Cryptography Part 2

CMSC 23200, Winter 2023, Lecture 5

Grant Ho & Blase Ur

University of Chicago

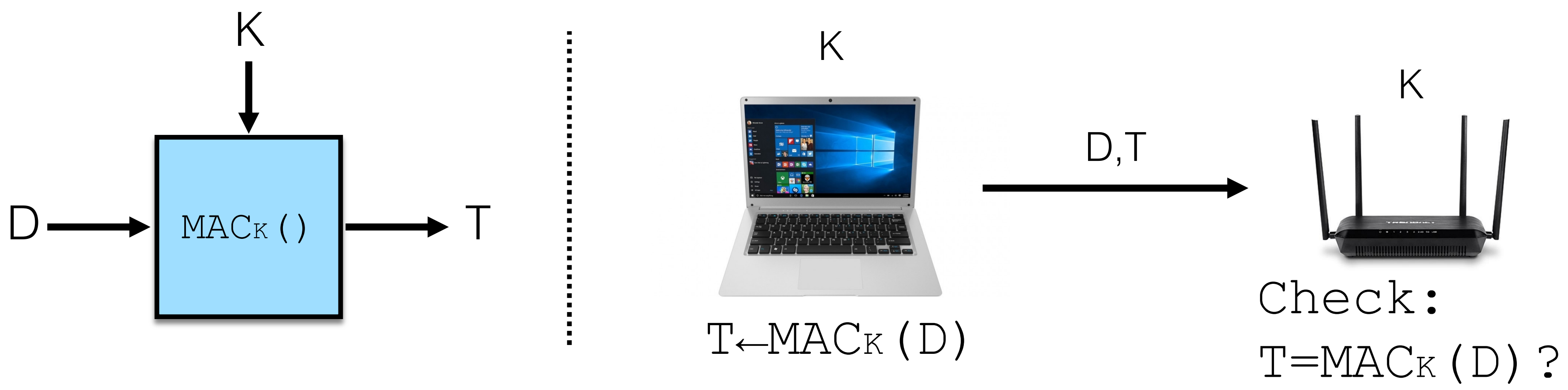
Outline: Crypto Part 2

- **Symmetric Key Cryptography**
  - Hash functions and MACs
  - Authenticated Encryption (and Block Ciphers)
- **Asymmetric (Public) Key Cryptography**
  - Public-Key Encryption
  - Digital Signatures
- **Hybrid Encryption:** Building secure channels from scratch*

Recall: Integrity (Message Authentication Codes)

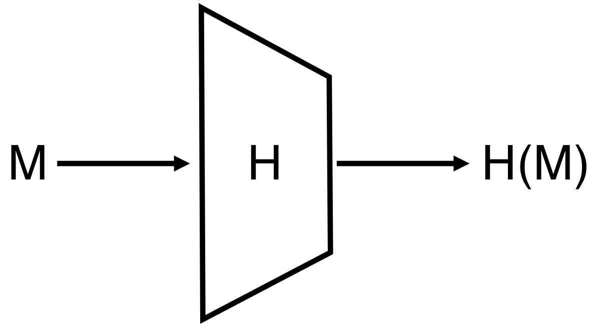
Provide integrity by attaching a MAC (tag T) to each message (D), where the tag is:

1) Short string that validates the message D
2) Unforgeable (can't create) without knowing secret key K



Building Block: Hash Functions

Definition: A <u>hash function</u> is a deterministic function H(…) that maps arbitrary strings to fixed-length outputs.



Properties of a ***secure*** hash function:

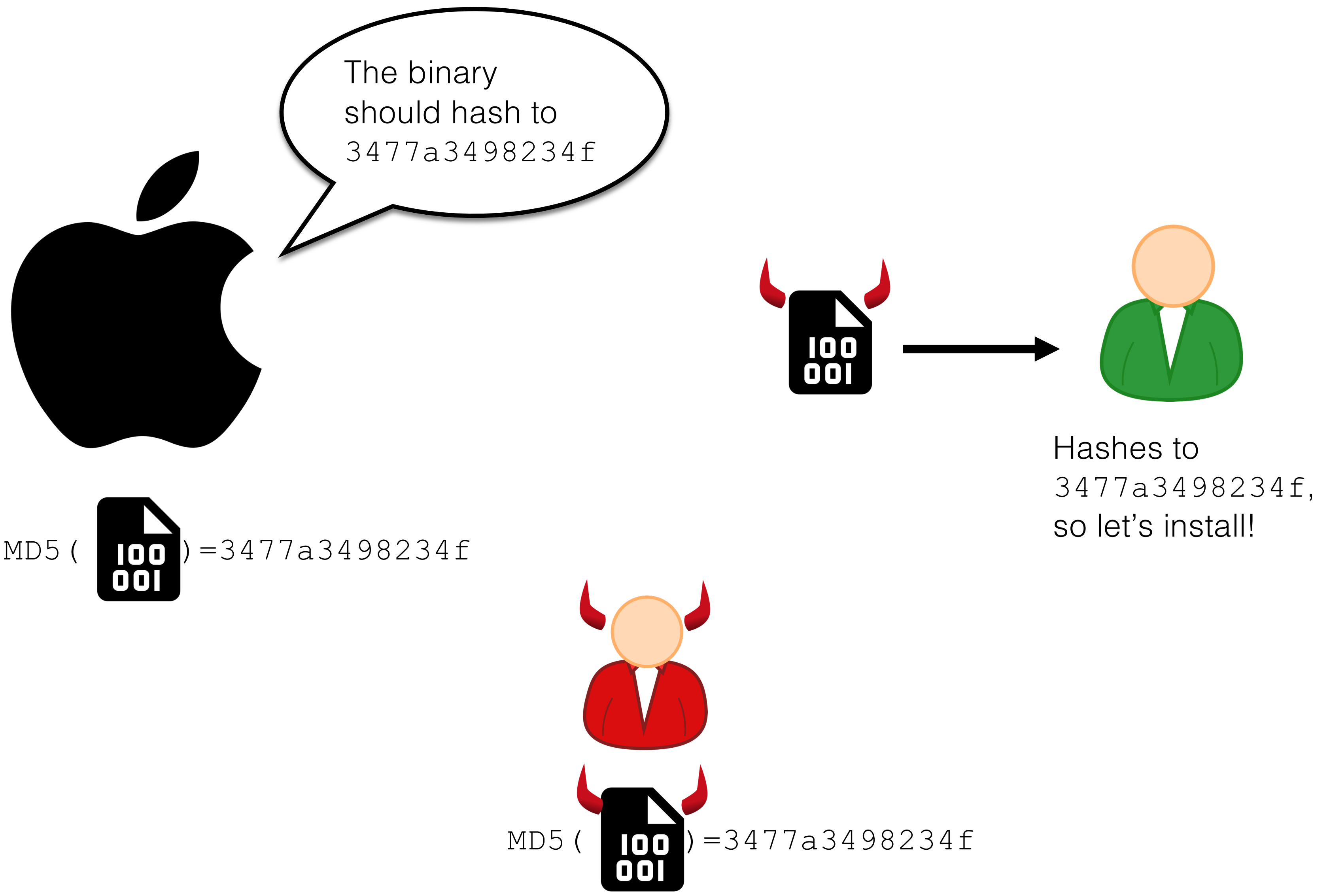
1. One-way function: given H(M), can't find M

2. Collision resistance: can't find M != M' such that H(M) = H(M')

3. Second-preimage resistance: given H(M), can't find M' s.t. H(M') = H(M)

- Note: Very different from hashes used in data structures!

