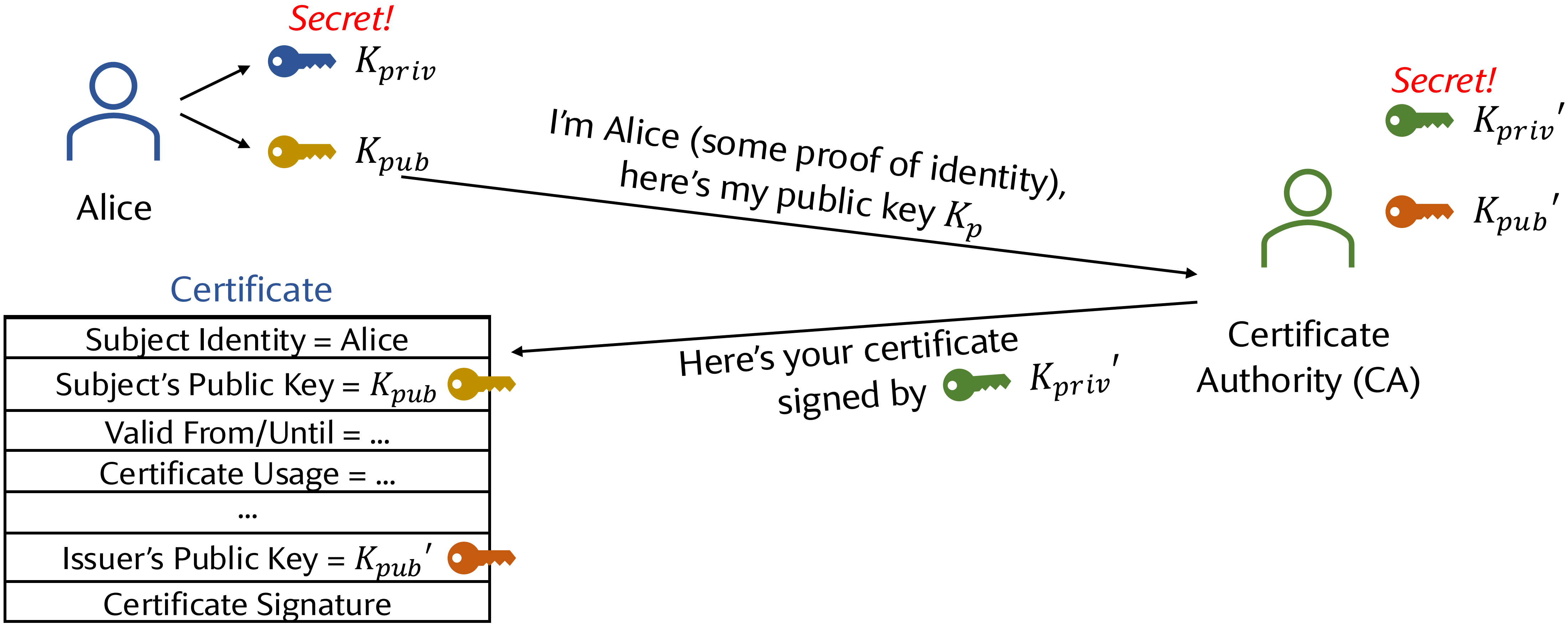
Dec(K_{pub}, Sig.)

Actual schemes are more complex than this

Software Attestation



Public Key Infrastructure



Secret!

K_{priv}

K_{pub}

Alice

I'm Alice (some proof of identity), here's my public key K_p

Secret!

K_{priv}'

K_{pub}'

Certificate Authority (CA)

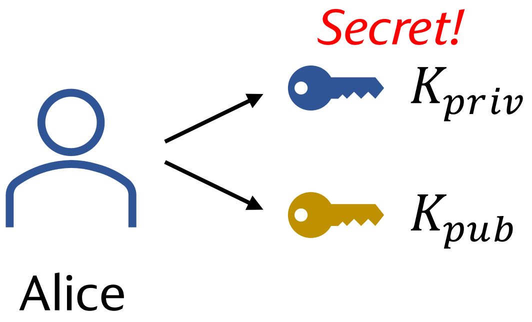
Here's your certificate signed by K_{priv}'

Certificate

Subject Identity = Alice
Subject's Public Key = K_{pub}
Valid From/Until = …
Certificate Usage = …
…
Issuer's Public Key = K_{pub}'
Certificate Signature

A Certificate is Like an ID Card

It binds the subject’s identity to their public key (or appearance)



Certificate

Subject Identity = Alice
Subject’s Public Key = K_{pub}
Valid From/Until = …
Certificate Usage = …
…
Issuer’s Public Key = K_{pub}'
Certificate Signature

