Advanced Cryptographic Primitives

CMSC 23200/33250, Autumn 2018, Lecture 9

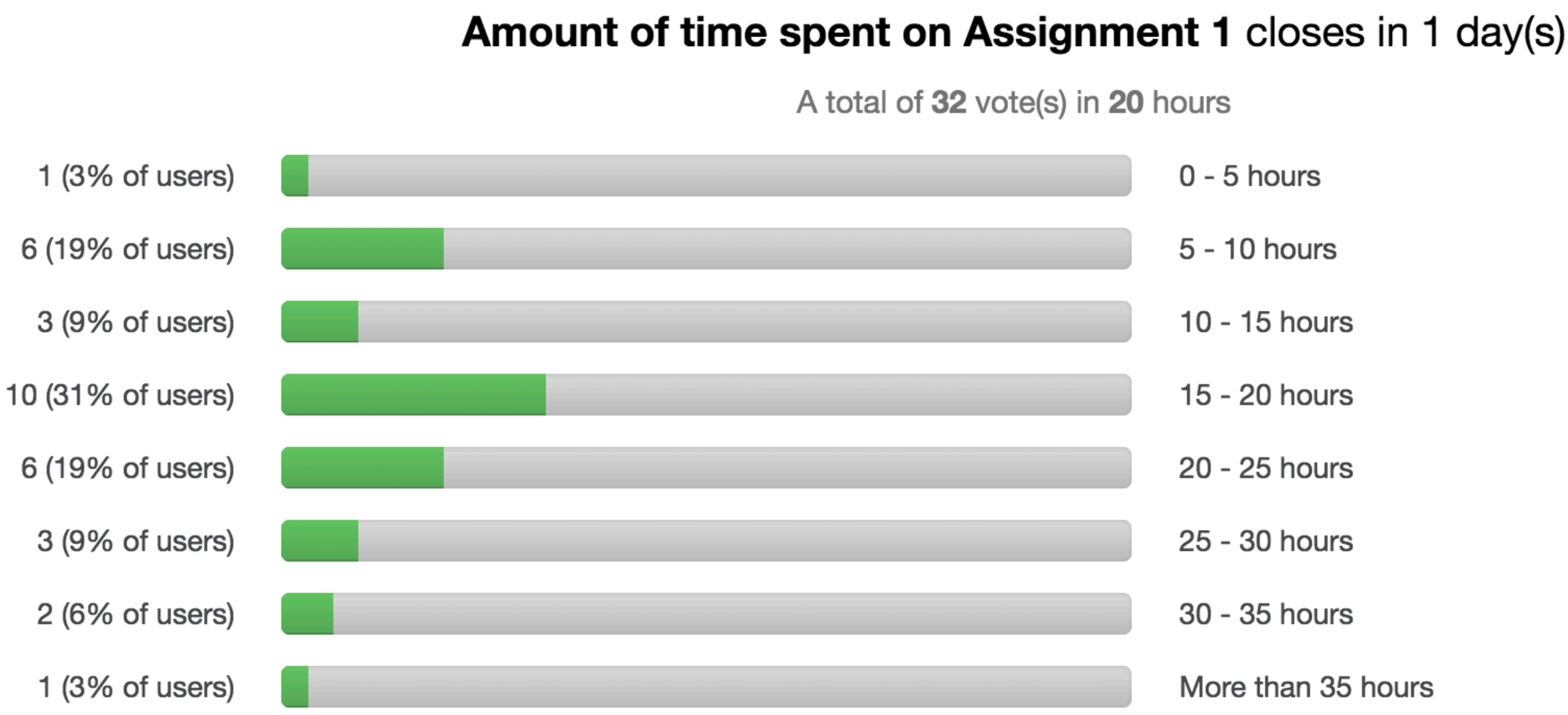
David Cash

University of Chicago

Plan

1. Wrap up signature padding
2. How signatures are used: Identification and Certs
3. Some advanced crypto tools

Assignment 1 Discussion



Assignment 1 Discussion

- It is not our intention that anybody needs to spend 30+ hours/ week on 232/332 assignments to succeed.
- These assignments are different from most classes
- You can still get an A, even if…
  - … your code does not perfectly succeed (ex: Only gets half of a flag)
  - … if your code has some bugs
  - … you don't solve every problem completely
- That is, we don't intend for everyone to solve every problem. They are meant to be challenging, thought-provoking, and educational.
- They are also a glimpse of real-world penetration testing.

Assignment 2 is Online

Due Next Wednesday at 11:59pm

Other RSA Padding Schemes: Full Domain Hash

N: **n**-byte long integer.
H: Hash fcn with **m**-byte output.
k = ceil((n-1)/m)

← Ex: **SHA-256, m=32**

Sign((N,d),M):

1. X←00||H(1||M)||H(2||M)||…||H(k||M)
2. Output $\sigma = X^d \bmod N$

Verify((N,e),M,σ):

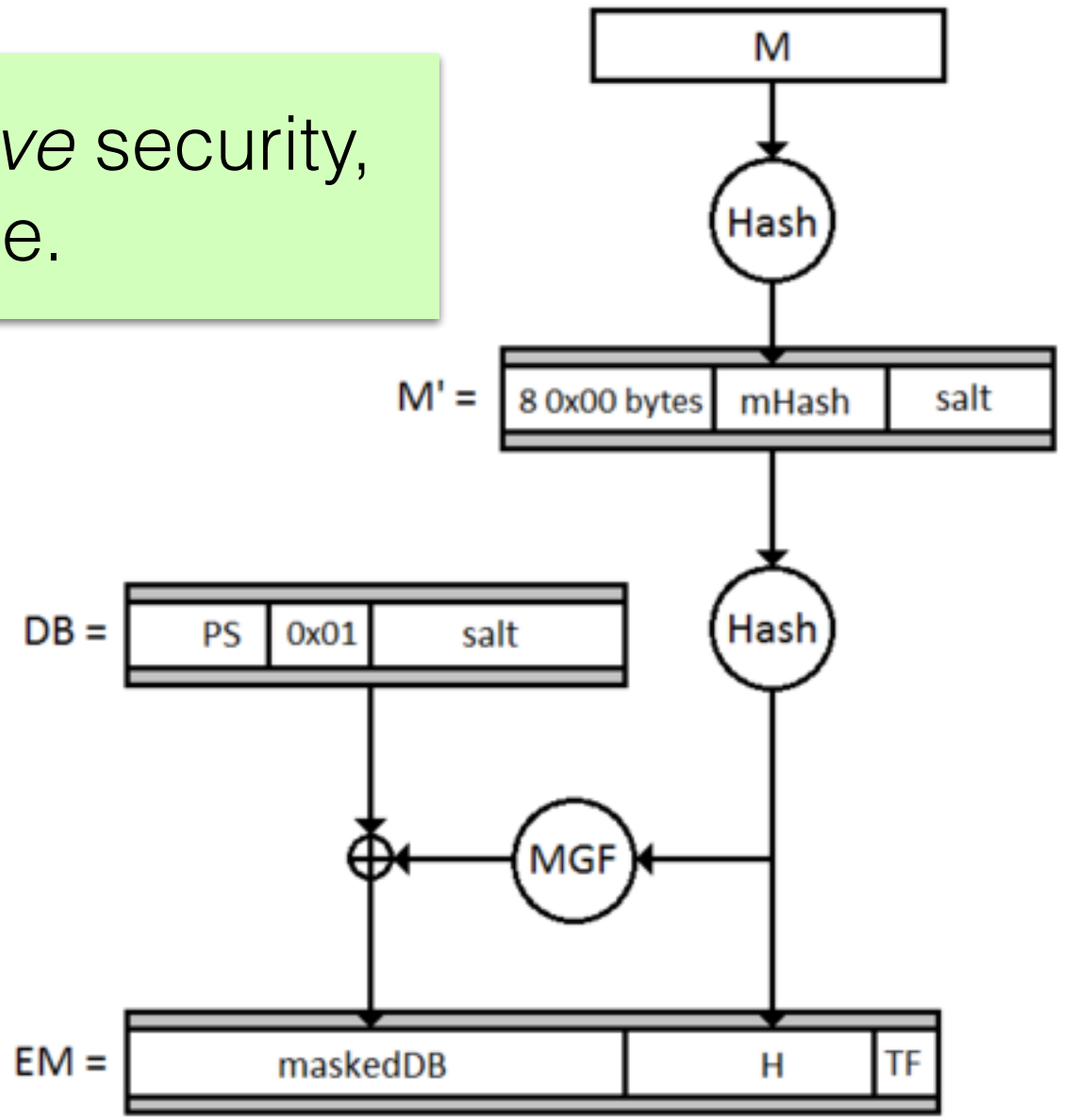
1. X←00||H(1||M)||H(2||M)||…||H(k||M)
2. Check if $\sigma^e = X \bmod N$