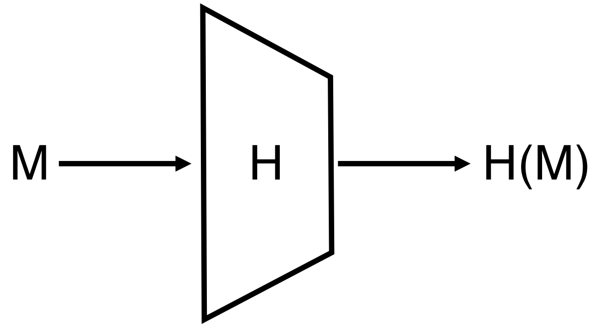
Why are hash collisions bad?

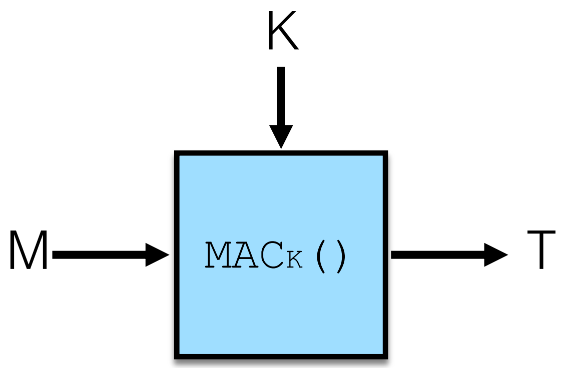


Practical Hash Functions

Name	Year	Output Len (bits)	Broken?
MD5	1993	128	Super-duper broken
SHA-1	1994	160	Yes
SHA-2 (SHA-256)	1999	256	No
SHA-2 (SHA-512)	2009	512	No
SHA-3	2019	>=224	No

Hash Functions are **not** MACs





Both functions map long inputs to short outputs… but hash func's do not use a key:
Attackers can compute hash of any message they want (not unforgeable)

Intuition: a MAC is like a hash function, that only the someone w/ the key can evaluate.

Building MACs from Hash Functions

Goal: Build a secure MAC out of a good hash function.

Construction: $MAC(K, D) = H(K \| D)$ **Warning: Broken**

- Totally insecure if H = MD5, SHA1, SHA-256, SHA-512

Secure MAC: Use standard **HMAC** function

$$MAC(K, D) = H(K \oplus opad \| H(K \oplus ipad \| D))$$

NEVER Design your own crypto algorithms, always use standard libraries!

Length Extension Attack on Insecure MACs

Construction: MAC(K, D) = H(K || D) **Warning: Broken**

Adversary goal: Find new message D' and a valid tag T' for D'

In other words: Given T=H(K || D), find T'=H(K || D') without knowing K.

- Attack: Can craft D' = D || XYZ, with some string XYZ that consists of (1) substr that attacker can freely choose and (2) substr to make attack work

In Assignment 3: Break this construction!

Outline: Crypto Part 2

- **Symmetric Key Cryptography**
  - Hash functions and MACs
  - Authenticated Encryption (and Block Ciphers)
- **Asymmetric (Public) Key Cryptography**
  - Public-Key Encryption
  - Digital Signatures
- **Hybrid Encryption:** Building secure channels from scratch*

Four Cryptography Problems / Tools

Security Goal	Confidentiality	Authenticity/Integrity
	Symmetric Encryption	Message Authentication Code (MAC)
	Security: Ciphertext reveals ***nothing*** about plaintext message	Security: Tag for new msg is impossible to compute without secret key

Authenticated Encryption

Authenticated Encryption algorithms provide both **confidentiality** and **integrity**.

- One approach: Built using a good stream cipher and a MAC.
  - Ex: Salsa20 with HMAC-SHA2

- Best solution: Use ready-made Authenticated Encryption
  - Ex: AES-GCM is the standard

Brief Detour: AES & Block Ciphers

Blockciphers: common crypto building block for solving many problems.

Informal definition: A blockcipher is essentially a substitution cipher with a very large alphabet and a very compact key.

Typical parameters:
Alphabet = $\{0,1\}^{128}$
Key length = 16 bytes.

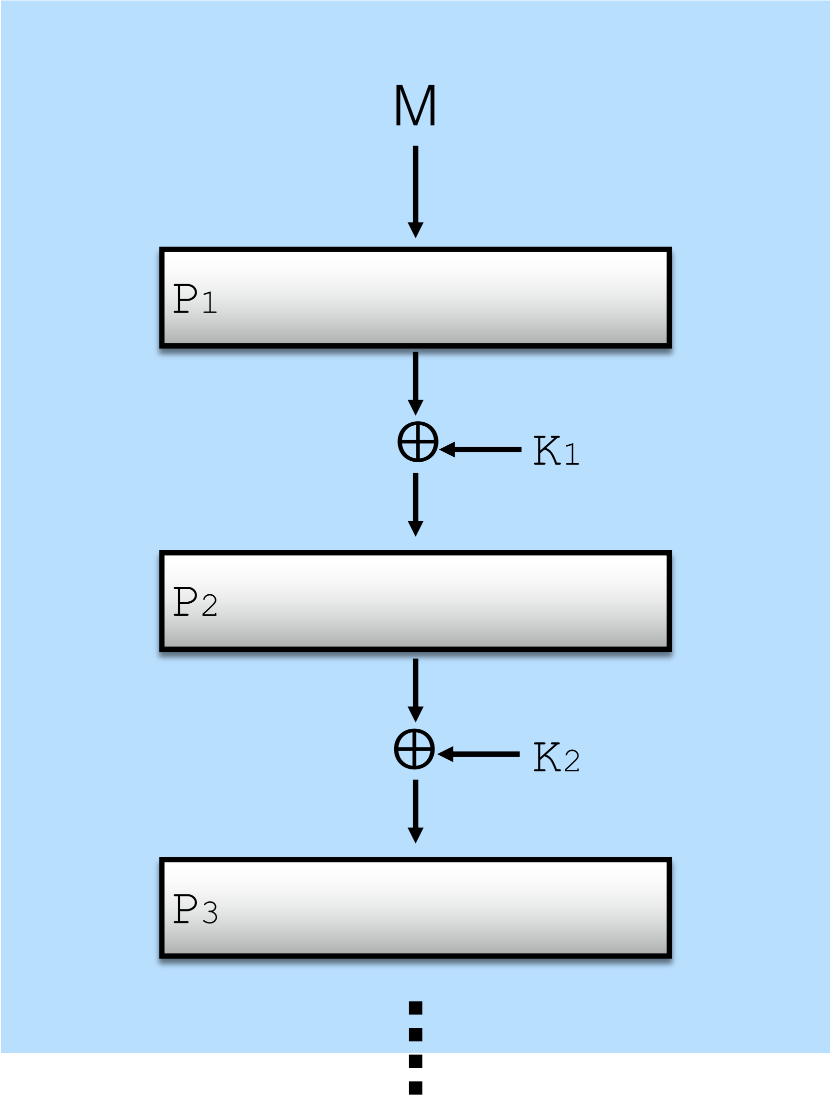
Can build many higher-level protocols from a good blockcipher.