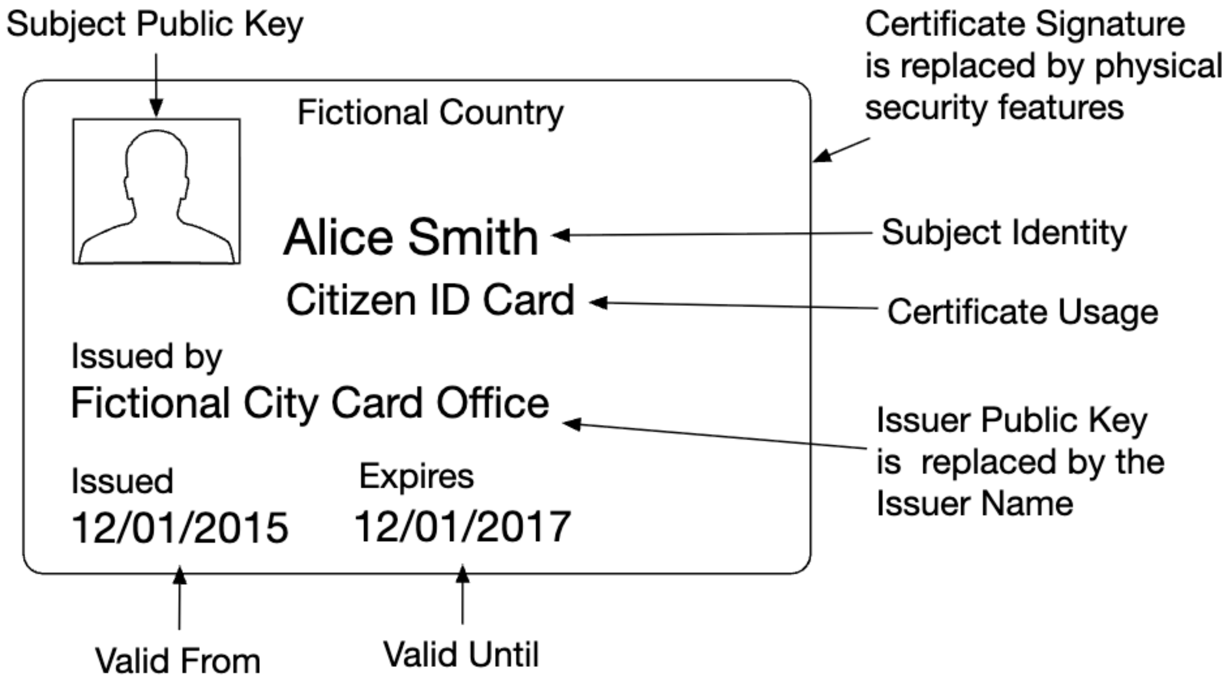
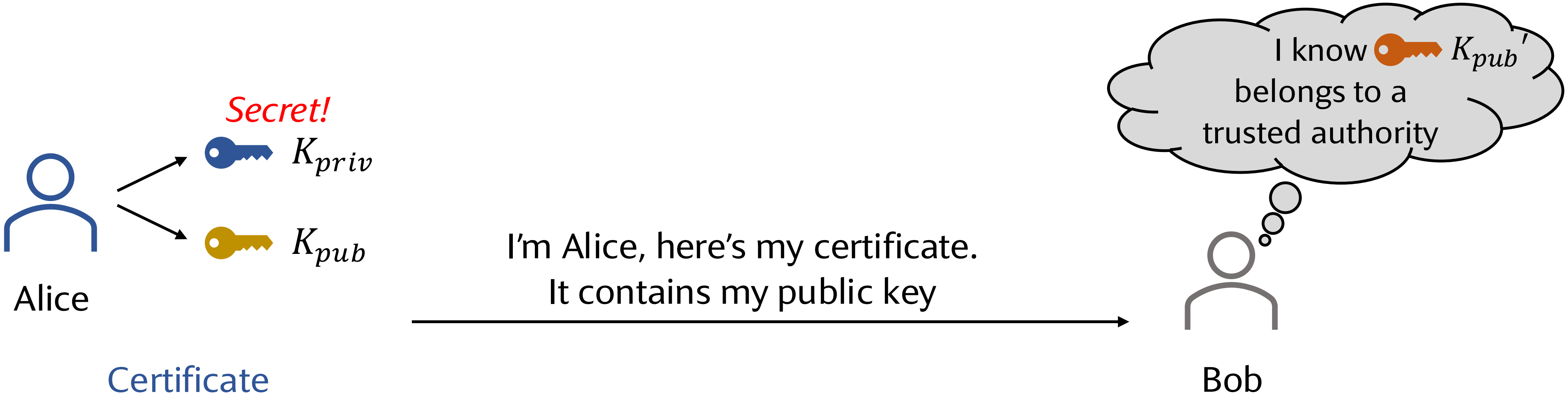


Illustration from “Intel SGX Explained” by Costan et al.

