Cryptography Part 2

CMSC 23200, Spring 2025, Lecture 5

Grant Ho

University of Chicago, 04/08/2025

Logistics

Assignment 2 (Buffer Overflow): **Due Thursday, 11:59pm**

- For Assignment 2 only, you will use a ***different course VM*** read the assignment instructions for details
- **Test login for the new VM by end of tonight**

Discussion Section #2: tomorrow (Wed) @ assigned section times

Outline: Crypto Part 2

- **Symmetric Key Cryptography**
  - Block Cipher & Encryption Wrap-up
  - Integrity: MACs and Hash functions
  - Authenticated Encryption
- **Asymmetric (Public) Key Cryptography**
  - Public-Key Encryption
  - Digital Signatures

Block Ciphers: Symmetric Encryption Tool

- Block Ciphers (AES) act like Pseudo-random Permutations (PRP's)
  - If the attacker doesn't know the secret key (K), then: AES(K, x) = Random-looking string for different inputs (x)

- AES only encrypts 16 bytes at a time

- To encrypt more than 16 bytes, AES has different *modes* of operation that break up & encrypt a message as a series of 16-byte blocks
  - ECB : do not use – insecure!!
  - CTR & CBC : confidentiality, but not integrity
  - GCM: authenticated encryption

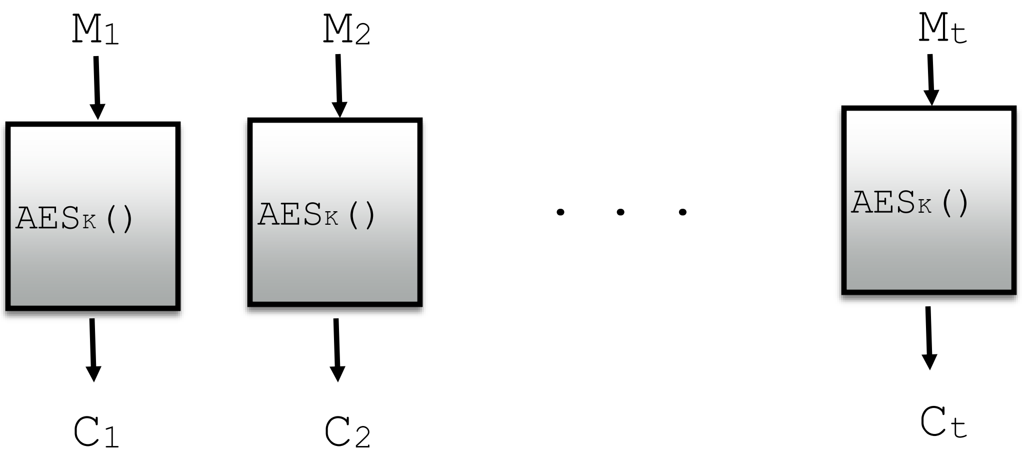
ECB Mode: Insecure!

Warning: Broken

ECB = "Electronic Code Book"

$\underline{\texttt{AES-ECB}_k\texttt{(M)}}$

- Split M into blocks $M_1, M_2, \ldots, M_t$
  // all blocks except M_t are 16 bytes
- Pad last block, M_t, up to 16 bytes
- For $i=1\ldots t$:
  - $C_i \leftarrow AES_k(M_i)$
- Return $C_1, \ldots, C_t$



Intuitively:

- Break message up into 16-byte chunks and encrypt each block with AES.

Insecure!

- Encrypting the same plaintext message multiple times always produces the same ciphertext

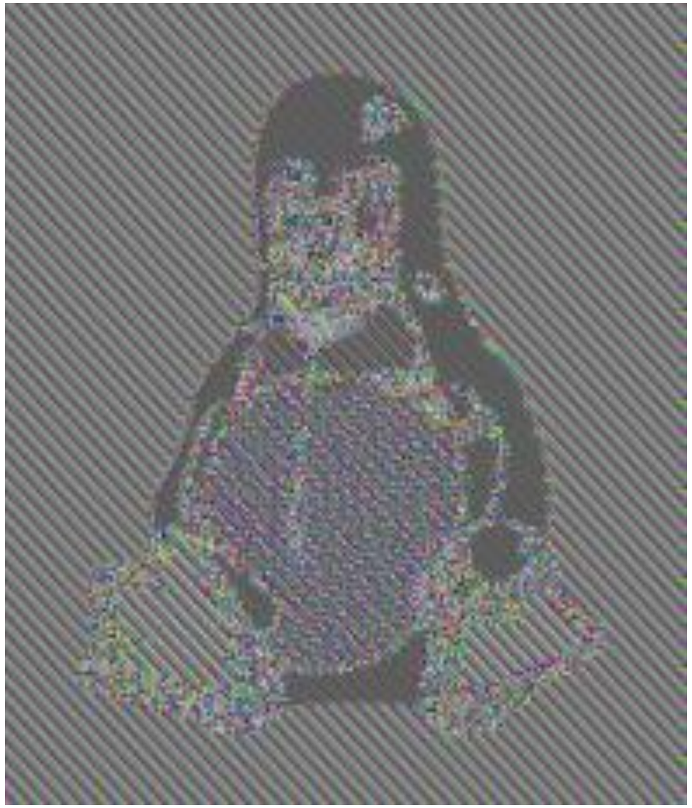
Example: The ECB Penguin

Warning: Broken

Treat pixel values as one long string & encrypt the string

Plaintext

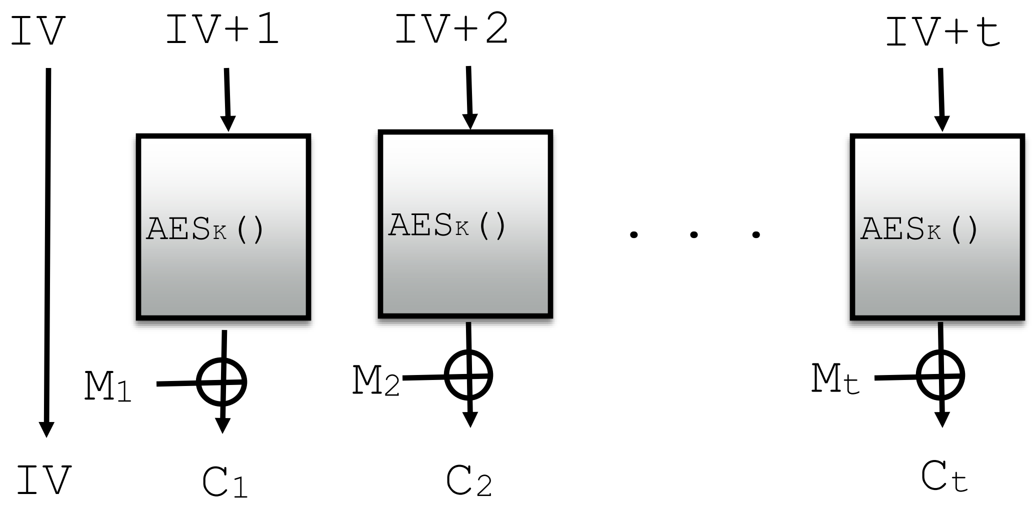
ECB Ciphertext



AES-CTR Mode: Secure Confidentiality

CTR = "Counter Mode"
- Idea: Build a stream cipher using AES & nonces

<u>AES-CTR$_k$(IV,M)</u>
- Split M into blocks $M_1, M_2, \ldots, M_t$
  // all blocks except M_t are 16 bytes
- IV ← random value
- For i=1...t:
  - $C_i \leftarrow M_i \oplus AES_k(IV+i)$
- Return IV, $C_1, \ldots, C_t$