Advanced Encryption Standard (AES)

- NIST ran competition to develop standard encryption algorithms in 1997
- Several submissions, *Rijndael* chosen and standardized

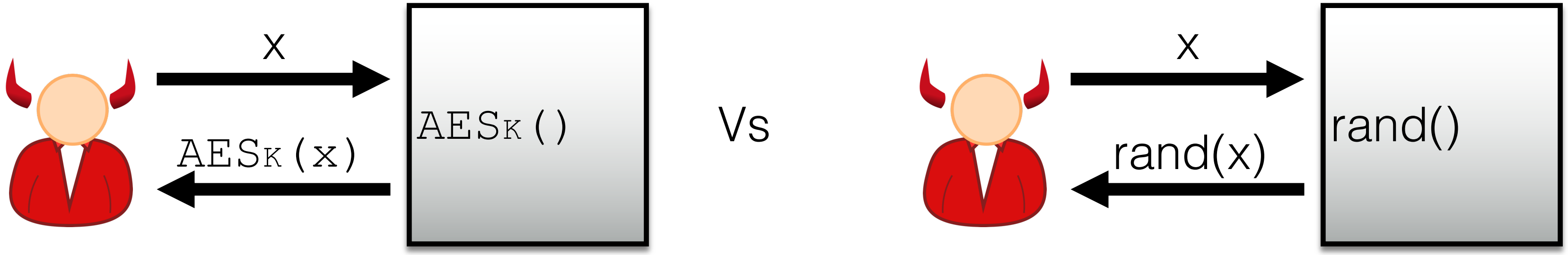
Rijmen and Daemen
- Block length n = 128
- Key length k = 128,192,256
- 10 rounds of "substitution-permutation network"

- Break msg M into blocks and encrypt each block



Blockcipher Security (Confidentiality)

- AES is thought to be a good “Pseudorandom Permutation”



- Outputs all look random and independent, even when inputs are maliciously controlled.
- Formal definition in CS284.

Advanced Encryption Standard (AES)

- AES is now the gold standard blockcipher
    - Very fast; Intel & AMD CPU chips have built-in AES instructions

- AES has different *modes* of operation
    - Some common modes: ECB, CTR, CBC, GCM
    - ECB : do not use – insecure!!
    - CTR & CBC do not provide integrity
    - GCM: authenticated encryption (both conf & integrity)

Outline: Crypto Part 2

- **Symmetric Key Cryptography**
  - Hash functions and MACs
  - Authenticated Encryption (and Block Ciphers)
- **Asymmetric (Public) Key Cryptography**
  - Public-Key Encryption
  - Digital Signatures
- **Hybrid Encryption:** Building secure channels from scratch*

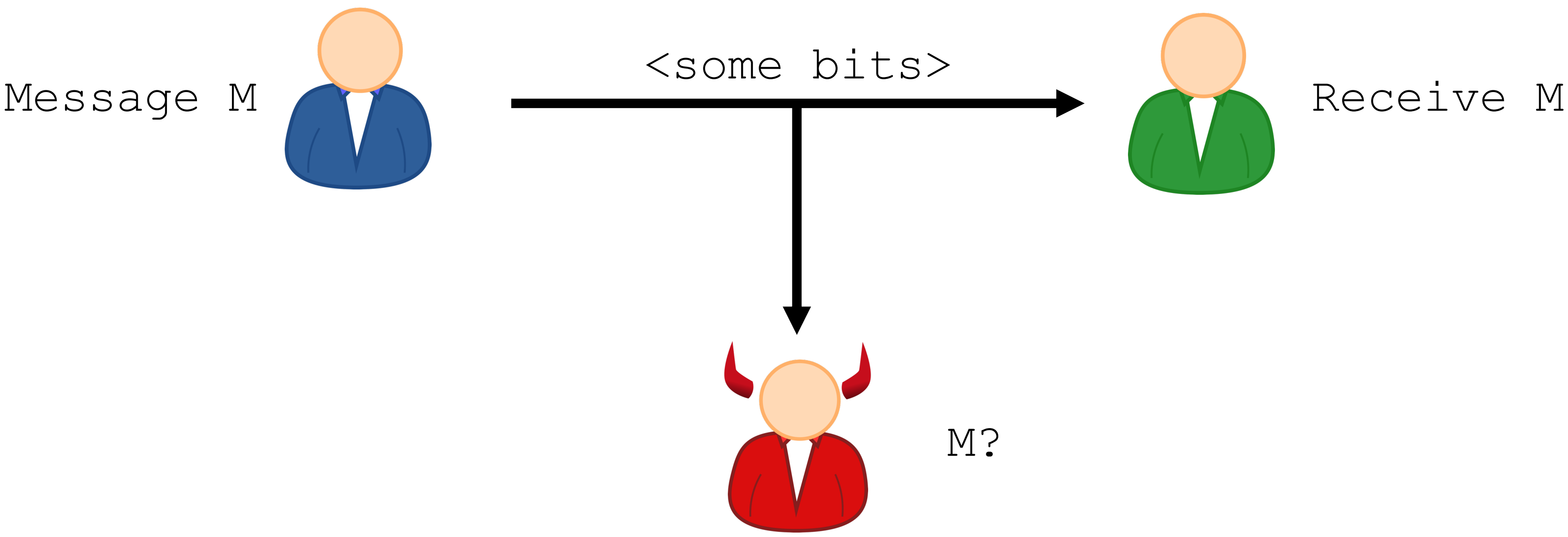
Why do we need Public-Key Cryptography?

Motivation: If two people do <u>not</u> have a pre-shared secret key, can they send private messages in the presence of an attacker?

Pre-shared key? \ Security Goal	Confidentiality	Authenticity/ Integrity
Yes ("Symmetric")	Symmetric Encryption	Message Authentication Code (MAC)
No ("Asymmetric")	Public-Key Encryption	Digital Signatures

Why do we need Public-Key Cryptography?

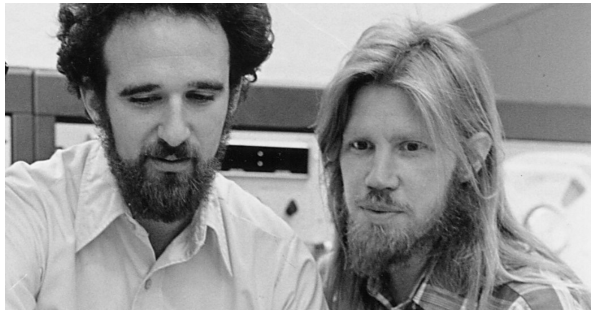
Motivation: If two people do <u>not</u> have a pre-shared secret key, can they send private messages in the presence of an attacker?