It’s a proof of identity-pubkey binding, not proof of identity

Validating a Certificate



Secret!

K_{priv}

K_{pub}

Alice

I'm Alice, here's my certificate.
It contains my public key

I know K_{pub}' belongs to a trusted authority

Bob

Certificate

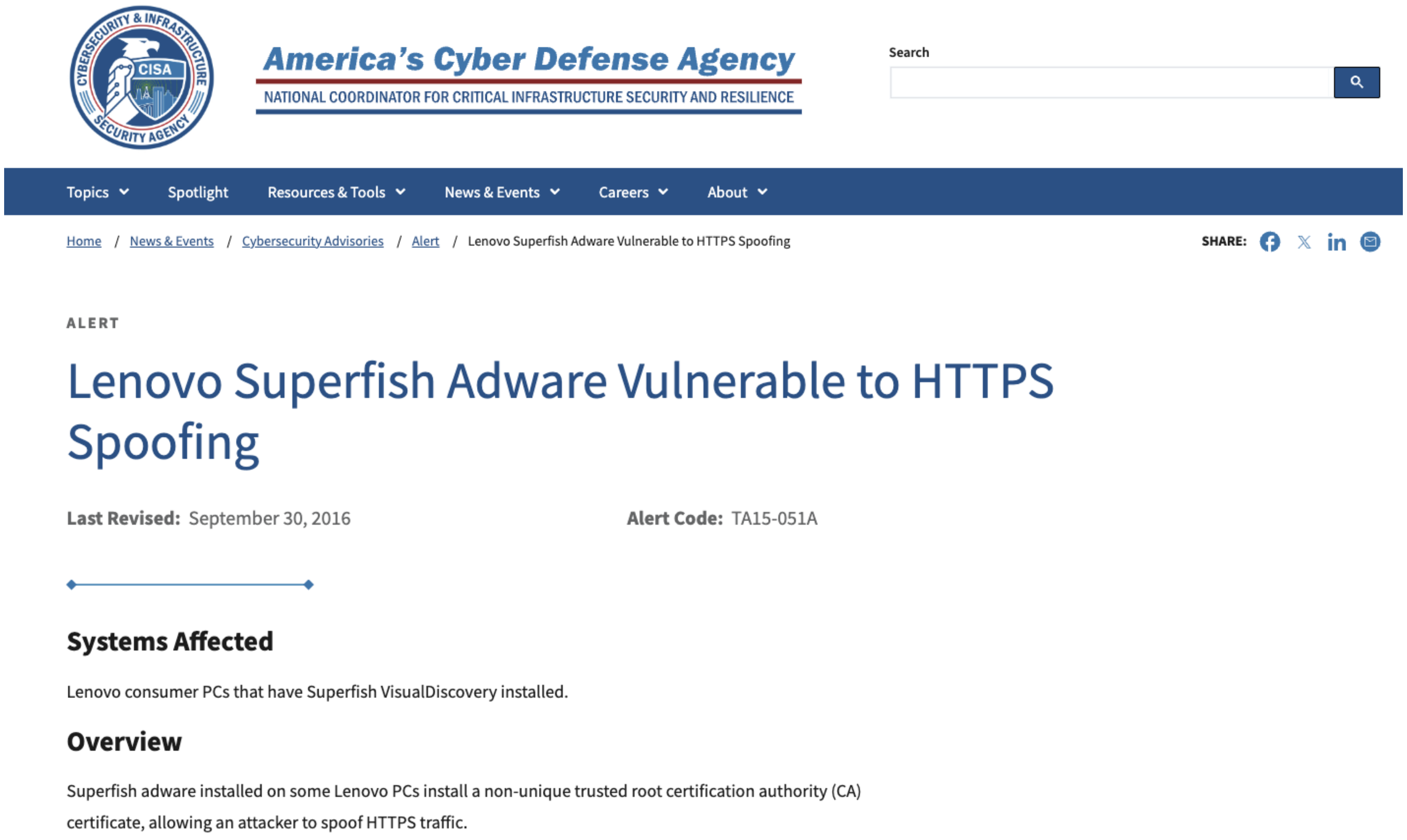
Subject Identity = Alice
Subject's Public Key = K_{pub}
Valid From/Until = …
Certificate Usage = …
…
Issuer's Public Key = K_{pub}'
Certificate Signature

Validate the certificate:

- Does it say "Alice"?
- Expired?
- Valid signature?
- Does the issuer's public key belong to a trust-worthy signer/CA
- …

Kicking the can down the road?