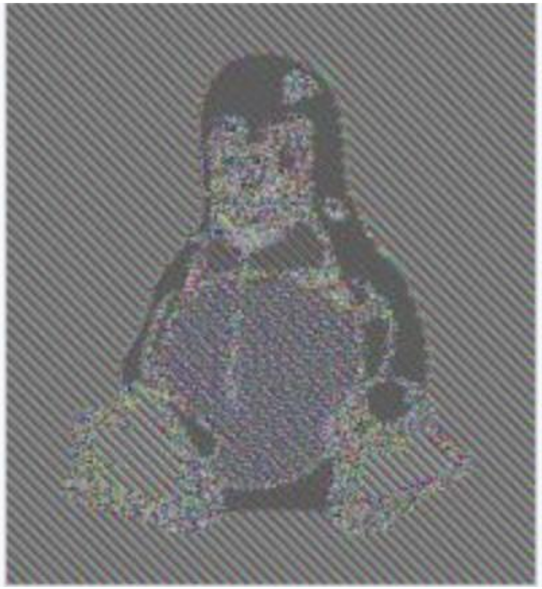
CTR mode creates "One-Time Pads" for each block, since AES output looks random for different inputs (nonces).

IV (nonce) chosen randomly & transmitted unencrypted.

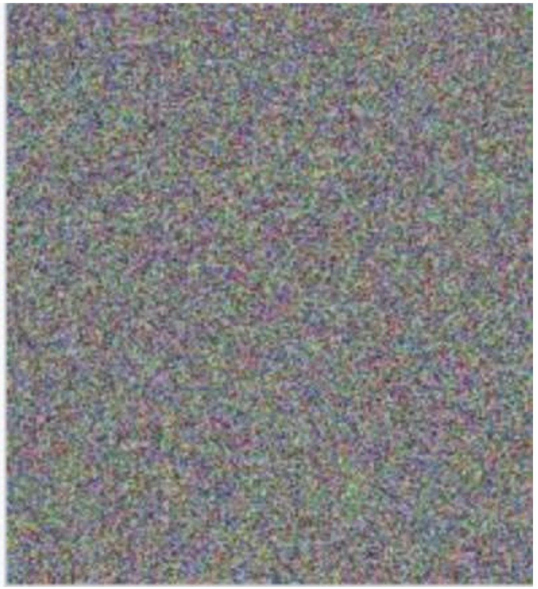
Penguin Sanity Check

Plaintext

ECB Ciphertext



CTR Ciphertext



Looks random

Encryption Summary

- **Security Goal (Confidentiality):** given encrypted ciphertexts, the attacker can learn nothing new about their plaintext contents

- One-time pads = strong security if pad (key) is ***never*** reused, but are impractical

- Stream ciphers & Block ciphers can achieve practical + secure confidentiality

- Block cipher modes matter for encryption security
  - AES-ECB (naïve block cipher) is INSECURE
  - Modes like AES-CTR and AES-CBC (not discussed) provide confidentiality

Outline: Crypto Part 2

- **Symmetric Key Cryptography**
  - Block Cipher & Encryption Wrap-up
  - Integrity: MACs and Hash functions
  - Authenticated Encryption

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

Integrity: Message Authentication Codes (MACs)

- **Encryption** provides **confidentiality**:
  a ***passive*** attacker can't learn anything about the data we're storing or using

- **Integrity**: an (active) attacker cannot tamper with the data in an undetectable manner
  - i.e., allows user to check if the data they received is exactly what was sent or if it has been modified

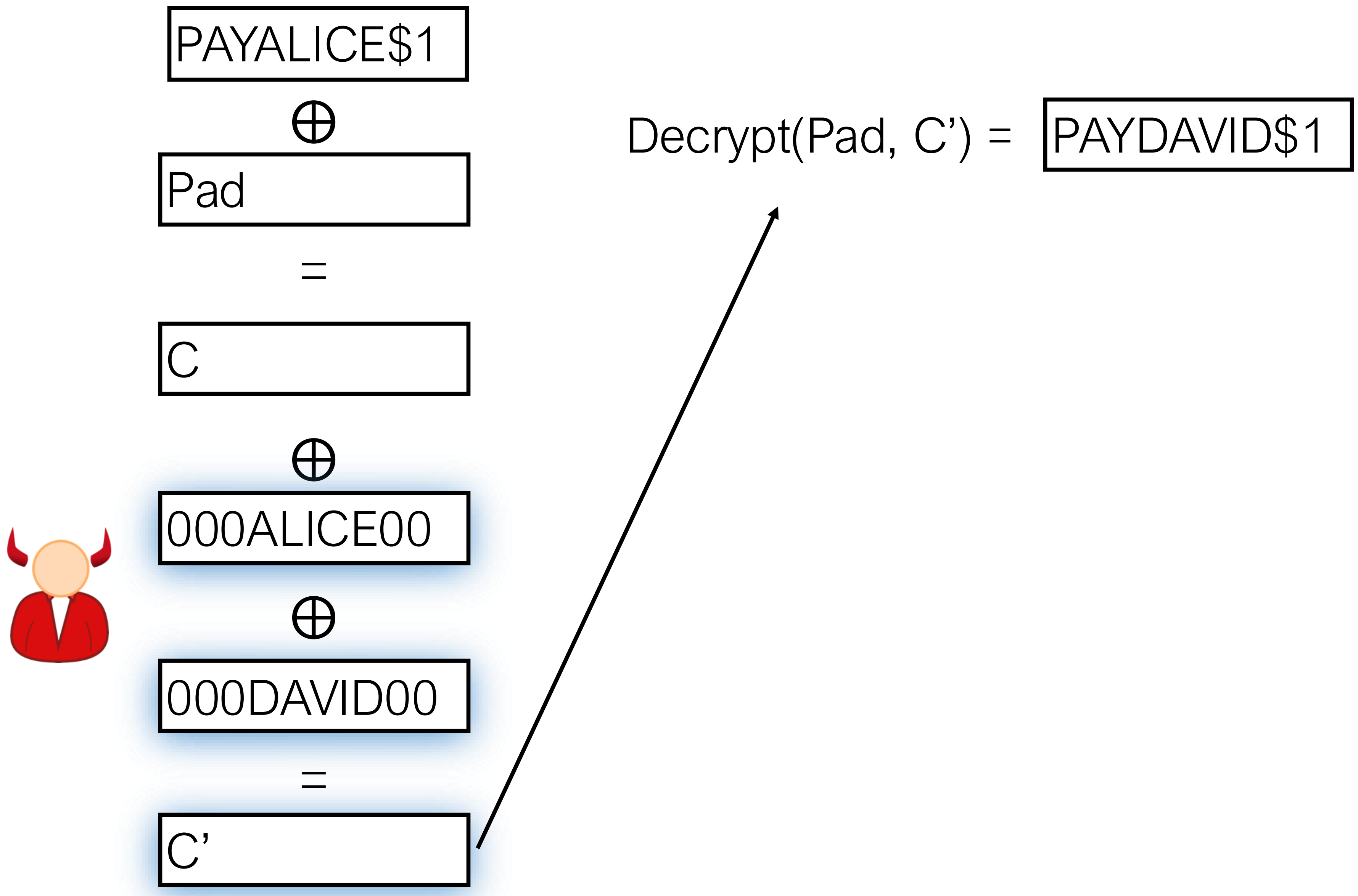
Integrity: New Threat Model (Active Attacker)