Formally impossible (in some sense):
No difference between receiver and adversary.

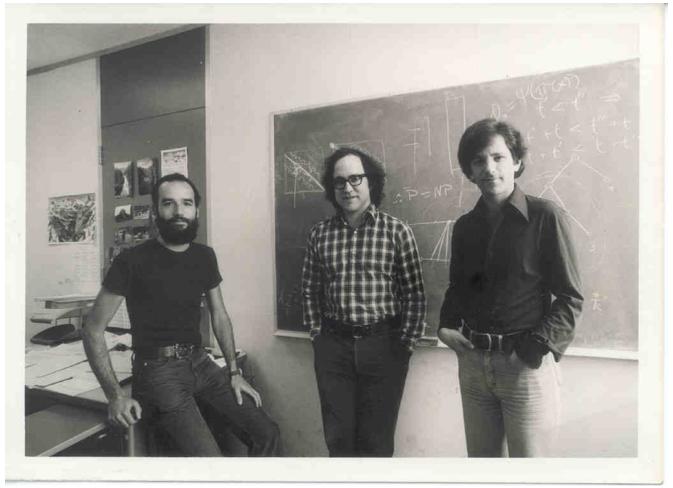
Why do we need Public-Key Cryptography?

Motivation: If two people do not have a pre-shared secret key, can they send private messages in the presence of an attacker?



Diffie and Hellman
in 1976: **Yes!**

Turing Award, 2015



Rivest, Shamir, Adleman
in 1978: **Yes, differently!**

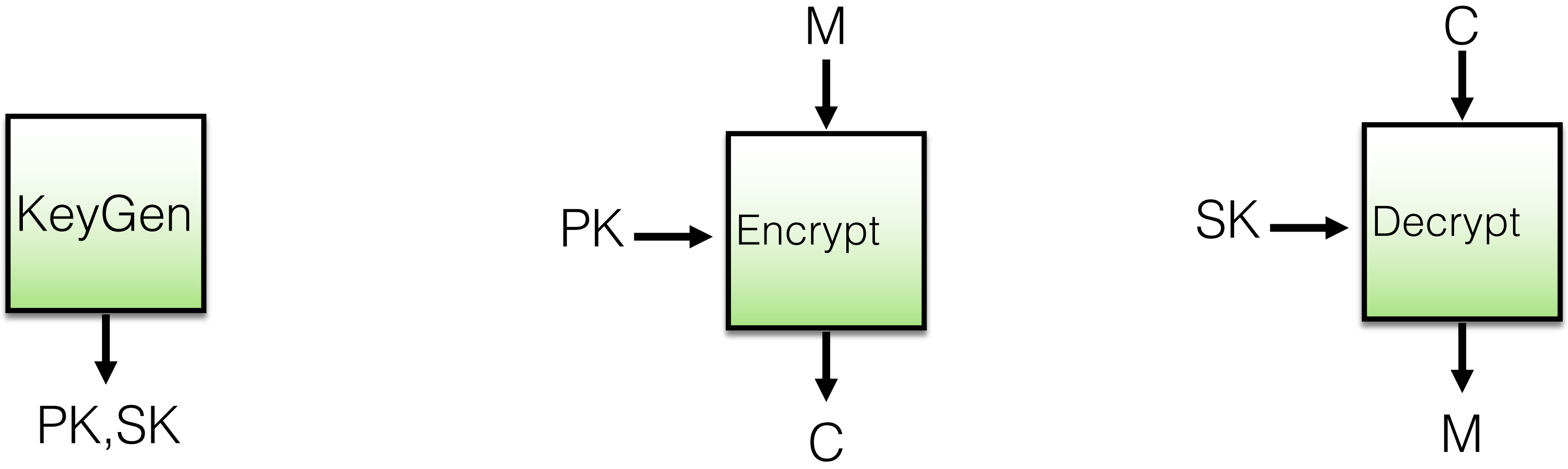
Turing Award, 2002

Cocks, Ellis, Williamson
in 1969, at GCHQ:
Yes…

Public-Key Encryption (Confidentiality)

A <u>public-key encryption scheme</u> consists of three algorithms:

`KeyGen`, **`Encrypt`**, and **`Decrypt`**



<u>`KeyGen`</u>: Outputs two keys. PK published openly, and SK kept secret.

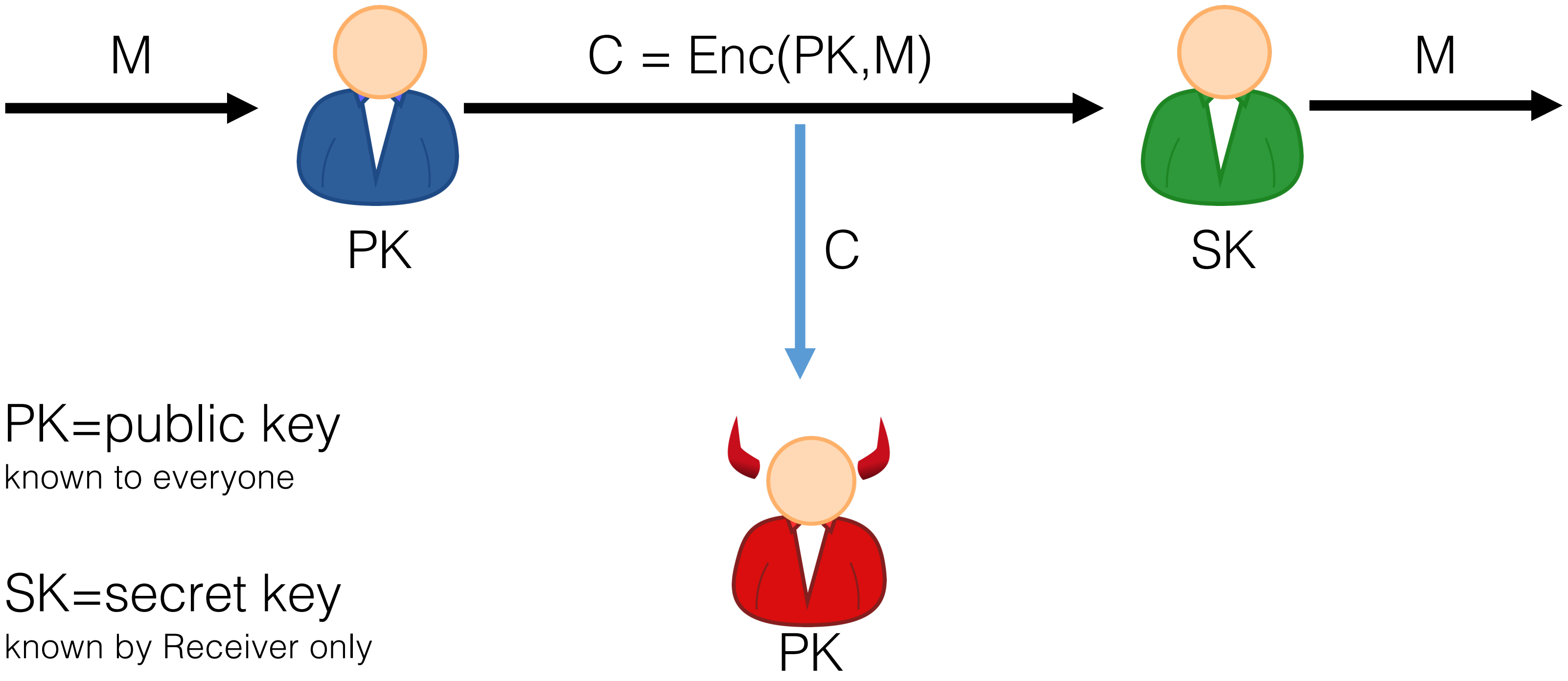
<u>`Encrypt(PK, M)`</u>: Uses PK and M to produce a ciphertext C.