Bonus: Can *prove* security, in a strong sense.

Other RSA Padding Schemes: PSS

- Somewhat complicated
- *Randomized* signing

Bonus: Can *prove* security, in a strong sense.



RSA Signature Summary

- Plain RSA signatures are very broken
- PKCS#1 v.1.5 is widely used, in TLS, and fine if implemented correctly
- Full-Domain Hash and PSS should be preferred
- Don’t roll your own RSA signatures!

Other Practical Signatures: DSA/ECDSA

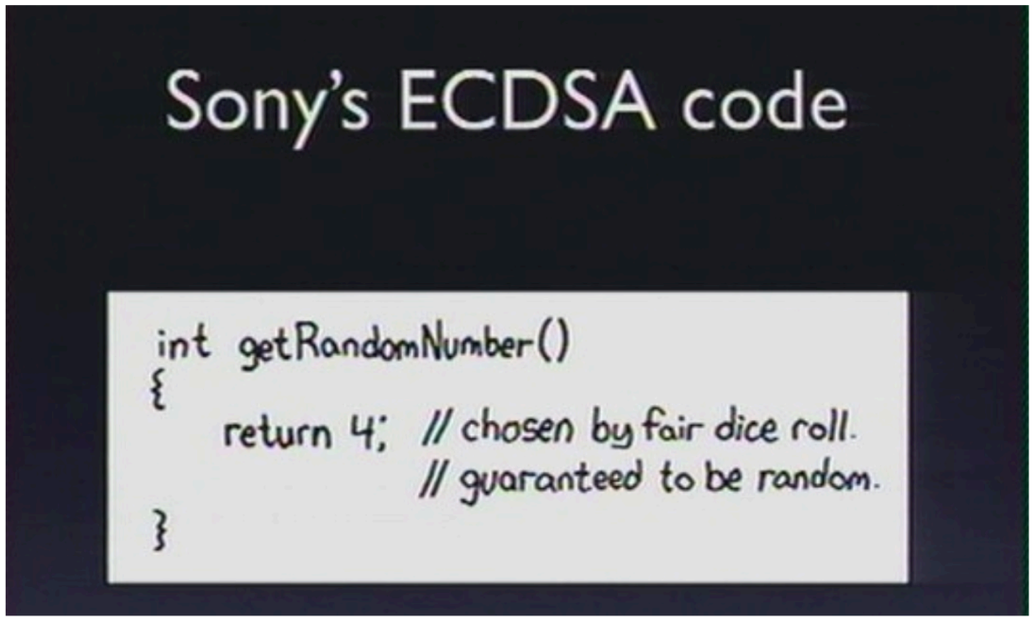
- Based on ideas related to Diffie-Hellman key exchange
- Secure, but ripe for implementation errors

—

Hackers obtain PS3 private cryptography key due to epic programming fail? (update)

