Cryptography, Part 1

CMSC 23200, Spring 2026, Lecture 5

David Cash and Grant Ho

University of Chicago

Amazon.com: Online Shopp

https://www.amazon.com

www.amazon.com

Your connection to this site is private.

Details

Permissions | Connection

Chrome verified that Symantec Class 3 Secure Server CA - G4 issued this website's certificate. The server did not supply any Certificate Transparency information.

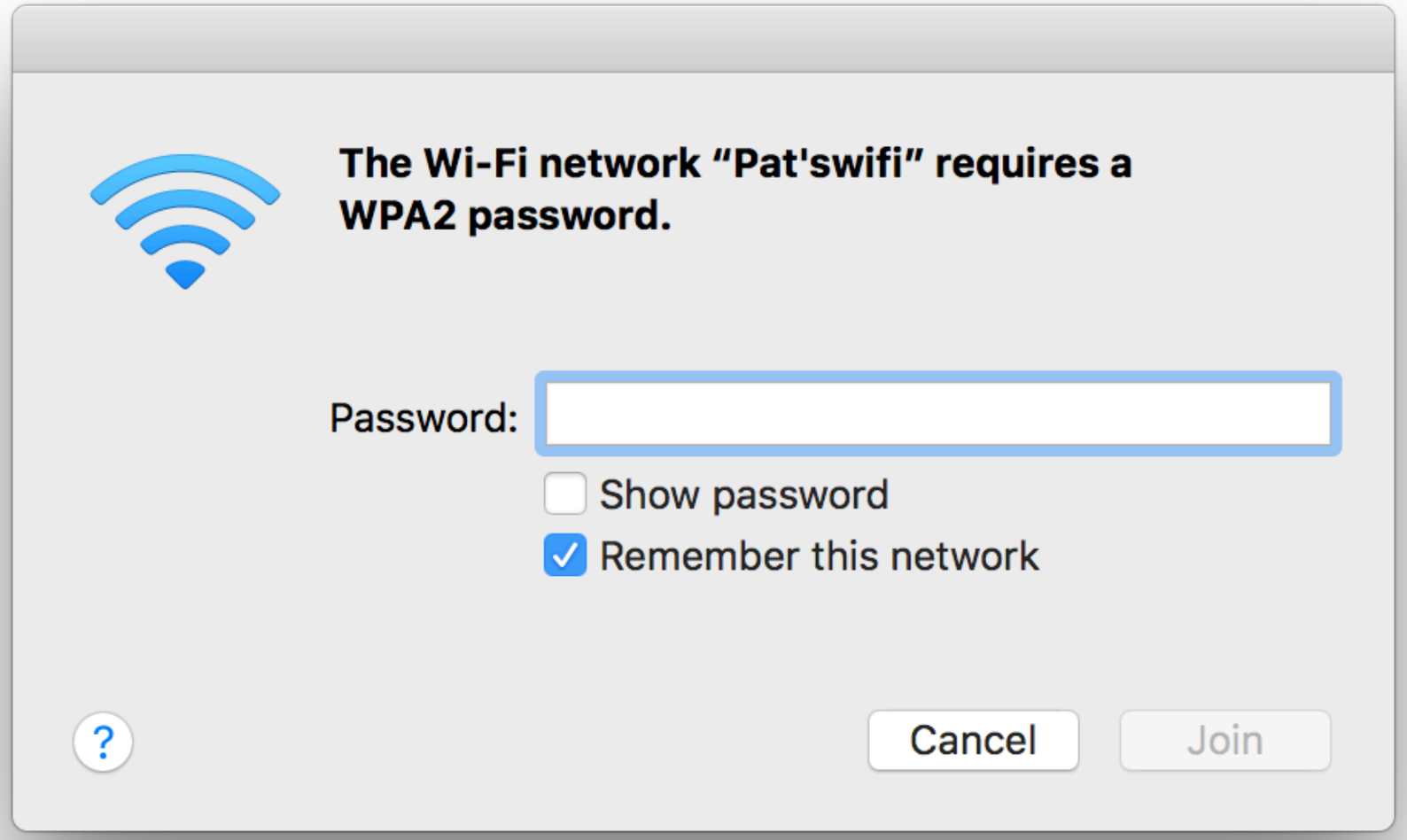
Certificate Information

Your connection to www.amazon.com is encrypted using a modern cipher suite.