<u>`Decrypt(SK, C)`</u>: Uses SK and C to recover M.

Public-Key Encryption

Goal: Passive Attacker, knows algorithm implementations (Enc, Dec) and PK, but the ciphertext C reveals nothing about the plaintext message M

- Attacker might also have partial knowledge, e.g., other (M*, C*) pairs
- Encryption (symmetric too) not even allowed to reveal if a message repeated!



PK=public key
known to everyone

SK=secret key
known by Receiver only

Public Key Encryption Schemes: RSA

Key Generation:

- Pick p and q be *large* random prime numbers (around 2^{1024})
- Compute $N \leftarrow pq$
- Set e to a default value ($e = 3$ and $e = 65537$ are common)
- Compute d such that $ed = 1 mod (p-1)(q-1)$
- Output:
  - Public key $pk = (N, e)$
  - Secret key $sk = (N, d)$

Example:

- $p = 5, q = 11, N = 55$
- $e = 3, d = 27$

Plain RSA Encryption

$PK = (N, e)$ $SK = (N, d)$ where $N = pq, ed = 1 mod(\phi(N))$

Note: Taking modular roots is believed to be computational hard

Encryption & Decryption:

$$\mathrm{Enc}((N, e), x) = x^e mod N$$

$$\mathrm{Dec}((N, d), y) = y^d mod N$$

Using number theory from CMSC 27100, can show:

$$\mathrm{Dec}(\mathrm{Enc}((N, e), x)) = (x^e)^d = x \; mod N$$

Never use directly as encryption!

Warning: Broken

Best Known Attack on RSA: Factoring

- Factoring N allows recovery of secret key… can compute $\phi(N) = (p-1)(q-1)$
- Challenges posted publicly by RSA Laboratories

Bit-length of N	Year
400	1993
478	1994
515	1999
768	2009
795	2019

- Recommended bit-length today: 2048 or greater
- Note that fast factoring algorithms force such a large key.
  - 512-bit N defeats naive factoring

Outline: Crypto Part 2

- **Symmetric Key Cryptography**
  - Hash functions and MACs
  - Authenticated Encryption (and Block Ciphers)

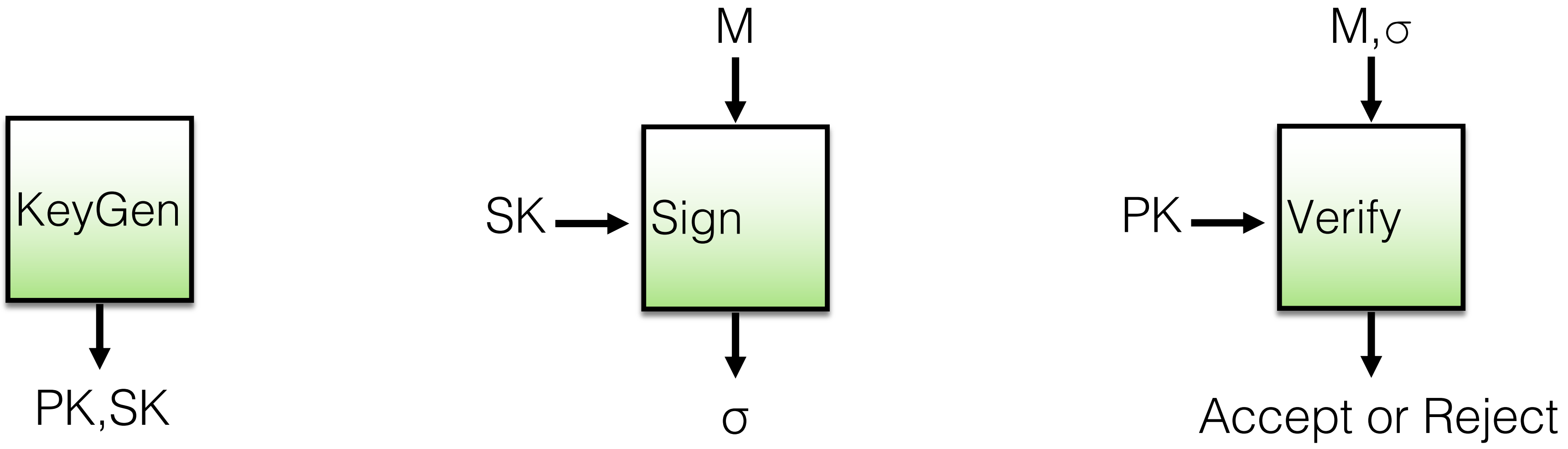
- **Asymmetric (Public) Key Cryptography**
  - Public-Key Encryption
  - Digital Signatures

- **Hybrid Encryption:** Building secure channels from scratch*

Digital Signatures Schemes (Integrity & Auth)