- **Threat model:** *Active* attacker that can tamper with communication

- Attacker not only sees all ciphertexts, but can also actively **modify** ciphertexts during transmission, **inject** their own data as additional "ciphertexts", **reorder or delete** ciphertexts

- Often known as a Man-in-the-Middle (MITM) attacker

OTP & Stream Ciphers Do Not Provide Integrity



Stream ciphers do not give integrity

M = please pay ben 20 bucks

C = b0595fafd05df4a7d8a04ced2d1ec800d2daed851ff509b3e446a782871c2d

C′ = b0595fafd05df4a7d8a04ced2d1ec800d2daed851ff509b3e546a782871c2d

M′ = please pay ben 21 bucks

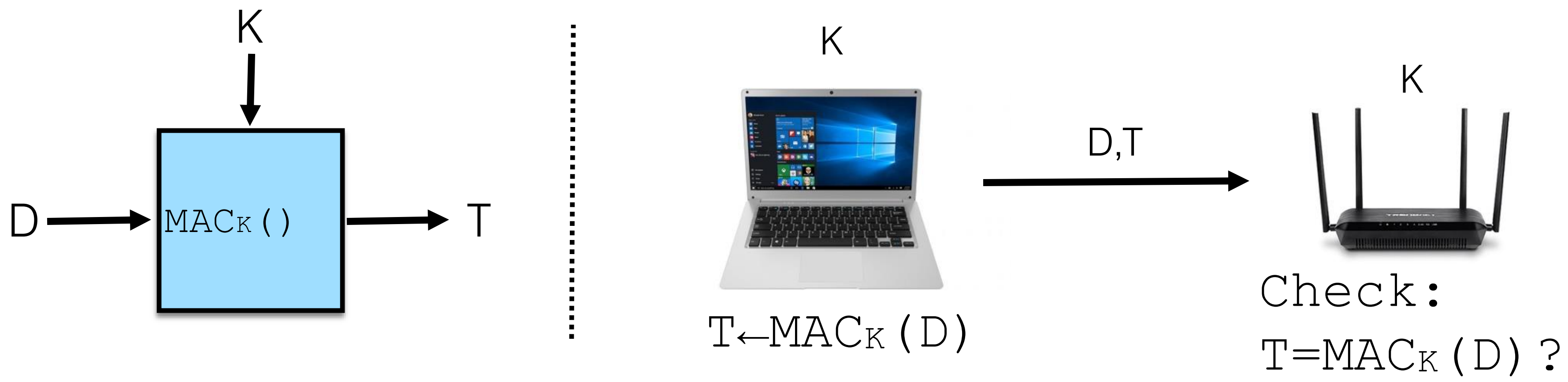
Encryption alone does not provide integrity
(fundamentally not designed to)

Providing Integrity: Message Authentication Code

Idea: Append a special tag to each message that
(1) validates the message content (different msg = different tag)
and (2) can only be computed if a user knows the secret key K

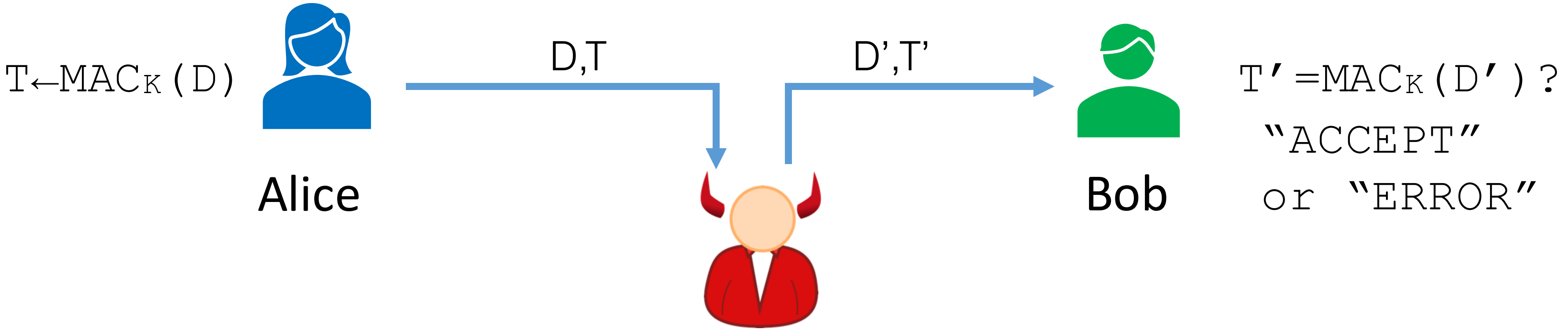
Providing Integrity: Message Authentication Code

A **message authentication code (MAC)** is an algorithm that takes as input a key and a message, and outputs an “unpredictable” **tag.**



D will usually be a ciphertext, but is often called a “message”.

MAC Security Goal: Unforgeability



MAC satisfies **unforgeability** if it is infeasible for Adversary to fool Bob into accepting `D'` and `T'` as a valid (msg, MAC) pair, for a `D'` that has not been previously seen

MAC Security Goal: Unforgeability

D = please pay ben 20 bucks

T = 827851dc9cf0f92ddcdc552572ffd8bc

D’= please pay ben 21 bucks

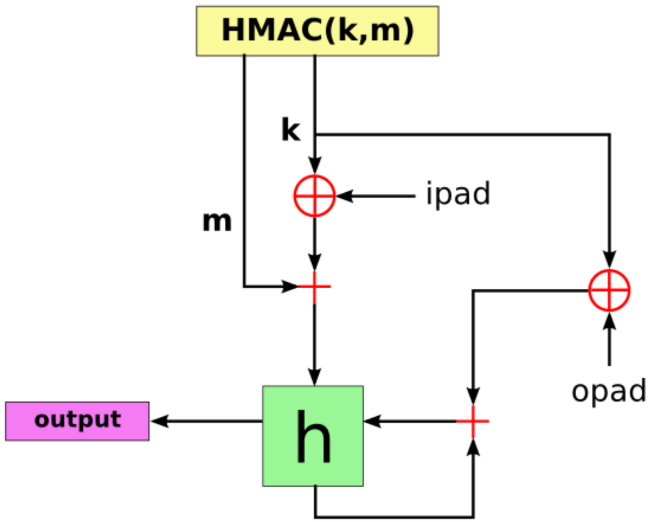
T’= baeaf48a891de588ce588f8535ef58b6

Unforgeability: Attacker cannot create T’ for any new D’.