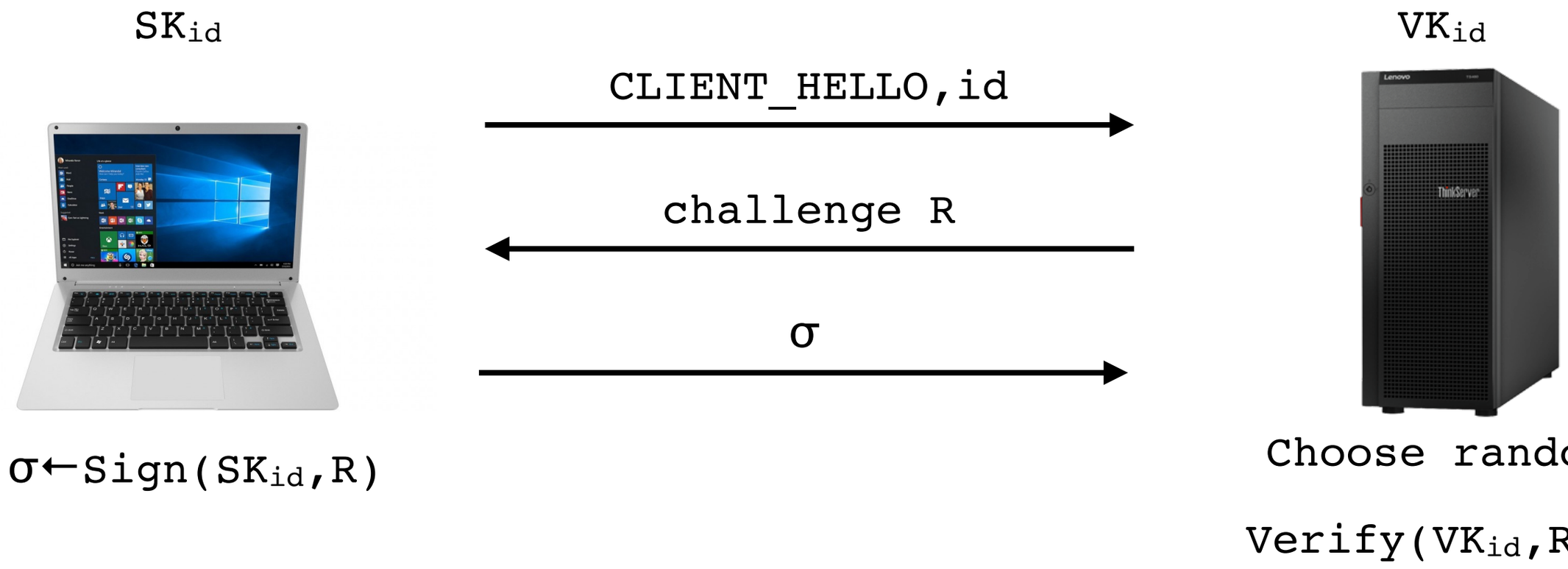
Sean Hollister
12.29.10

2
Shares



How signatures are applied: Identification Protocols

- SSH and TLS

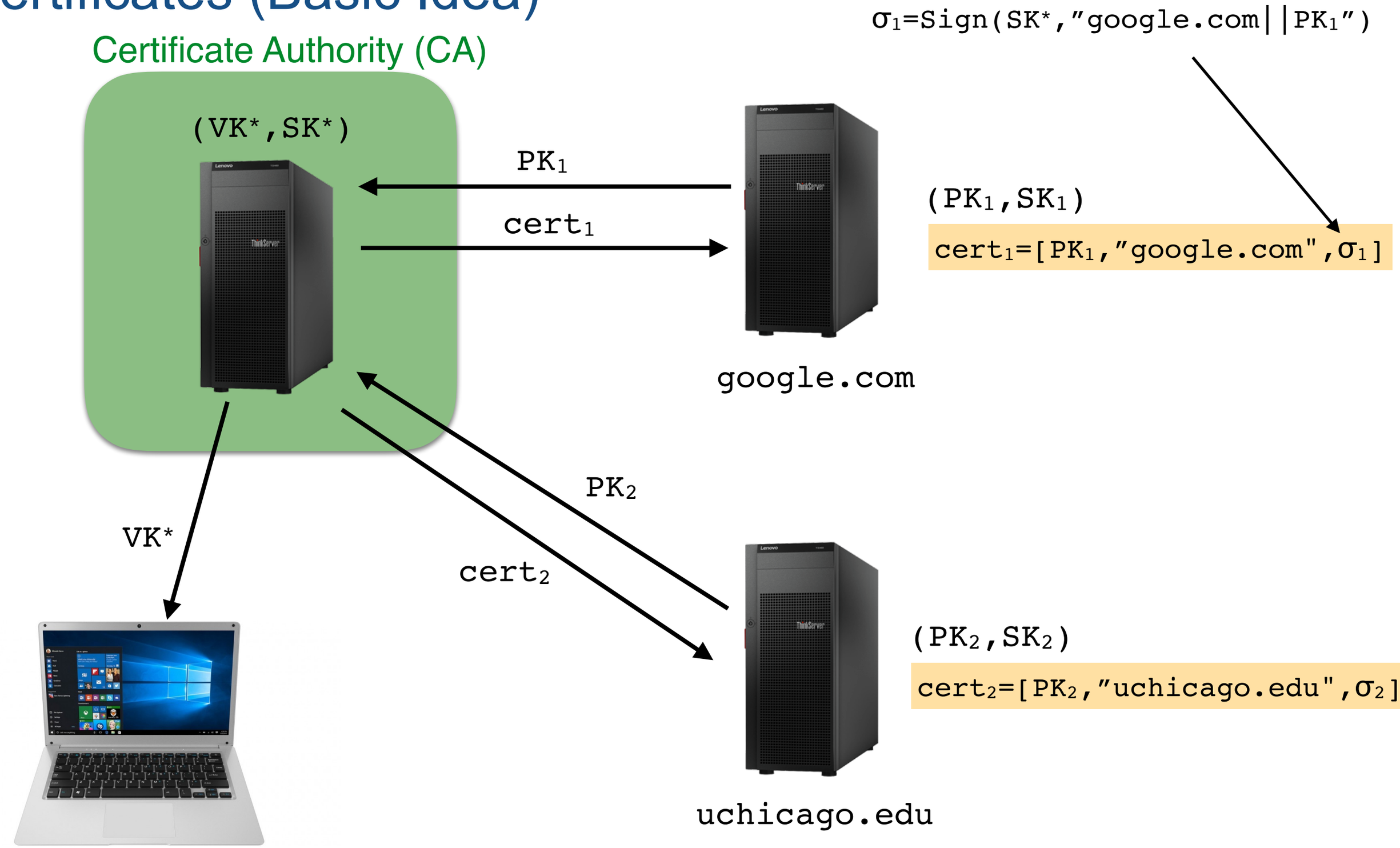


- Randomizing **R** stops *replay attacks*

How signatures are applied: Certificates

- Main application for digital signatures are *certificates*, used in TLS and other protocols

Certificates (Basic Idea)



- Trusted CA “issues certs”, i.e. signs public keys of other orgs.

Certificates (Basic Idea)

- Certificates in general are a tuple **`(PK,metadata,σ)`**
  - **`PK`** is public-key (may be for encryption, or for signature verification)
  - The **`metadata`** domain name, company info, sometimes addresses, crypto protocols to use, expiration date, etc.
  - **σ** is a signature on **`PK+metadata`** under CA's signing key.
- Issuing a certificate involves varying levels of due diligence by CA
- If CA is negligent, then entire system is not trustworthy!

Authenticated Key Exchange with Certs

CA's verification key VK^*

(Pick random AES key K)

K

$ClientHello$

$cert_1=[PK_1,"google.com",\sigma_1]$

$C = Enc(PK_1,K)$

PK_1,SK_1

$Dec(SK_1,C)$

K

VK^* correct⇒ PK_1 correct⇒ Person-in-the-Middle defeated!

Authenticated Key Exchange with Certs

CA's verification key $\mathtt{VK}^*$

(Pick random AES key $\mathtt{K}$)

$\mathtt{PK}_1, \mathtt{SK}_1$

$\mathtt{ClientHello}$ → $\mathtt{ClientHello}$

$\mathtt{cert}_1 = [\mathtt{PK}', \text{"google.com"}, ?]$ ← $\mathtt{cert}_1 = [\mathtt{PK}_1, \text{"google.com"}, \sigma_1]$