The connection uses TLS 1.2.

The connection is encrypted and authenticated using AES_128_GCM and uses ECDHE_RSA as the key exchange mechanism.

What do these mean?


The Wi-Fi network “Pat'swifi” requires a WPA2 password.
Password:
Show password
Remember this network
?
Cancel
Join

nothing

Today 11:11

Can you please come over asap to help me move the couch?

I need to be out of here by 3pm

I guess you forgot your phone at home or something

Delivered

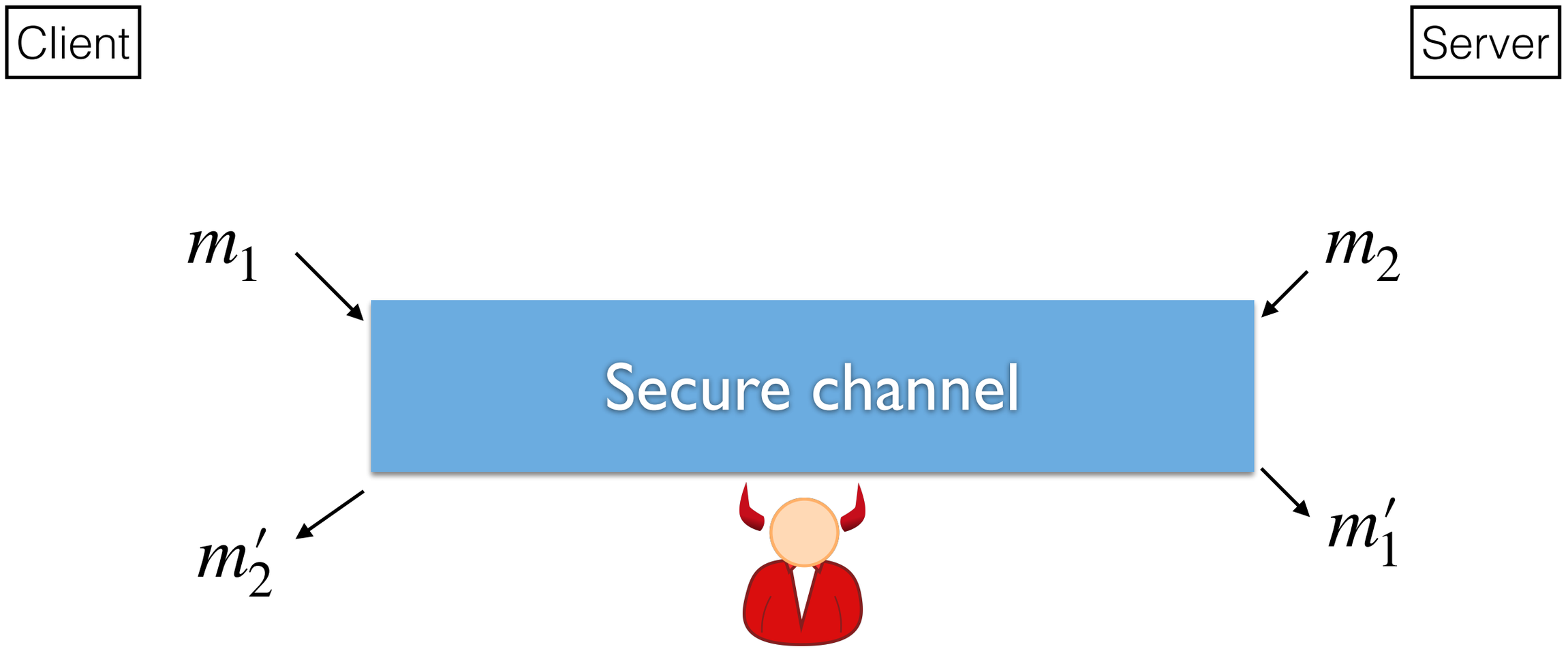
Send

iMessage

5100 21

5100

Typical Abstract Goal of Crypto



Confidentiality: Adversary does not learn anything about messages m_1, m_2

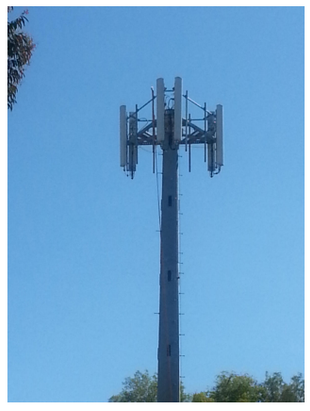
Integrity/Authenticity: $m_1' = m_1$ and $m_2' = m_2$

Example settings

Web

amazon

Cell phones



Disk encryption

macOS®

Cryptography in CMSC 23200

- We will cover the goals and applications of commonly used crypto tools
- We will peel back the abstract a few layers to understand some crypto designs and the attacks that motivate the designs
- In later lectures we see how these tools are used in more detail
- Not covered: Theory, proofs.
- Consider taking CMSC 28400, “Introduction to Cryptography”!

Roadmap for Cryptography in CMSC 23200

- We will cover tools for confidentiality and authenticity/integrity in two settings:
  - End-points have previously shared a key ("Symmetric" case)
  - End-points have not shared a key ("Asymmetric" case)

Pre-shared key? \ Security Goal	Confidentiality	Authenticity/Integrity