- MACs do NOT need to provide any confidentiality (no encryption shown here)

MACs In Practice: Use HMAC or Poly1305-AES

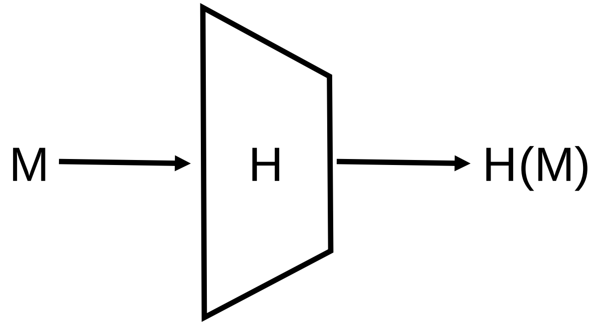
- More precisely: Use HMAC-SHA2.



- Other, less-good option: AES-CBC-MAC (bug-prone)

Building Block: Hash Functions

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

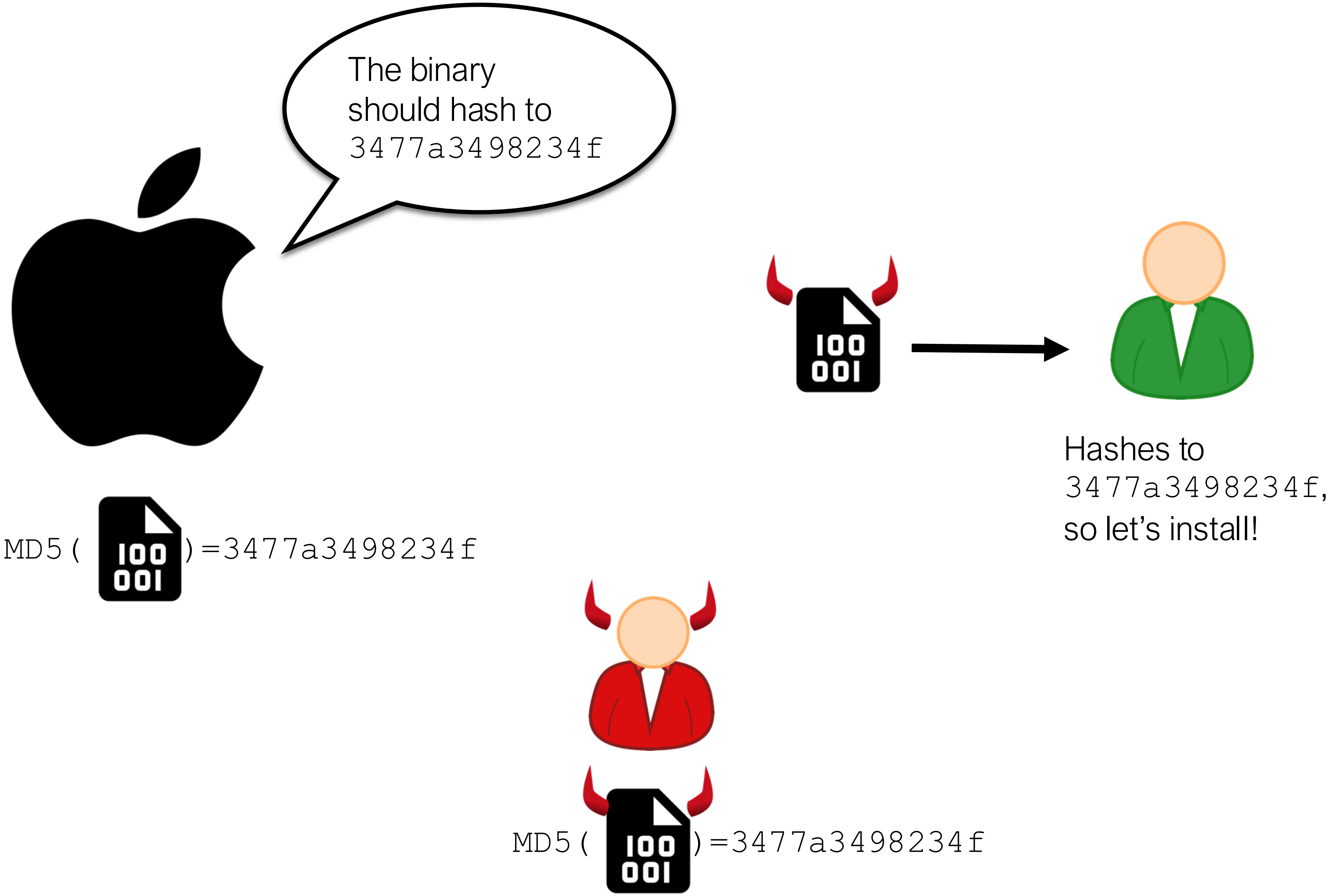


Properties of a ***secure*** hash function:

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!

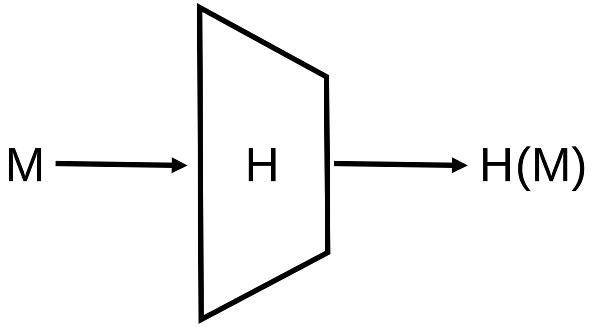
Why are hash collisions bad?

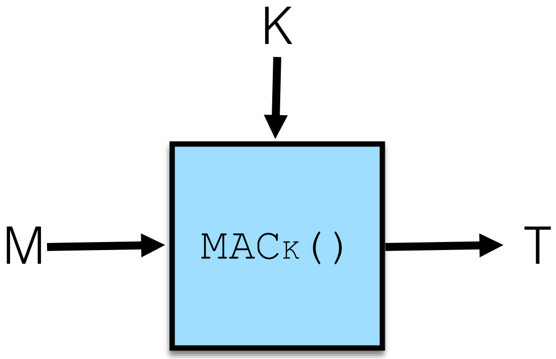


Practical Hash Functions

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,
but that only someone w/ the key can compute.