$\mathtt{K} \leftarrow \mathtt{Dec}(\mathtt{SK}_1, \mathtt{C})$

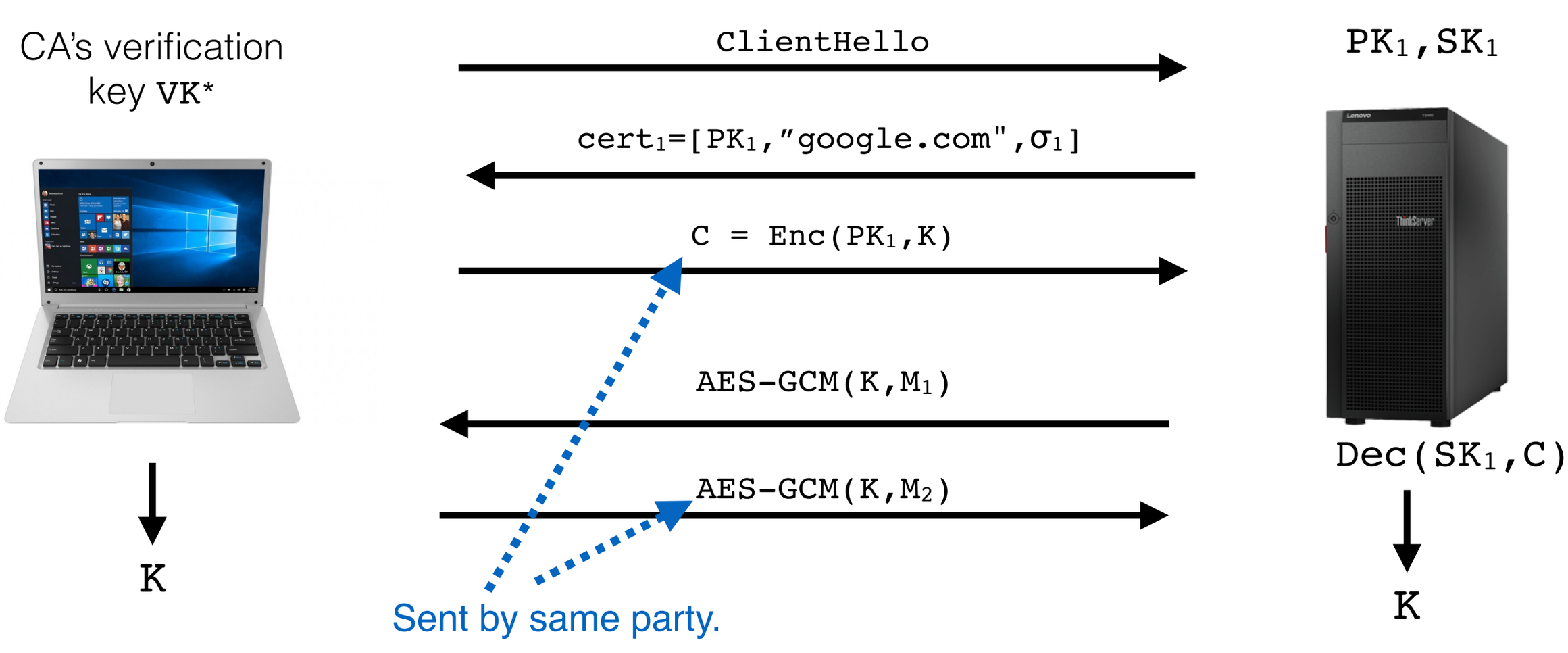
$\mathtt{K}$

$\mathtt{K}$

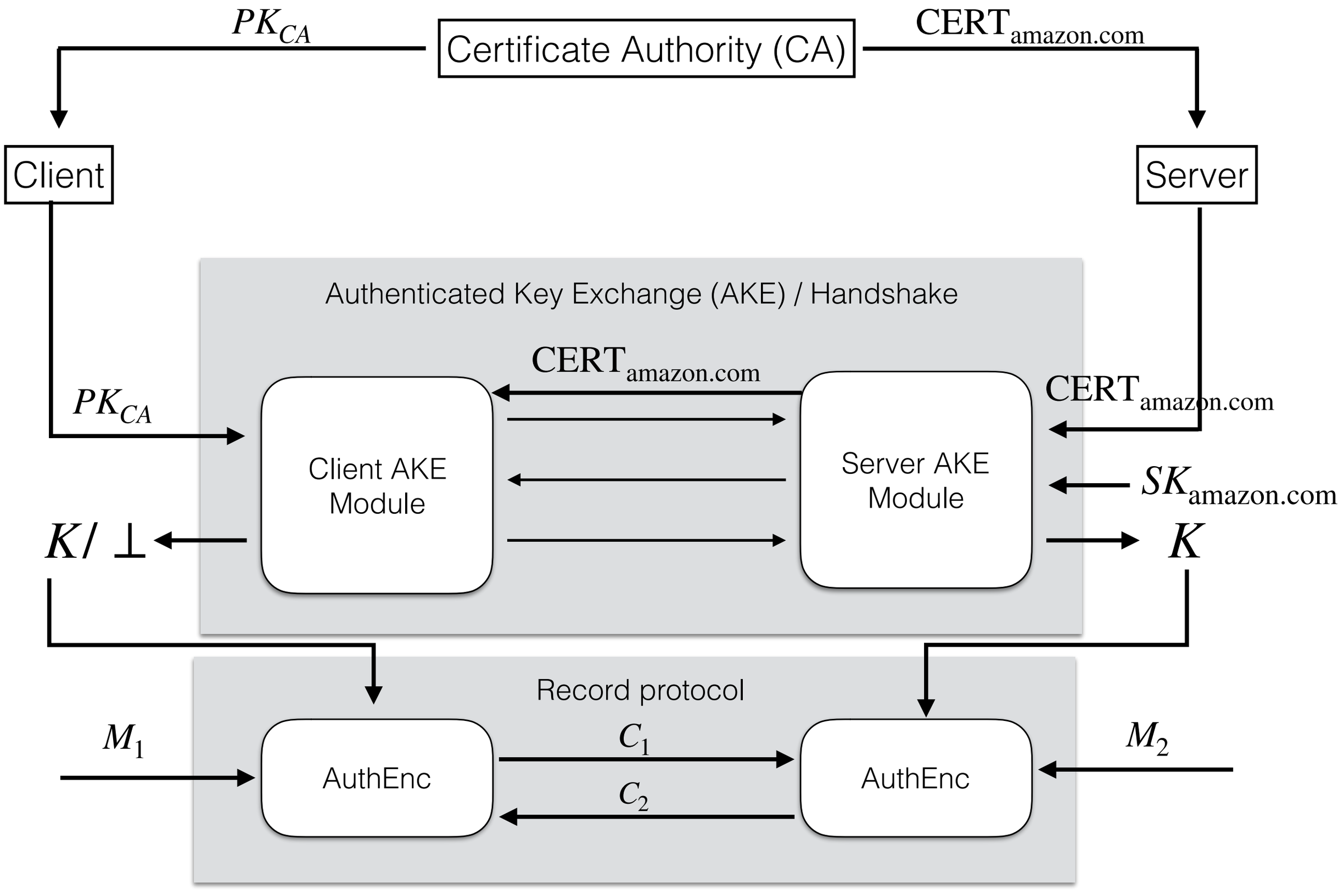
Adversary must forge signature, or trick CA into issuing cert.

Authenticated Key Exchange Notes

- Authentication is “unilateral” or “one-sided”
  - You are convinced you’re talking to **`google.com`**, but **`google.com`** has no idea who they are talking to.
  - However **`google.com`** knows they are continuing to talk to whoever sent **`C`**
  - You convince **`google.com`** of your identity using a password, not TLS.



Looking back: TLS



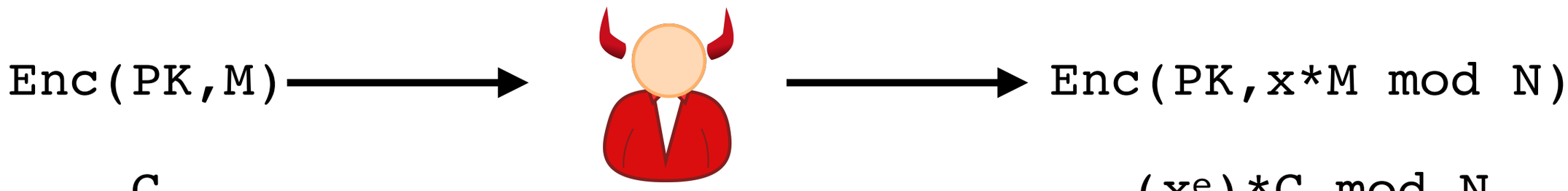
Questions to address (later in quarter):

- Certificate ecosystem
- Certificate details
- TLS details (negotiating which crypto to use…)
- Privacy: What does TLS hide about your browsing, and from whom?
- Authentication: How should users be authenticated?
- Security of protocols built on top of TLS
- Software vulnerabilities (other than the crypto bugs we've seen)