A digital signature scheme consists of three algorithms

`KeyGen`, **`Sign`**, and **`Verify`**

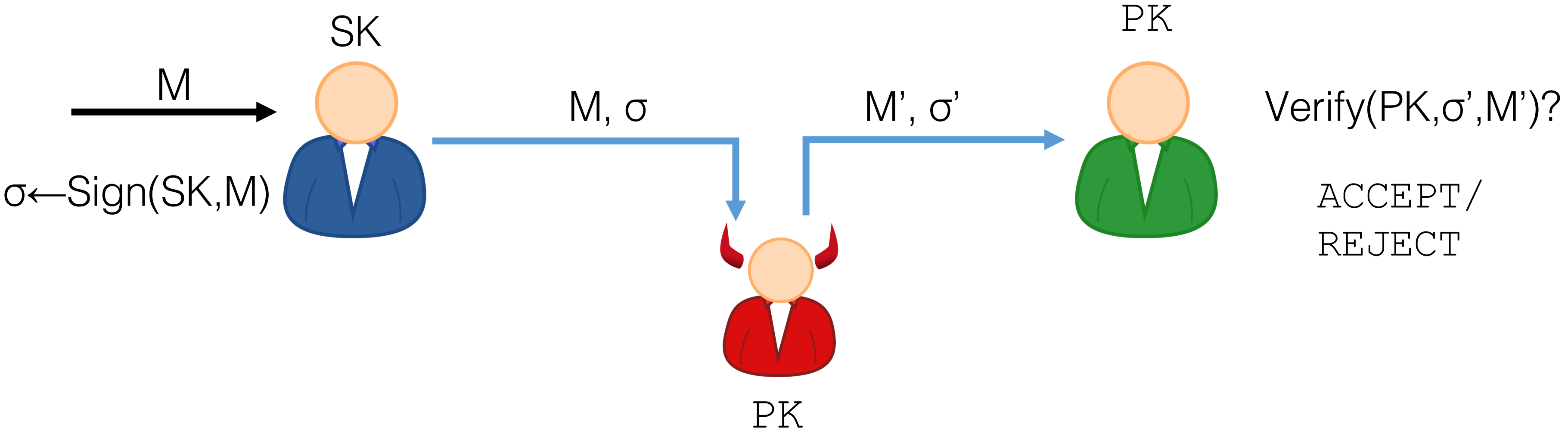


`KeyGen`: Outputs two keys. PK published openly, and SK kept secret.

`Sign`: Uses SK to produce a "signature" σ on M.

`Verify`: Uses PK to check if signature σ is valid for M.

Digital Signature Security Goal: Unforgeability



Scheme satisfies **unforgeability** if an Adversary (who knows PK) cannot to fool Bob into accepting (M', σ') that Alice has not sent.

“Plain” RSA Signature with No Encoding

Broken

KeyGen is same as regular RSA:

$PK = (N, e)$ $SK = (N, d)$ where $N = pq, ed = 1 mod \phi(N)$

$e = 3$ is common for fast verification.

$$\text{Sign}((N, d), M) = M^d mod N$$

$$\text{Verify}\big((N, e), M, \sigma\big): \sigma^e = M mod N?$$

"Plain" RSA Weaknesses

Broken

Assume $e=3$.

$$\text{Sign}((N,d),M) = M^d mod N \qquad \text{Verify}((N,3),M,\sigma): \sigma^3 = M mod N?$$

To forge a signature on message M': Find number σ' such that $(\sigma')^3 = M' \mod N$

Trivial Attack: Easy to forge signature for M'=1: Take σ'=1:

$$(\sigma'^3) = 1^3 = 1 = M' \mod N$$

Cube-M weakness: For any M' that is a perfect cube, it is easy to forge.
Attack: Signature σ'= $\sqrt[3]{M'}$, i.e. the usual cube root of M'

Example: To forge on $M'=8$, which is a perfect cube, set $\sigma'=2$.

$$(\sigma')^3 = 2^3 = 8 = M' \mod N$$

(Intuition: If cubing does not "wrap modulo N", then it is easy to un-do.)

More “Plain” RSA Weaknesses

Broken

Assume e=3.

$\text{Sign}((N,d),M) = M^d \mathit{modN}$ $\quad$ $\text{Verify}((N,3),M,\sigma)\colon \sigma^3 = M\mathit{modN}?$

To forge a signature on message M': Find number σ' such that $(\sigma')^3 = M' \bmod N$

***Malleability* weakness:** If σ is a valid signature for M, then it is easy to forge a signature for new msg $M' = (8M \bmod N)$,

Given (M,σ), compute forgery (M',σ') as

$$M' = (8*M \bmod N), \text{ and } \sigma' = (2*\sigma \bmod N)$$

This is a valid pair because: `Verify((N,3), M',σ')` checks:

$$(\sigma')^3 = (2*\sigma \bmod N)^3 = (2^3*\sigma^3 \bmod N) = (2^3*M \bmod N) = 8M \bmod N$$

$\sigma^3 = M \bmod N$ because σ is valid sig. on M

Secure RSA Signatures with Encodings

$PK = (N, e)$ $SK = (N, d)$ where $N = pq, ed = 1 mod \phi(N)$

$$\text{Sign}((N,d), M) = (\text{encode}(M))^d mod N$$

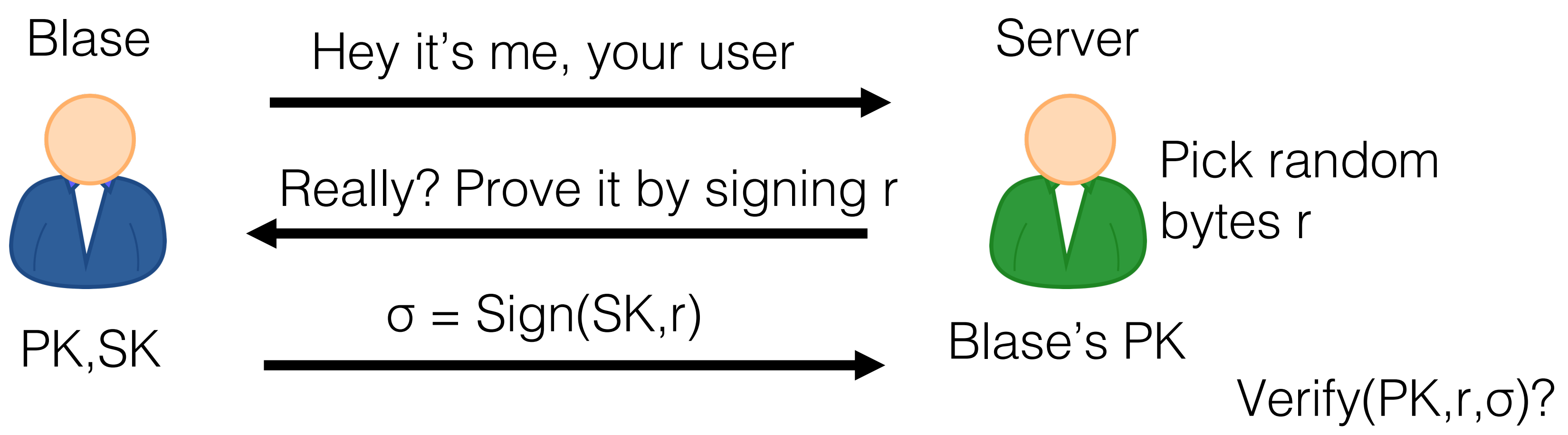
$$\text{Verify}\big((N,e), M, \sigma\big): \sigma^e = \text{encode}(M) mod N?$$

encode maps bit strings to numbers between 0 and N

```
Encoding must be chosen
with extreme care.
```

Broken

Authentication via Digital Signatures



- “Challenge – Response” Protocol
- This and similar ideas used in SSH, TLS, etc.

Digital Signature Summary

As with all crypto schemes:
do not build your own signature schemes!

- Plain RSA signatures are very broken!
- Several secure RSA options in widely deployed libraries available:
  - PKCS#1 v.1.5 is widely used, in TLS, and fine if implemented correctly
  - Full-Domain Hash and PSS should be preferred

- There are also other signature schemes that aren't based on RSA (e.g., DSA/ECDSA)

Outline: Crypto Part 2

- **Symmetric Key Cryptography**
  - Hash functions and MACs
  - Authenticated Encryption (and Block Ciphers)
- **Asymmetric (Public) Key Cryptography**
  - Public-Key Encryption
  - Digital Signatures
- **Hybrid Encryption:** Building secure channels from scratch*

Why not use asymmetric crypto for everything?