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**
  - Block Cipher & Encryption Wrap-up
  - Integrity: MACs and Hash functions
  - Authenticated Encryption
- **Asymmetric (Public) Key Cryptography**
  - Public-Key Encryption
  - Digital Signatures

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

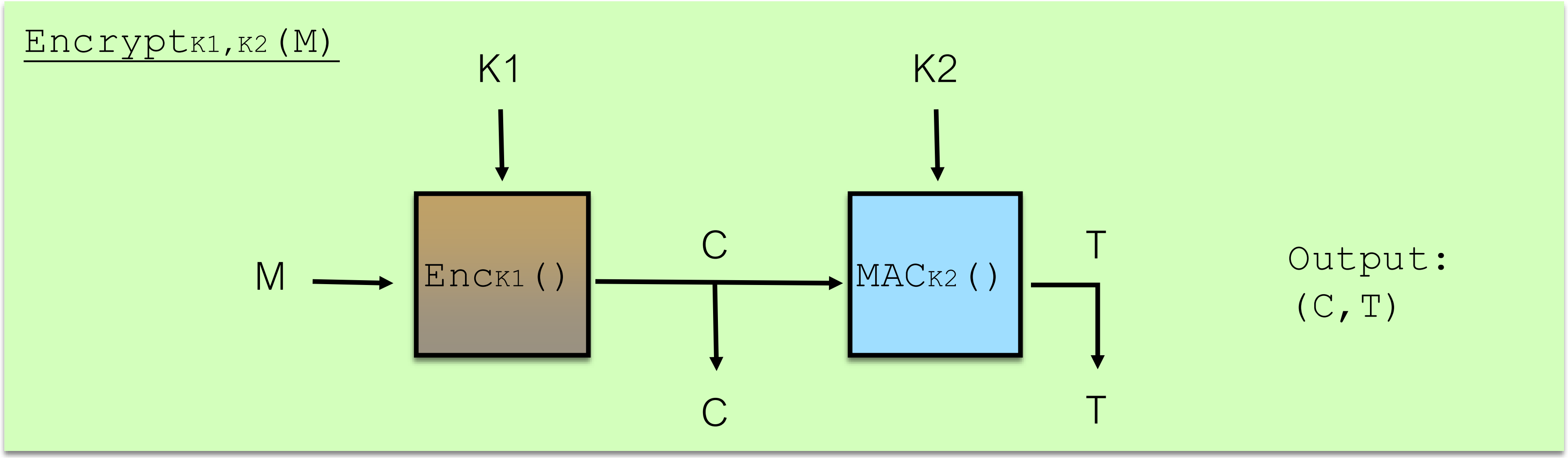
Authenticated Encryption

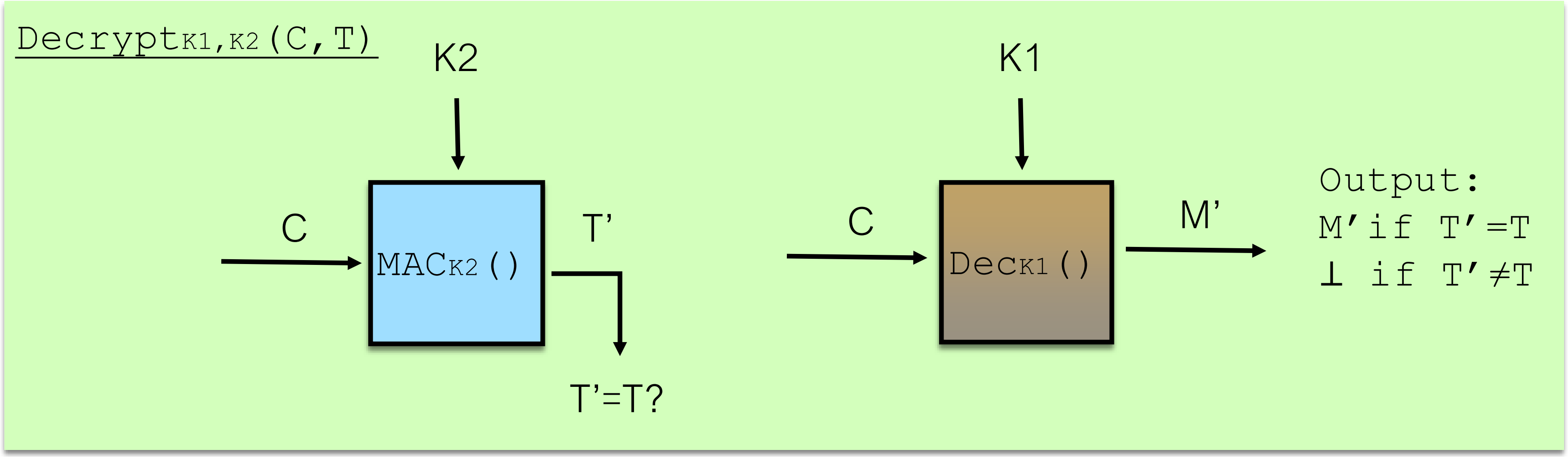
Authenticated Encryption algorithms provide both **confidentiality** and **integrity**.

- 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 (specific block cipher mode)

Building Authenticated Encryption





Encrypt message, then compute MAC on the ciphertext

5 MINUTE BREAK

Outline: Crypto Part 2

- **Symmetric Key Cryptography**
  - Block Cipher & Encryption Wrap-up
  - Integrity: MACs and Hash functions
  - Authenticated Encryption
- **Asymmetric (Public) Key Cryptography**
  - Public-Key Encryption
  - Digital Signatures

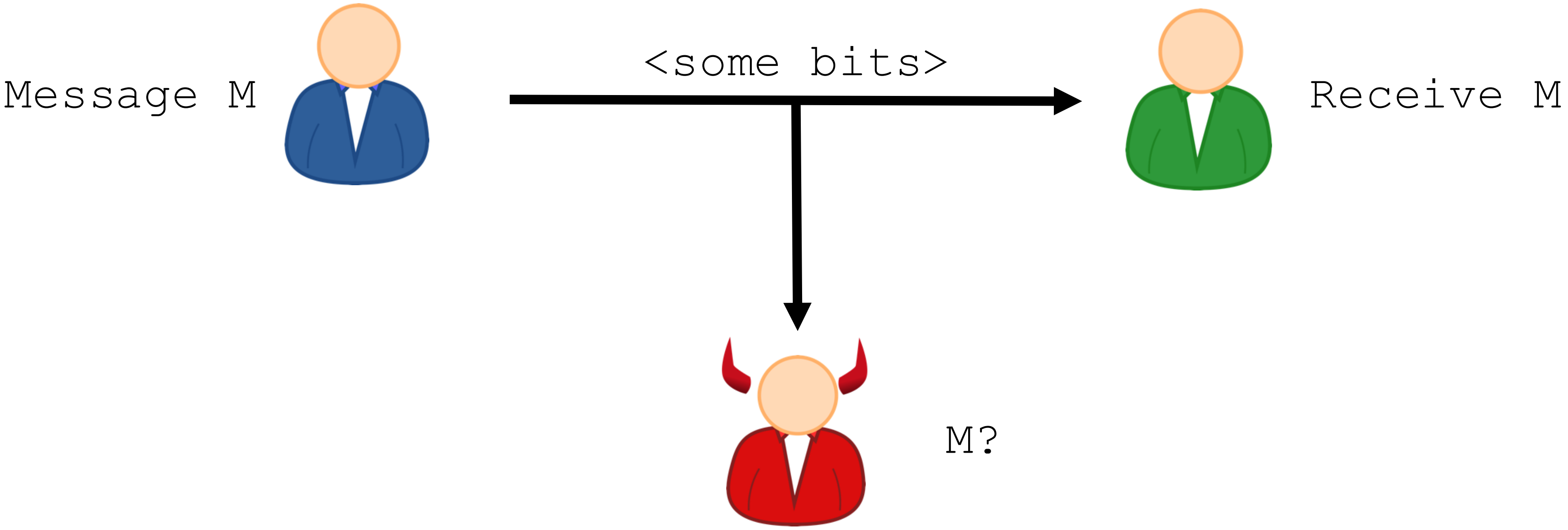
Why do we need Public-Key Cryptography?