Switching Gears: A Sampling of “Advanced” Crypto

1. Homomorphic Encryption
2. Zero-Knowledge Proofs

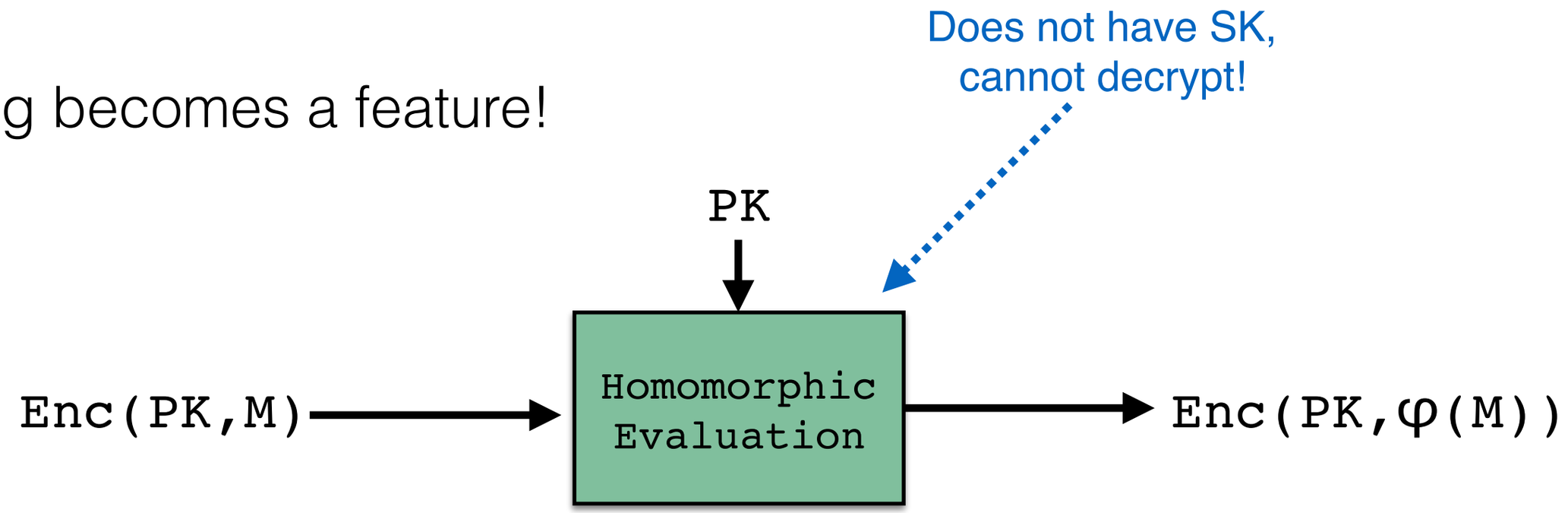
Malleable Encryption



- Malleability is usually a bad thing for Plain RSA Enc/Signatures
- Allows adversaries to predictably change plaintexts without permission
- See assignment 2

Homomorphic Encryption = Very Malleable Encryption

- A bug becomes a feature!



- RSA is homomorphic for multiplication by some fixed **x**:

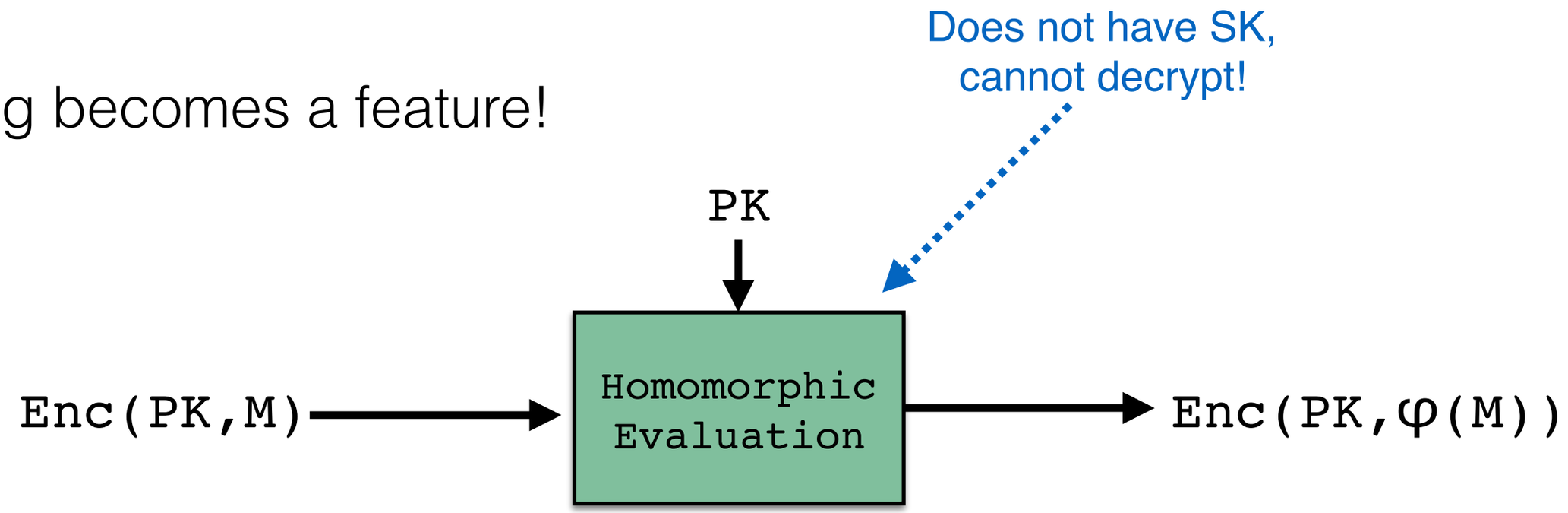
$$\varphi_x(M) = (x*M \bmod N)$$

- RSA does not appear to homomorphic for *addition* by some fixed **x**:

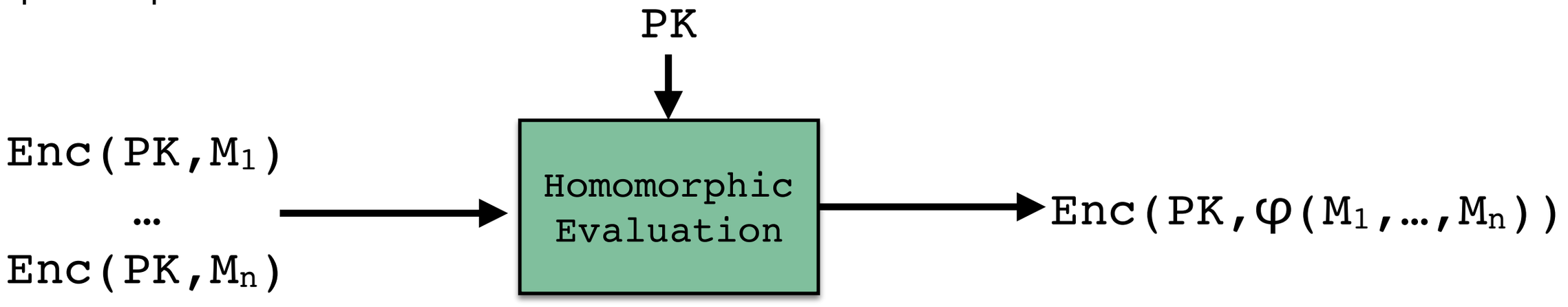
$$\varphi_x(M) = (x+M \bmod N)???$$

Homomorphic Encryption = Very Malleable Encryption

- A bug becomes a feature!