Lenovo Superfish Adware


America's Cyber Defense Agency
NATIONAL COORDINATOR FOR CRITICAL INFRASTRUCTURE SECURITY AND RESILIENCE
Search
Topics
Spotlight
Resources & Tools
News & Events
Careers
About
Home / News & Events / Cybersecurity Advisories / Alert / Lenovo Superfish Adware Vulnerable to HTTPS Spoofing
SHARE:

ALERT

Lenovo Superfish Adware Vulnerable to HTTPS Spoofing

Last Revised: September 30, 2016

Alert Code: TA15-051A

Systems Affected

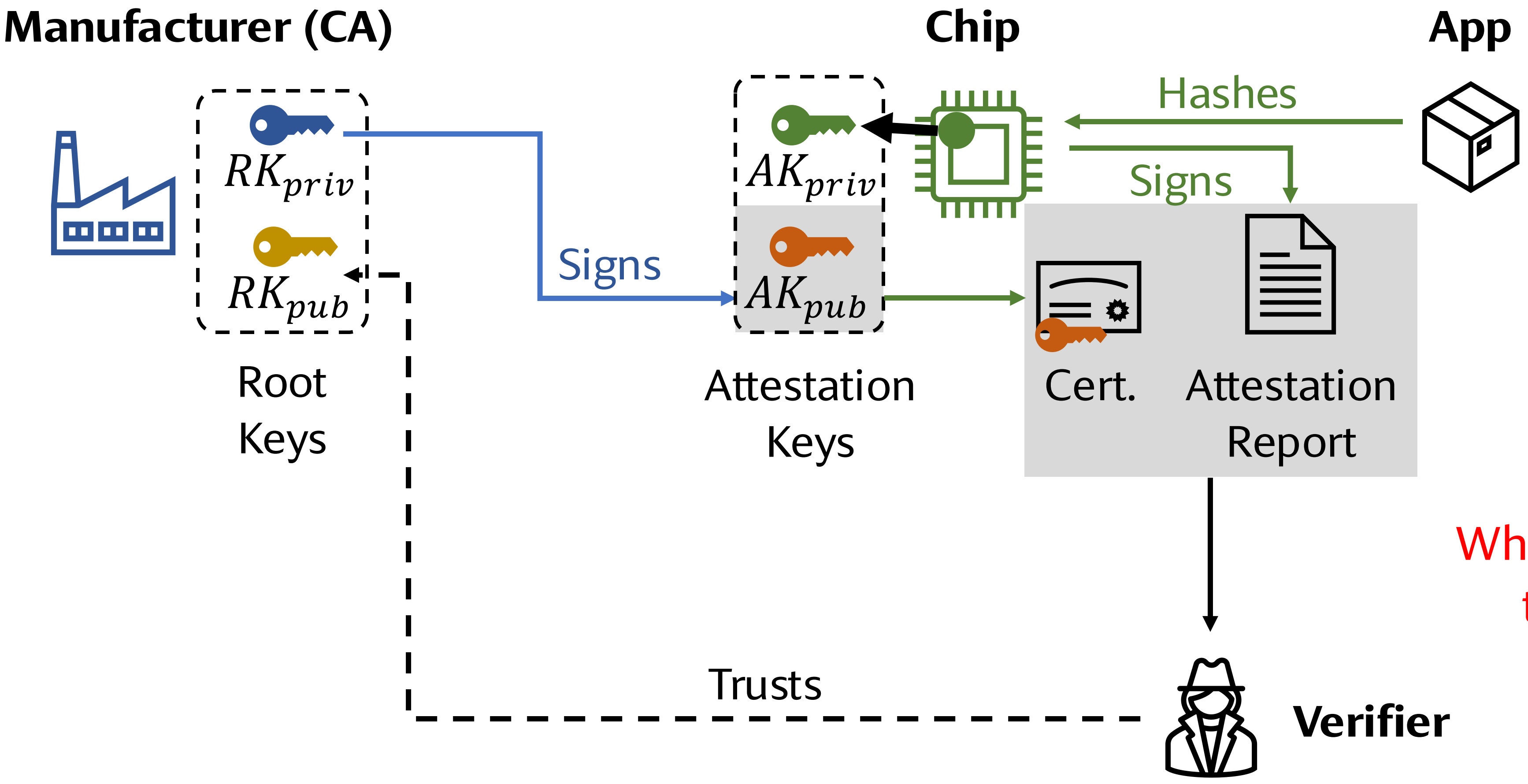
Lenovo consumer PCs that have Superfish VisualDiscovery installed.