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

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?

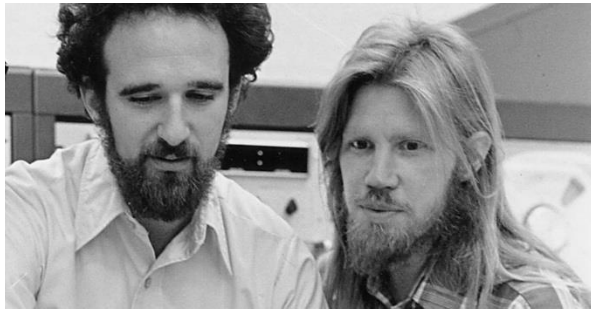
Motivation: If two people do <u>not</u> have a pre-shared secret key, can they send private messages in the presence of an attacker?



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!**

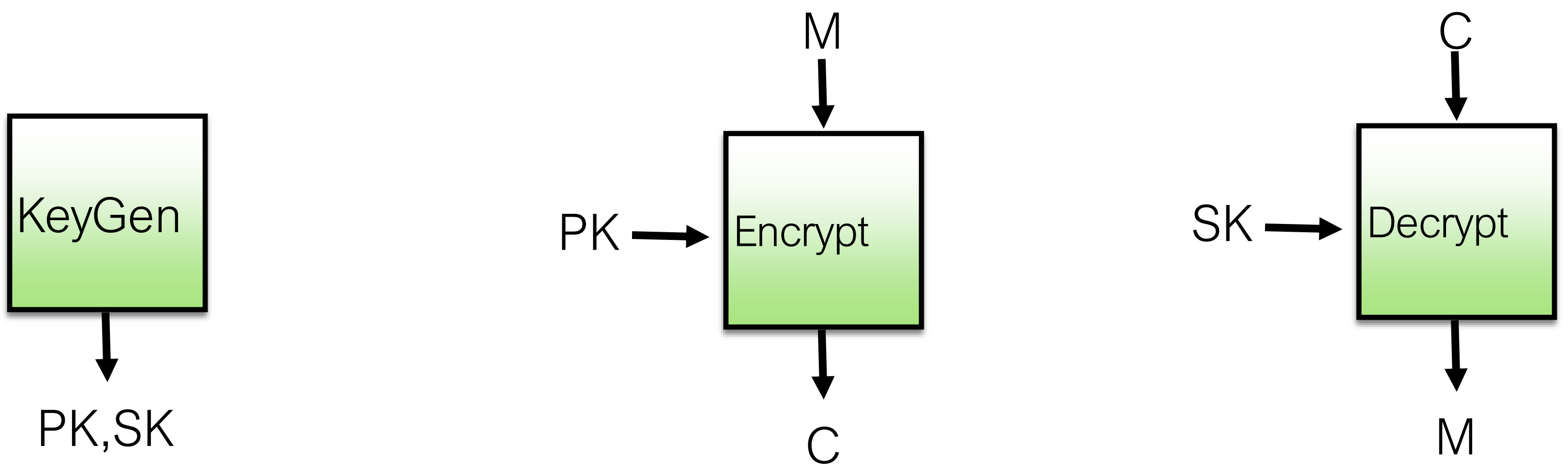
Turing Award, 2002

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

Public-Key Encryption (Confidentiality)

A public-key encryption scheme consists of three algorithms:

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



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

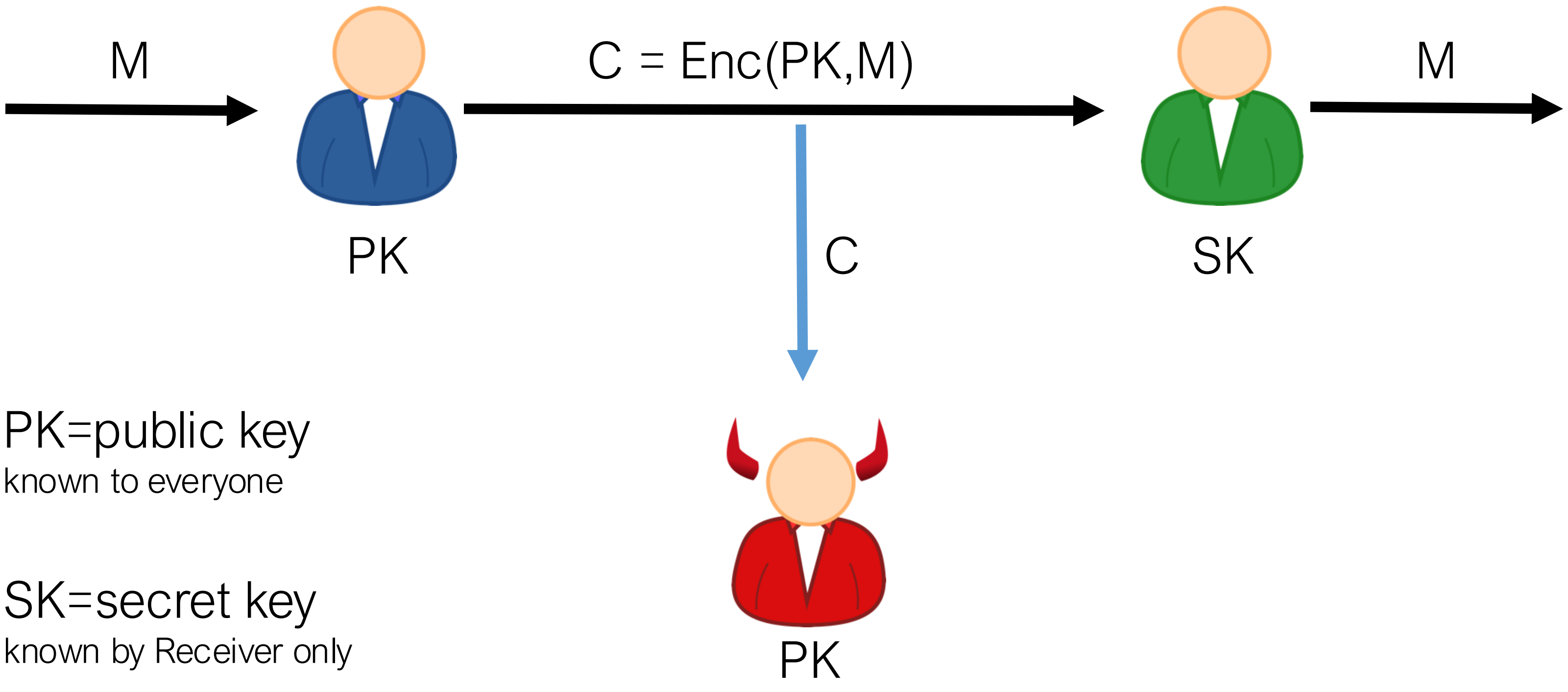
`Encrypt(PK, M)`:
Uses PK and M to produce a ciphertext C.