- Multiple-ciphertext version:



Homomorphic Encryption: The Grand Vision (1978)

ON DATA BANKS AND PRIVACY HOMOMORPHISMS

Ronald L. Rivest
Len Adleman
Michael L. Dertouzos

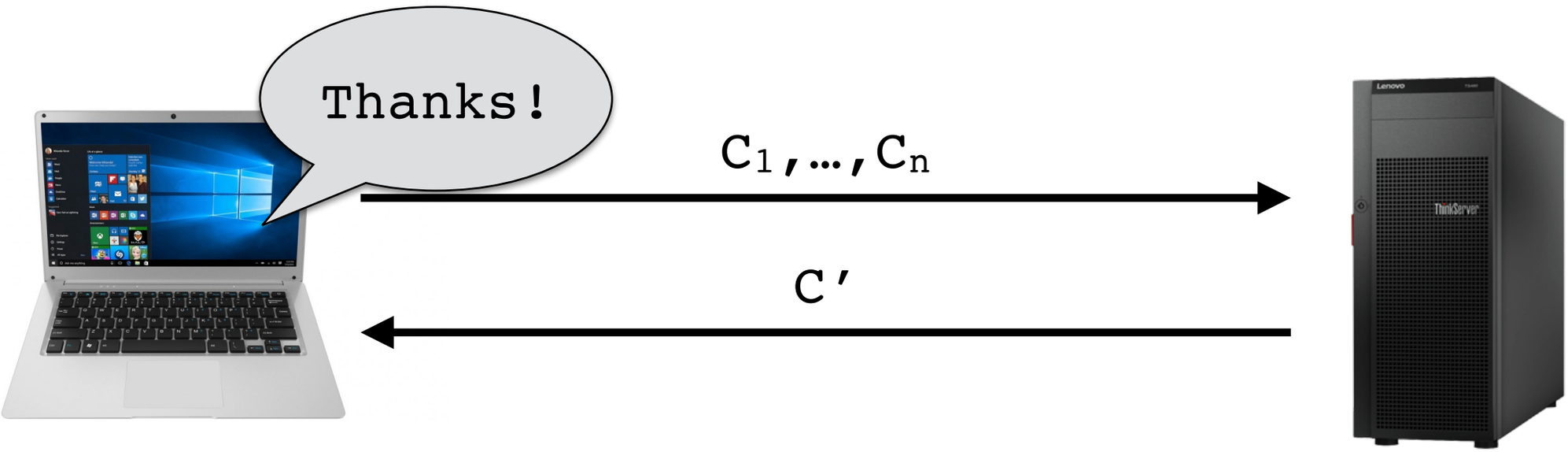
Massachusetts Institute of Technology
Cambridge, Massachusetts

I. INTRODUCTION

Encryption is a well-known technique for preserving the privacy of sensitive information. One of the basic, apparently inherent, limitations of this technique is that an information system working with encrypted data can at most store or retrieve the data for the user; any more complicated operations seem to require that the data be decrypted before being operated on.

Homomorphic Encryption: The Grand Vision (1978)

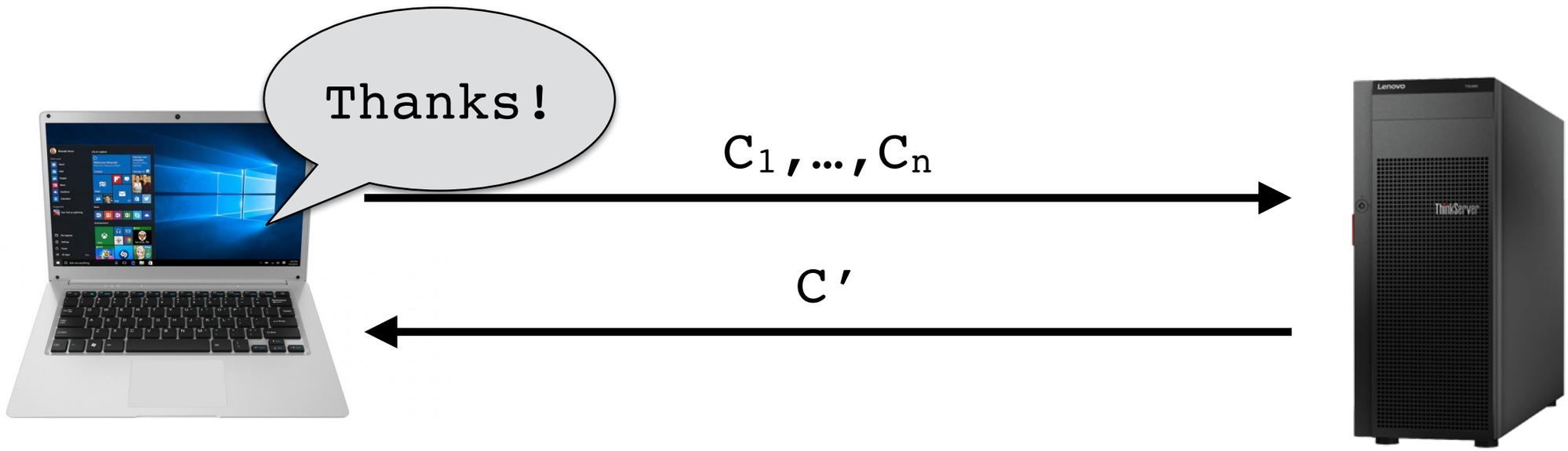
- Suppose **Enc** is homomorphic for φ using **HomEval**



- Client learns φ applied to its own data $M_1,\ldots,M_n$
- Client does not learn φ
- Server does not learn $M_1,\ldots,M_n$

Homomorphic Encryption: The Grand Vision (1978)

- Suppose **`Enc`** is homomorphic for φ using **`HomEval`**



$C_i \leftarrow Enc(PK, M_i)$

$\varphi(M_1,\ldots,M_n) \leftarrow Dec(SK, C'$ … $,C_n))$

- Client learns φ applie
- Client does not learn
- Server does not learn $M_1,\ldots,M_n$

What if **`Enc`** was homomorphic for **every** φ?

- Train private machine-learning models
- Run expensive simulations
- Query databases without decrypting

For which φ can we build homomorphic encryption?

- RSA (’78): **φ** = multiplication mod N of plaintexts and/or constants
- Paillier (’99): **φ** = *addition mod N* of plaintexts and/or constants
- …

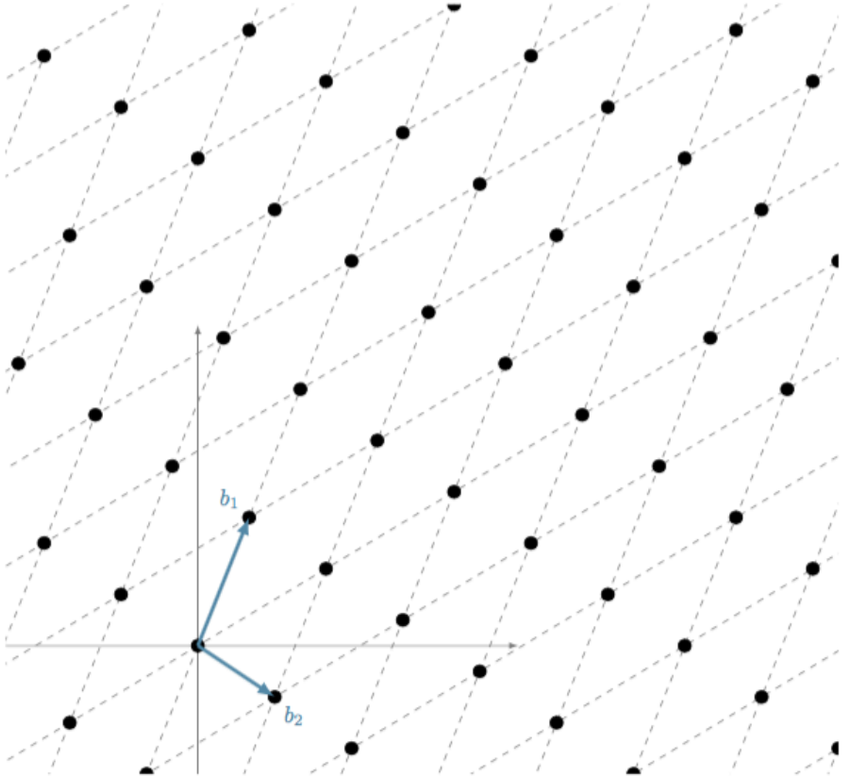
Observation: If an encryption is homomorphic for both additions and multiplications mod N, then it is homomorphic for any **φ**!