Overview

Superfish adware installed on some Lenovo PCs install a non-unique trusted root certification authority (CA) certificate, allowing an attacker to spoof HTTPS traffic.

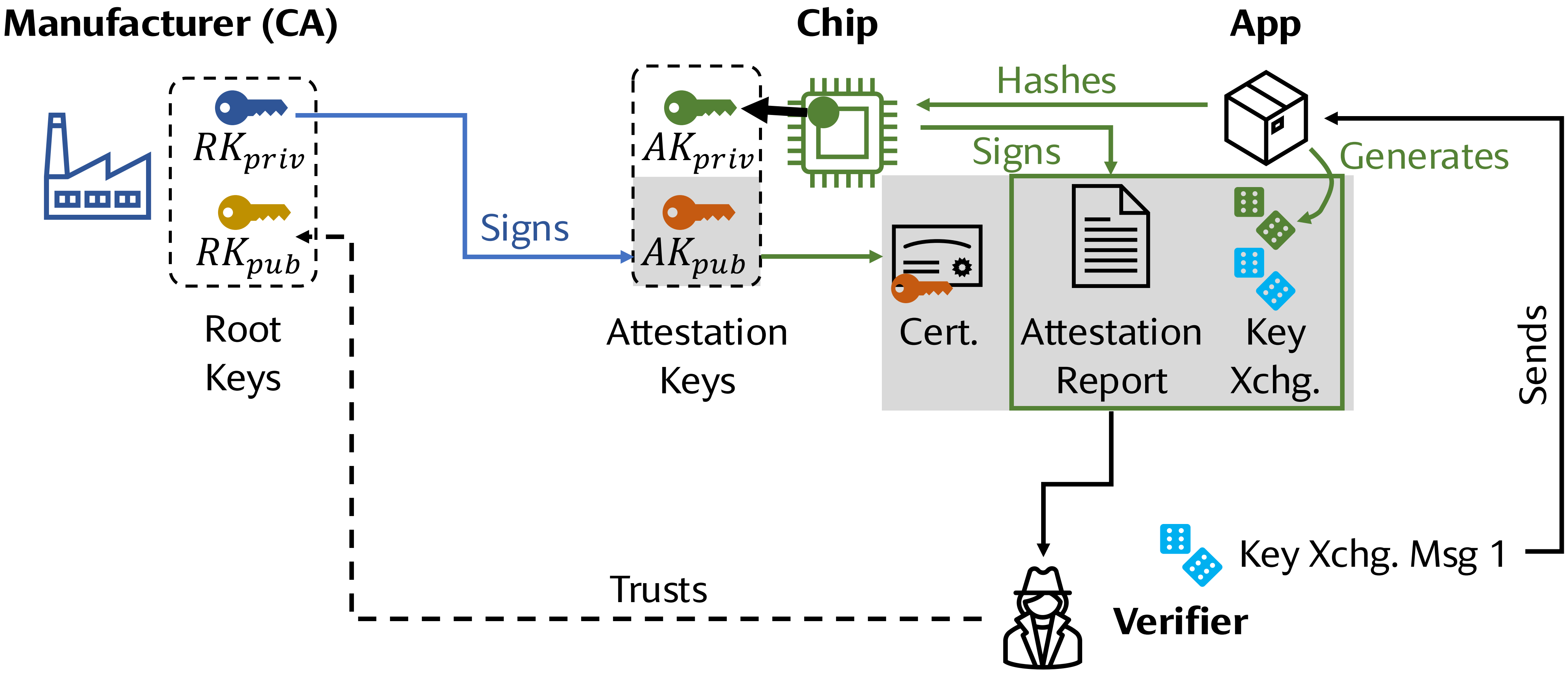
https://www.cisa.gov/news-events/alerts/2015/02/20/lenovo-superfish-adware-vulnerable-https-spoofing

Putting All the Pieces Together



What is wrong with this scheme?

Putting All the Pieces Together



Attestation in Practice (E.g., Legacy Intel SGX)

- Attestation report also contains hardware information
  - Is debug mode enabled?
  - Secure version numbers (for checking whether the TCB is up-to-date)
- Intel enhanced privacy ID (EPID) uses a group-signature scheme
  - Each processor belongs to an EPID group
  - Each processor creates signatures using its own private key
  - Signatures can be verified using the public key of the group that the processor belongs to
- Revocation list