`Decrypt(SK, C)`:
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

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

Note: Taking modular roots is believed to be computational hard

Encryption & Decryption:

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

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

Using number theory from CMSC 27100, can show:

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

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

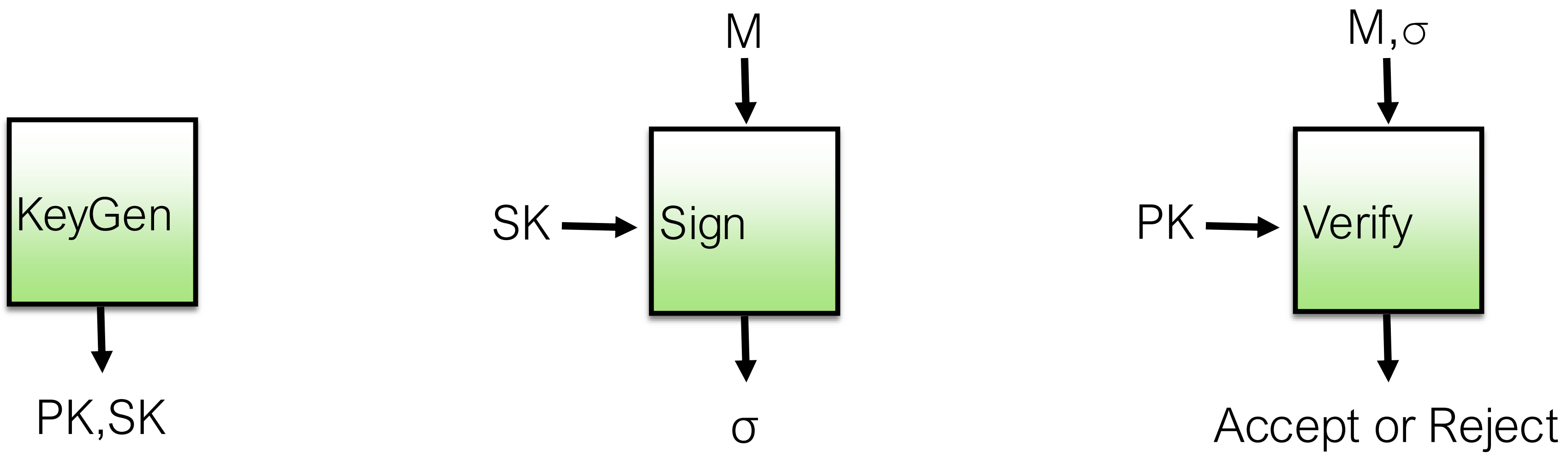
Outline: Crypto Part 2

- **Symmetric Key Cryptography**
  - Block Cipher & Encryption Wrap-up
  - Integrity: MACs and Hash functions
  - Authenticated Encryption
- **Asymmetric (Public) Key Cryptography**
  - Public-Key Encryption
  - Digital Signatures

Digital Signatures Schemes (Integrity & Auth)

A digital signature scheme consists of three algorithms

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

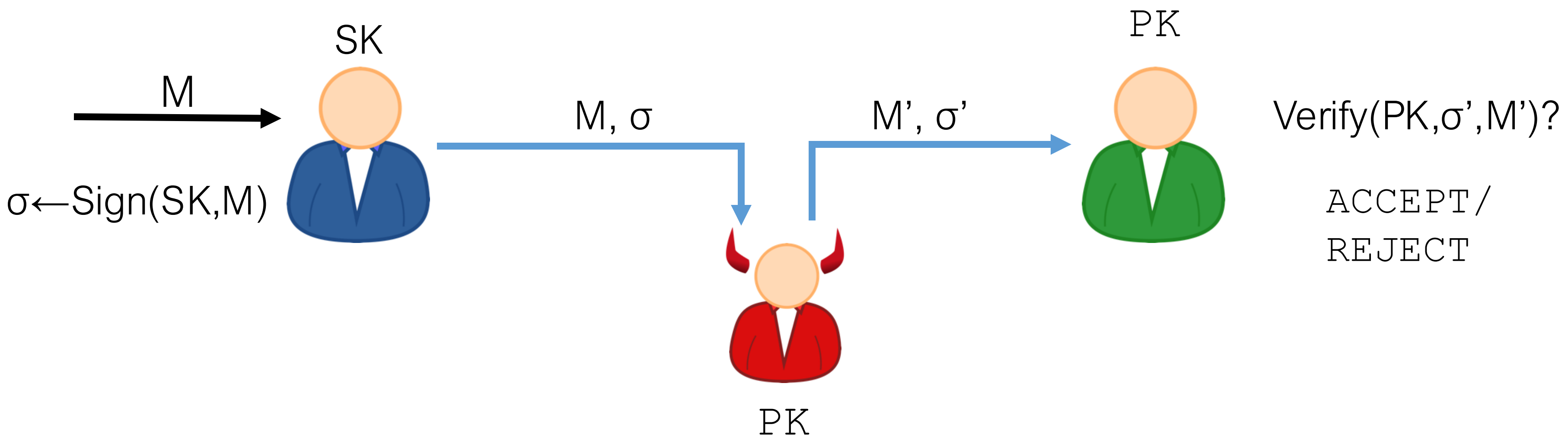


`KeyGen`: Outputs two keys. PK published openly, and SK kept secret.

`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.

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

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

"Plain" RSA Weaknesses

Broken

Assume $e=3$.

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

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

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

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

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 $\sigma'=2$.

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

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

More “Plain” RSA Weaknesses

Broken

Assume $e=3$.

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

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

***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: $\text{Verify}((N,3), M', \sigma')$ checks:

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

More “Plain” RSA Weaknesses

Broken

Assume e=3.

$Sign((N,d),M) = M^d mod N$ 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

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

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

M′ = (8*M mod N), and σ′ = (2*σ mod N)

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

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

More "Plain" RSA Weaknesses

Broken

Assume e=3.