<u>Answer</u>

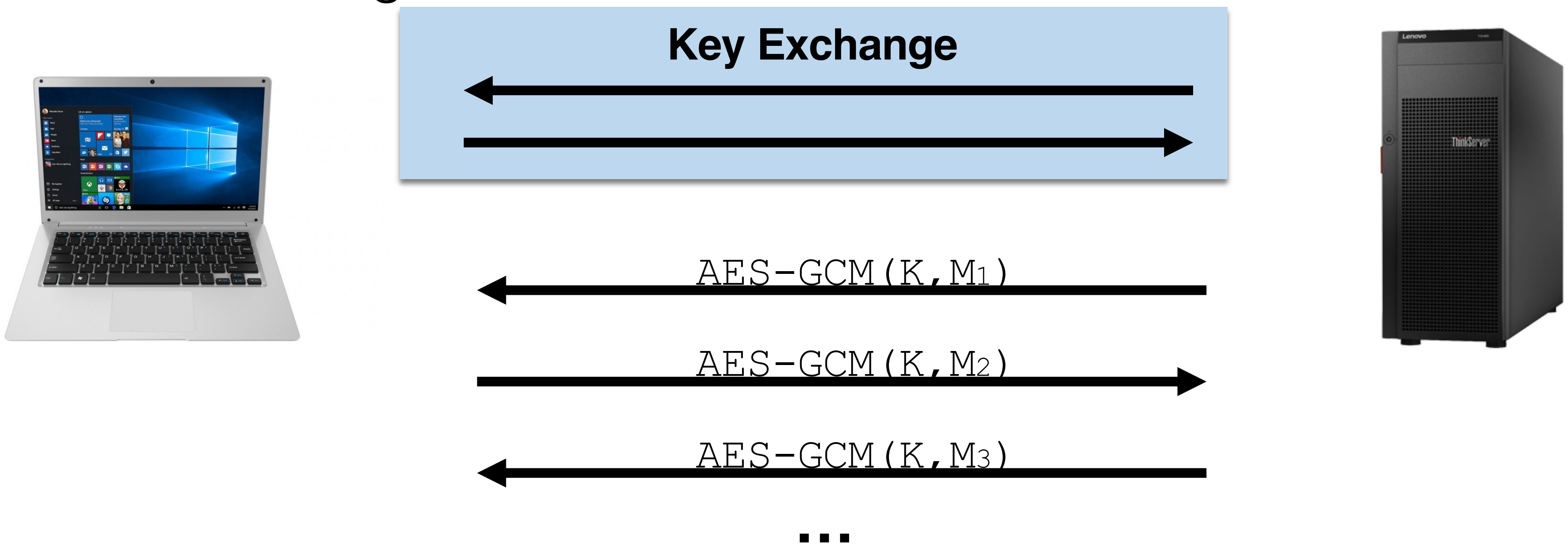
Symmetric key crypto algorithms are **MUCH** faster

Pre-shared key? \ Security Goal	**Confidentiality**	**Authenticity/Integrity**
Yes ("Symmetric")	Symmetric Encryption	Message Authentication Code (MAC)
No ("Asymmetric")	Public-Key Encryption	Digital Signatures

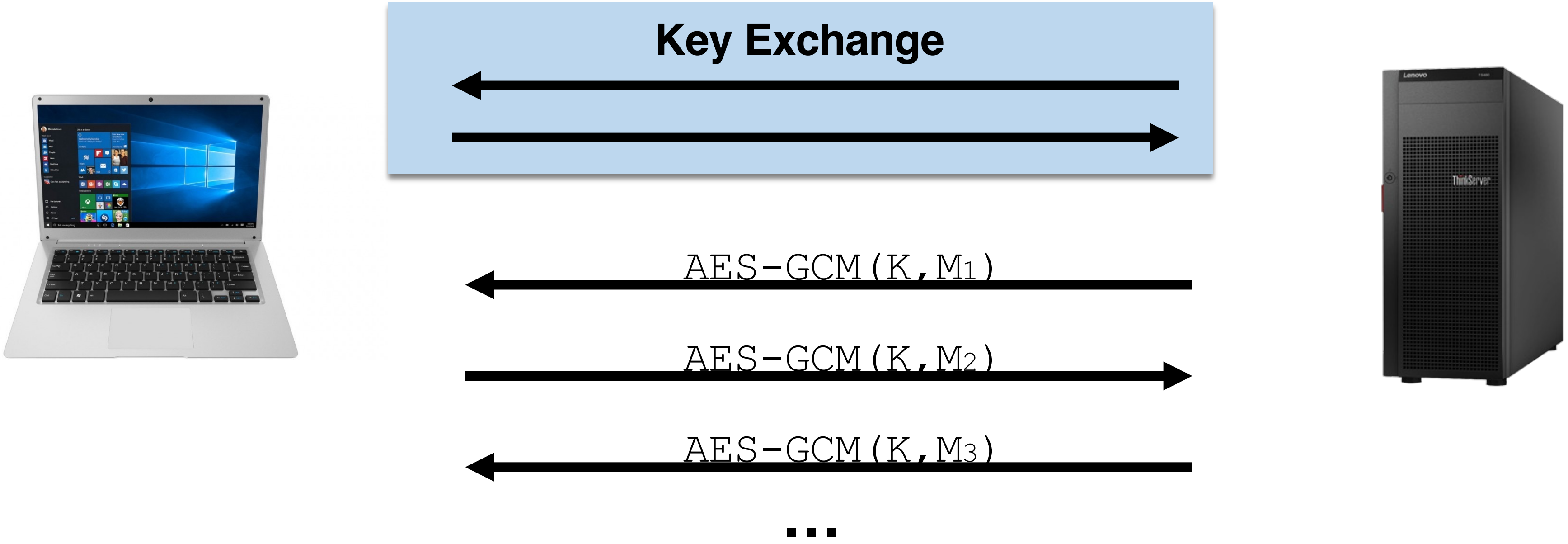
Hybrid Encryption: Real-world Secure Channels

Strategy:

1. Alice & Bob use a key exchange protocol to share their secret key(s)
2. Alice & Bob then use symmetric authenticated encryption (fast) for all their msg's