- BGN (’06): **φ** = many additions but only one multiplication
- …
- Gentry (’09): Any **φ**! Via new techniques.

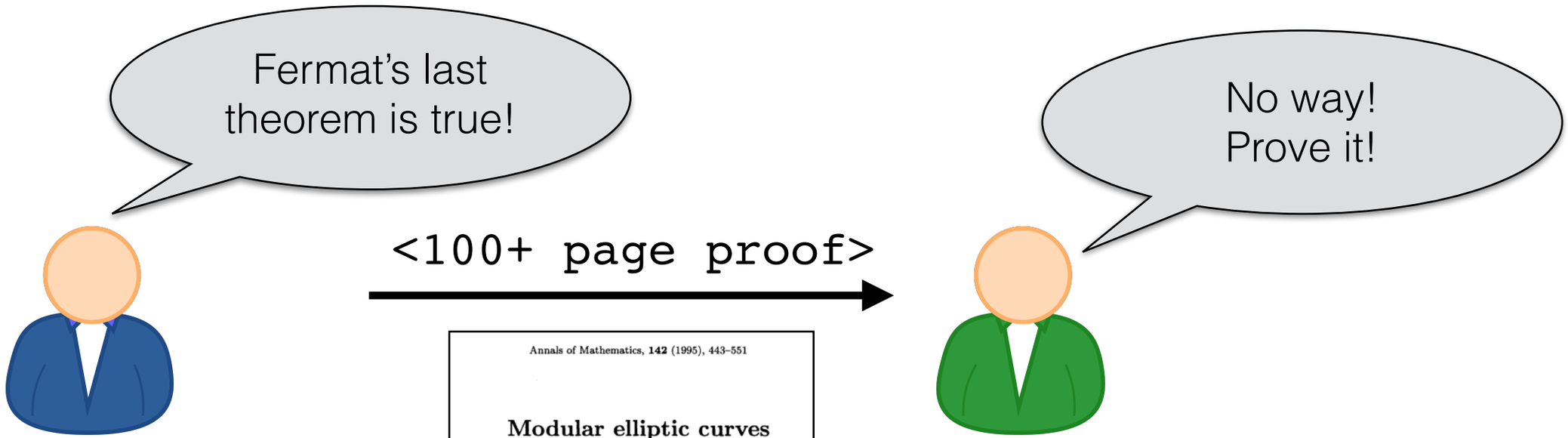
Homomorphic Encryption (Gentry'09)

- Based on different math (not RSA/Diffie-Hellman)
- Uses *lattices*, i.e. high-dimension integer grids

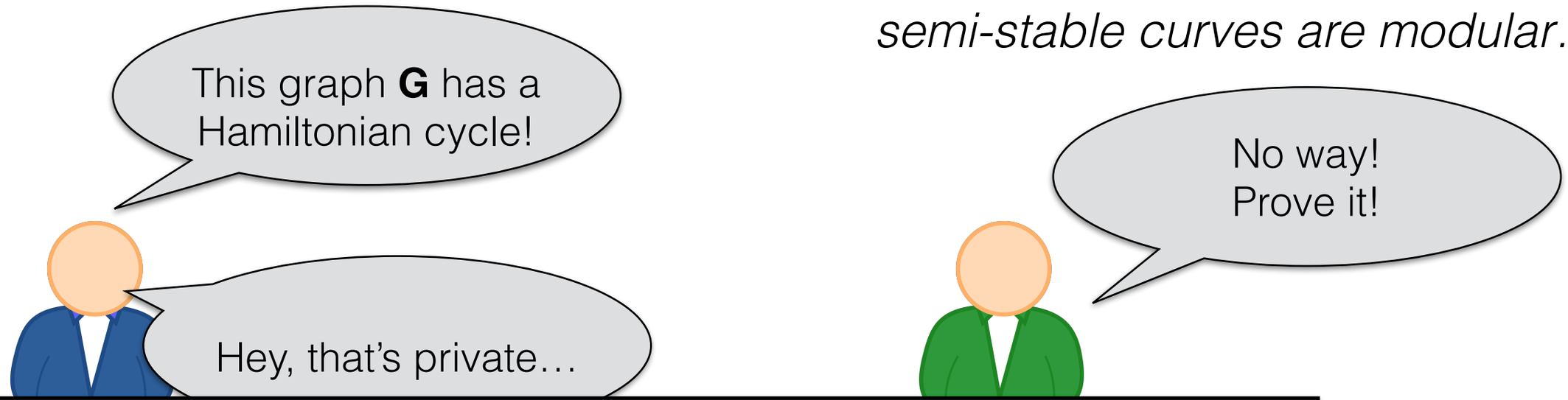
- Original construction was too slow
- Tons of research on making it faster
- So far, essentially no deployments…



Switching gears: Mathematical Proofs



- Convinced theorem is true
- Learns *<u>why</u> it's true (i.e. because all semi-stable curves are modular…)*