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

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

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

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

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

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

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

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

Secure RSA Signatures with Encodings

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

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

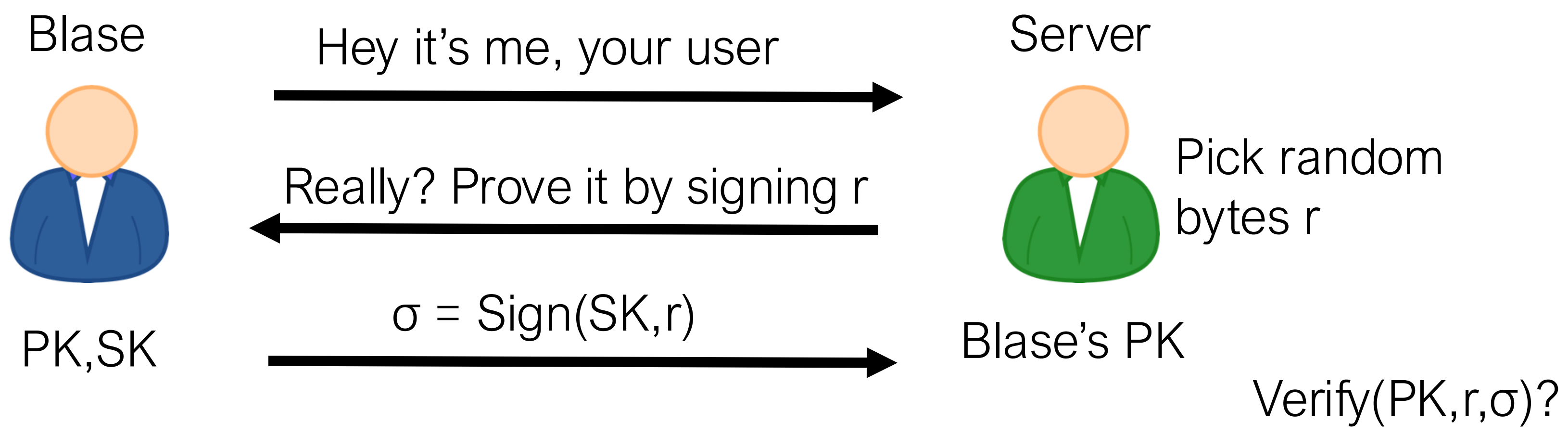
$$\text{Verify}\big((N,e), M, \sigma\big): \sigma^e = \text{encode}(M) mod N?$$

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**
  - Block Cipher & Encryption Wrap-up
  - Integrity: MACs and Hash functions
  - Authenticated Encryption

- **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>

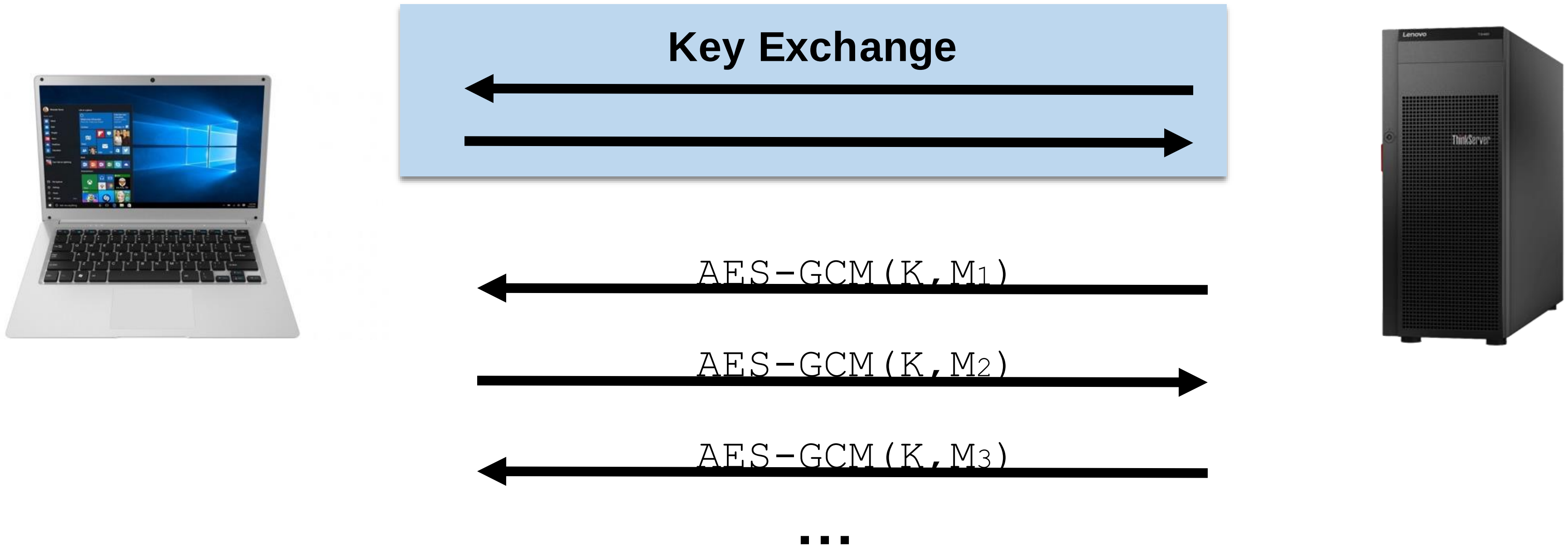
Symmetric key crypto algorithms are **MUCH** faster

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

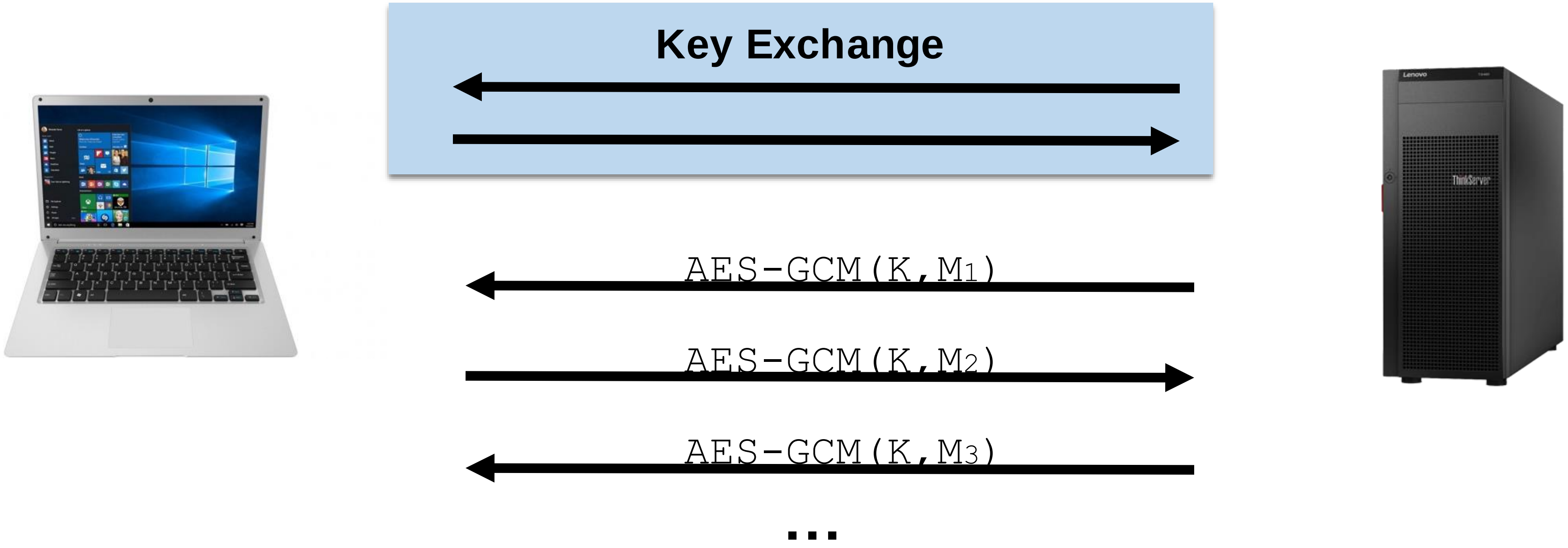
Hybrid Encryption: Real-world Secure Channels

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)

The End