Key Exchange Protocols



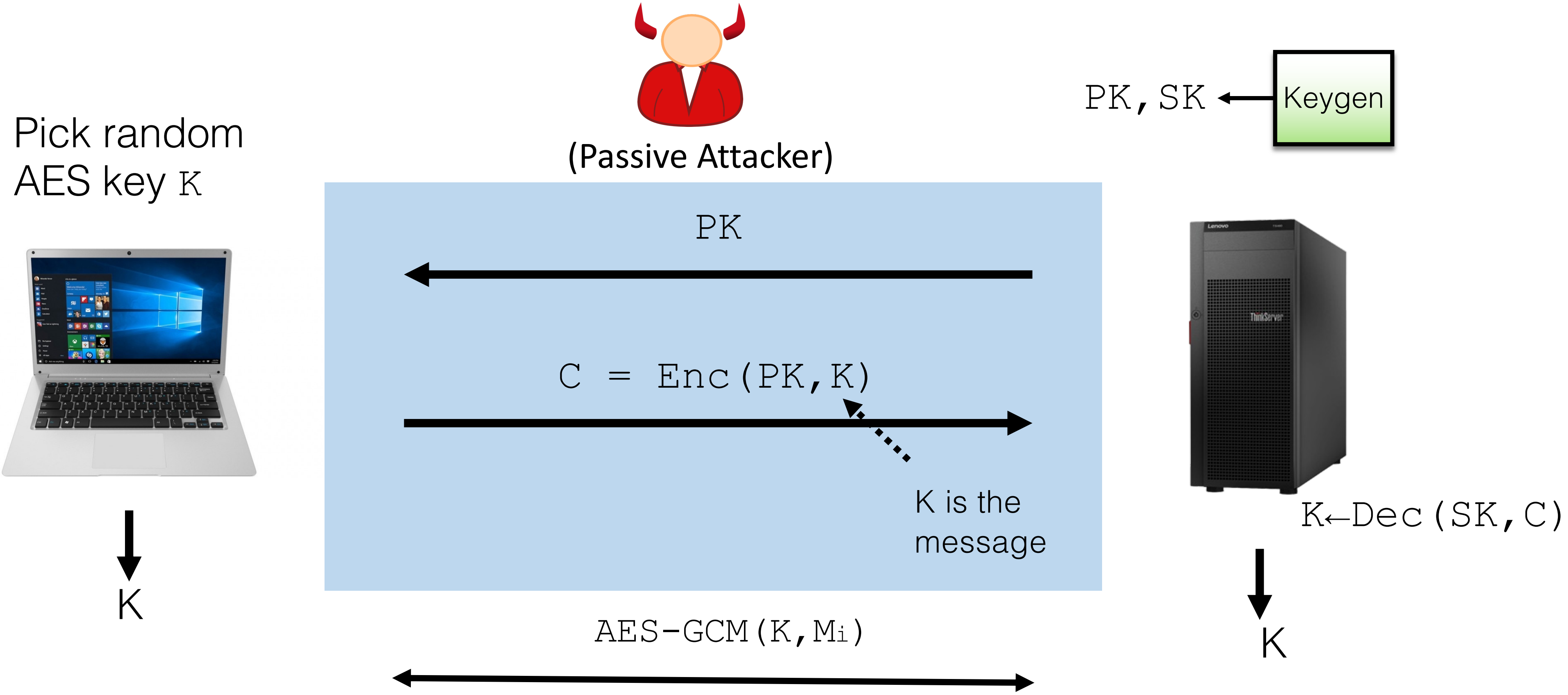
Options

1. Use public-key crypto algorithms (RSA encryption & signatures)
2. Use dedicated key exchange algorithms (Diffie-Hellman):
   Faster & recommended approach (e.g., TLS, SSH)

Key Exchange using Public Key Crypto

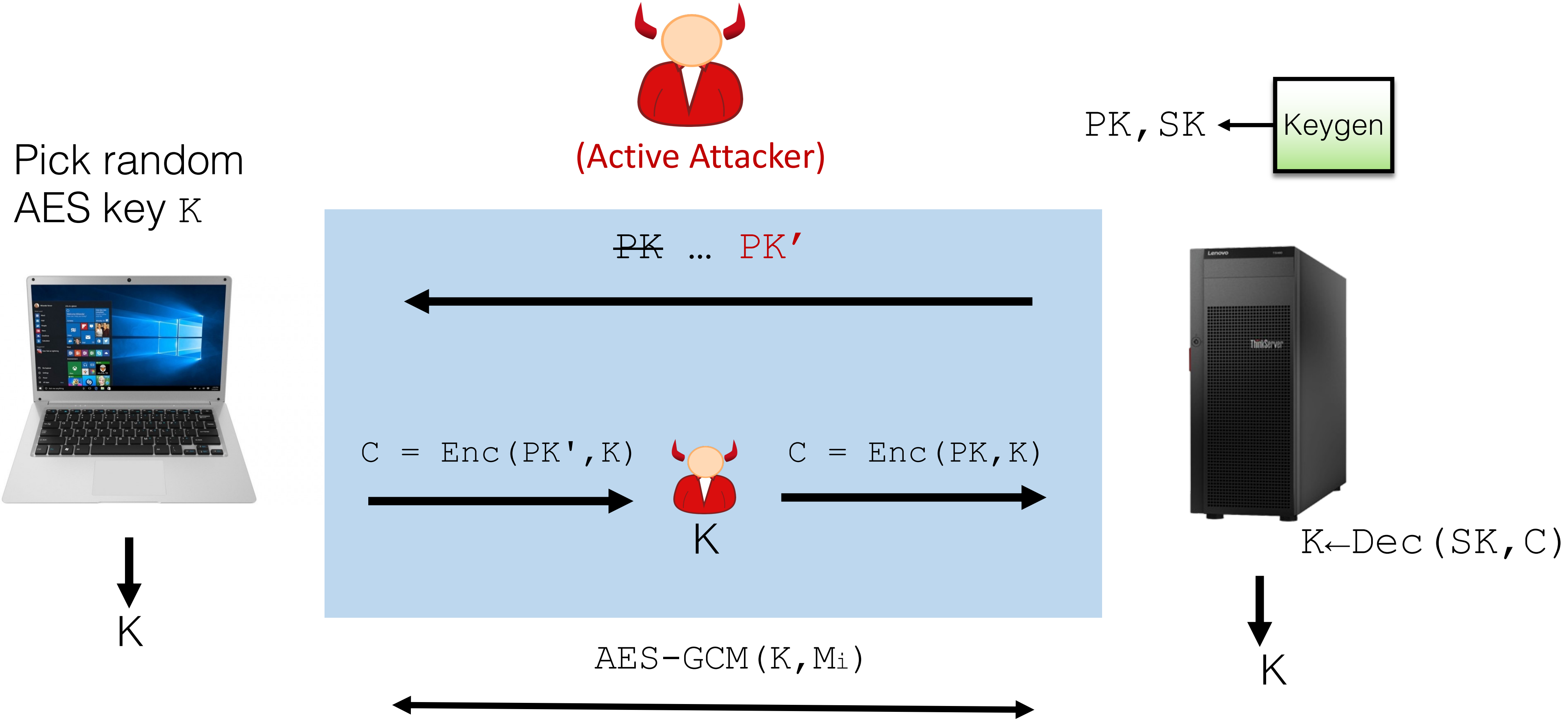
Goal: Establish secret key `K` to use with Authenticated Encryption.

`(KeyGen, Enc, Dec)` is a public-key encryption scheme.



Key Exchange using Public Key Crypto

Q: How do we make this secure against an active attacker?
A: Certificates w/ Signatures (Next Lecture)



The End