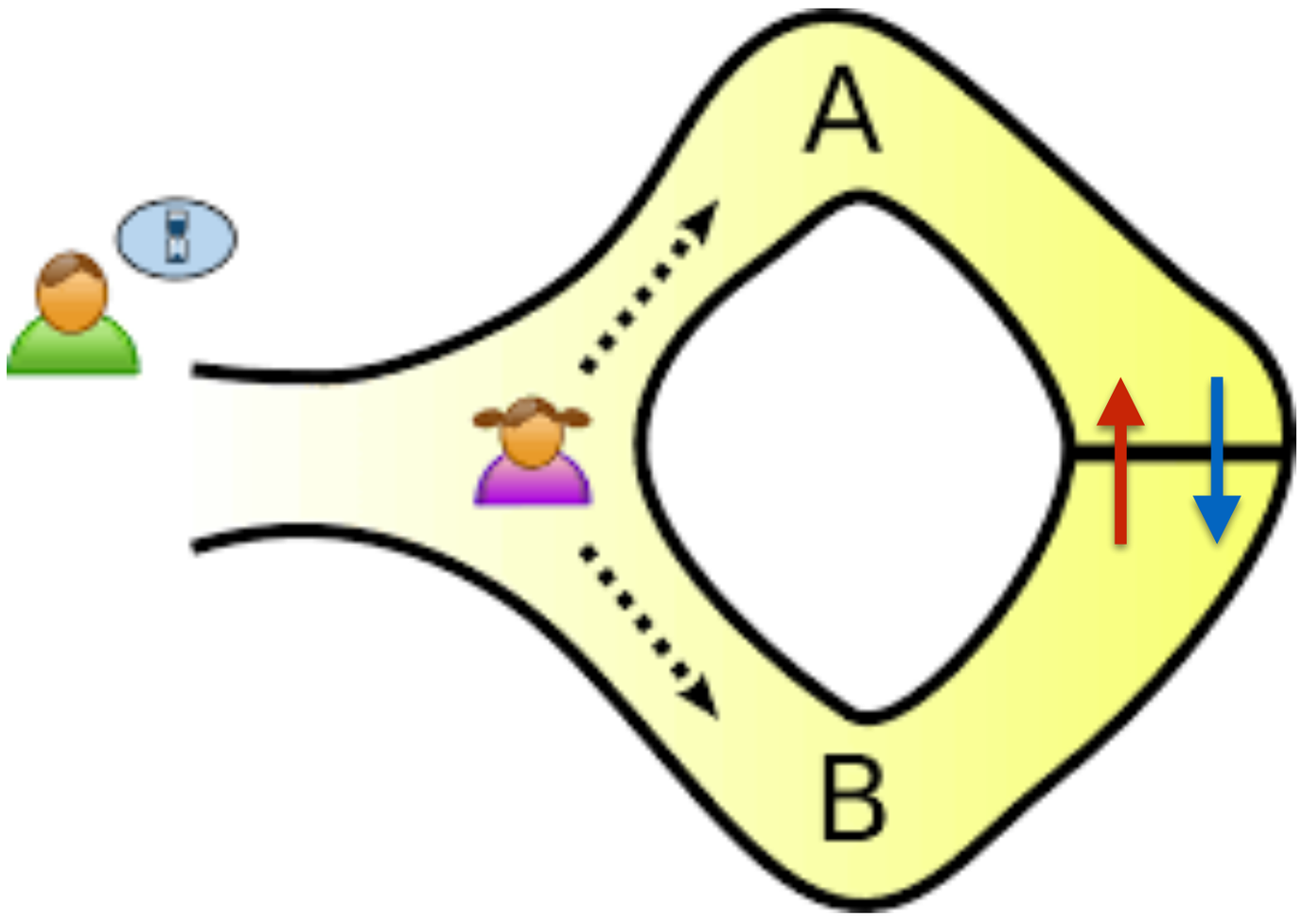


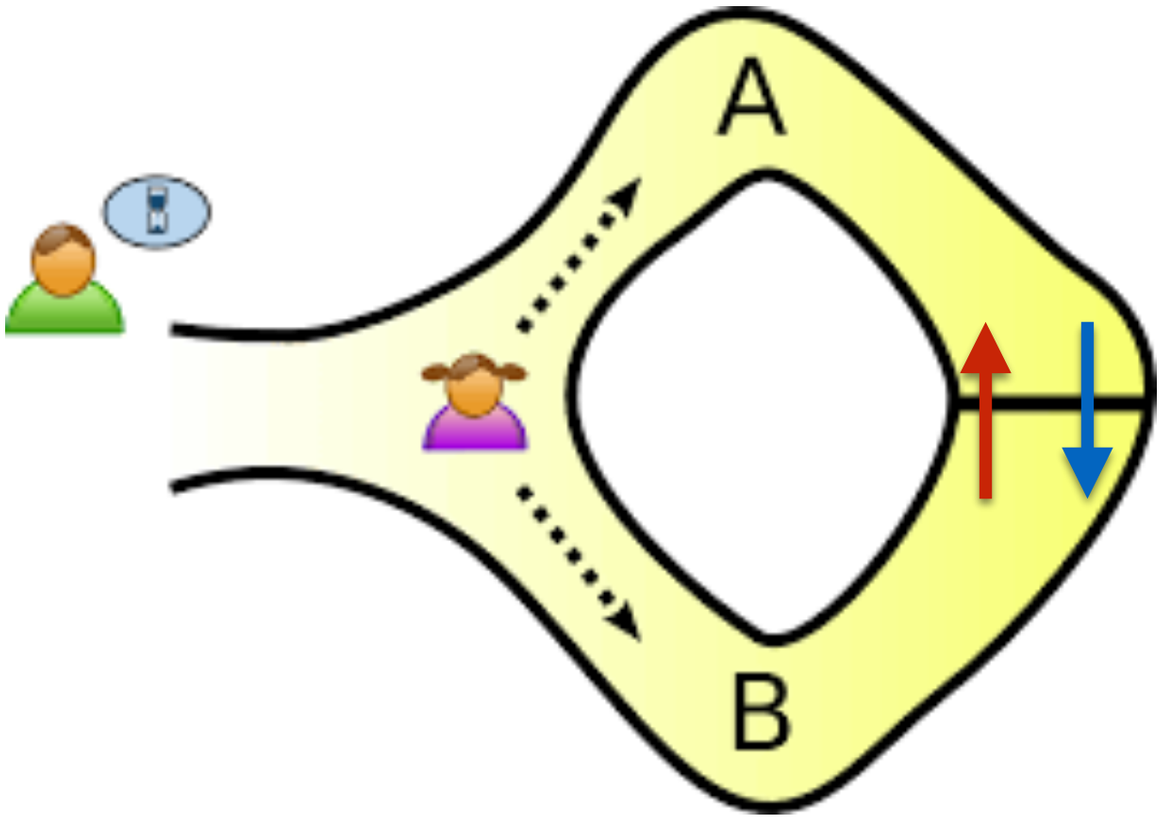
<u>Question</u>: Can one prove something is true…
…without revealing anything about why?

Zero-Knowledge Proofs (Goldwasser,Micali,Rackoff'85)



- Prover claims: There is a one-way door that opens between A and B
- Wants to hide: Which direction the door opens (A→B vs B→A)





Protocol:

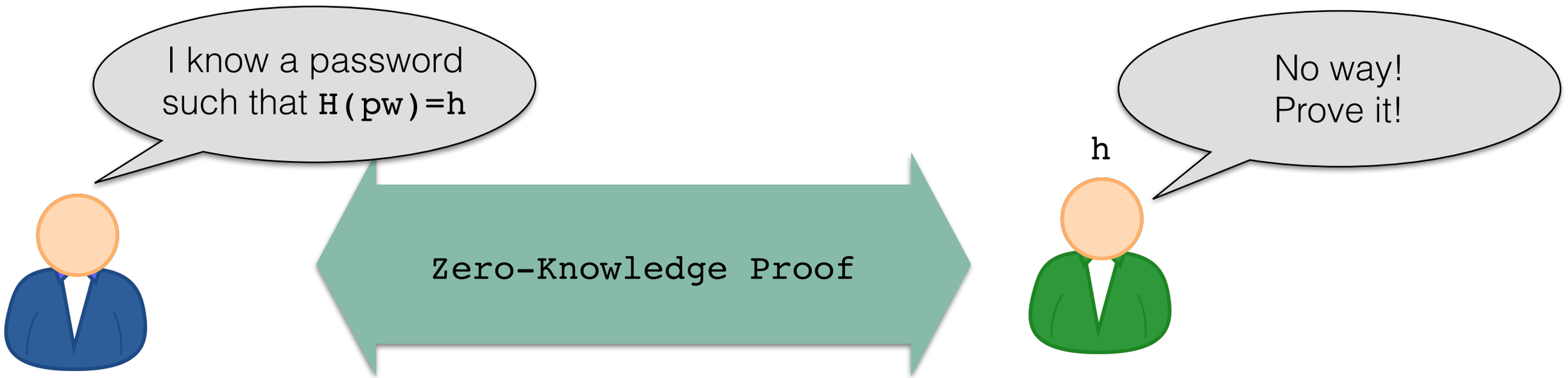
1. Prover walks into cave without Verifier watching
2. Verifier flips a coin and asks Prover to come out A or B side
3. Prove comes out that side, using door if necessary
4. Repeat 100 times. If prover is ever caught lying, **`REJECT`**.

Soundness: If there is (in fact) no door, then Prover only has $1/2^{100}$ chance to cheat.

Zero-knowledge: Even if Verifier tries to cheat, it won't learn anything about which way the door opens.

- Key insights:
  - Interaction
  - Randomness

Zero-Knowledge Proofs in Crypto



- Prover is convinced other person knows **pw**
- But everything else about **pw** is totally hidden

Theorem: Any provable mathematical statement has a zero-knowledge proof.

The End