Yes ("Symmetric")	Symmetric Encryption (Stream Ciphers and Block Ciphers)	Message Authentication Codes (MACs)
No ("Asymmetric")	Public-Key Encryption (RSA) and Key Exchange (Diffie-Hellman)	Digital Signatures

Outline of Lecture 5

1. Encryption Basics
2. One-Time Pad Encryption
3. Practical Tool #1: Stream Ciphers
4. Practical Tool #2: Blockciphers
5. Message Authentication
6. Hash Functions

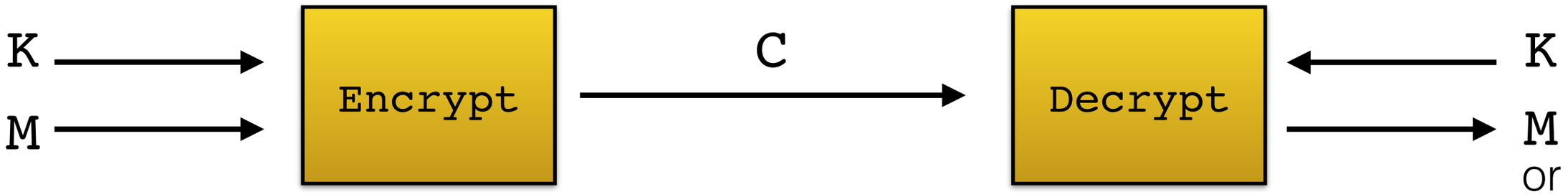
Outline of Lecture 5

1. **Encryption Basics**
2. One-Time Pad Encryption
3. Practical Tool #1: Stream Ciphers
4. Practical Tool #2: Blockciphers
5. Message Authentication
6. Hash Functions

Encryption Schemes (Abstractly)

Definition: A encryption scheme is a pair of algorithms (`Encrypt`, `Decrypt`) for encryption and decryption respectively.

- `K`: key (16 or 32 bytes)
- `M`: Plaintext (long)
- `C`: Ciphertext (long)



- `K` is a relatively short random value (typically 16 or 32 bytes) shared between endpoints and kept secret
  - Allow decryption to reject “malformed” ciphertexts
- If correct ciphertext is received, then plaintext should be recovered

Historical Example 1: Shift Cipher

- Key `K` is number from 0 to 25
- Simply shift characters forward in alphabet by `K` steps

```
Encrypt(K,M):
  C = ""
  For each character x in M:
    Shift x forward by K steps (wrap mod 26)
    Append result to C
 Output C
```

Examples:

Plaintext:	`DEFGH`
Key:	3
Ciphertext:	`FGHKL`

Plaintext:	`ATTACKATDAWN`
Key:	13
Ciphertext:	`NGGNPXNGQNJA`

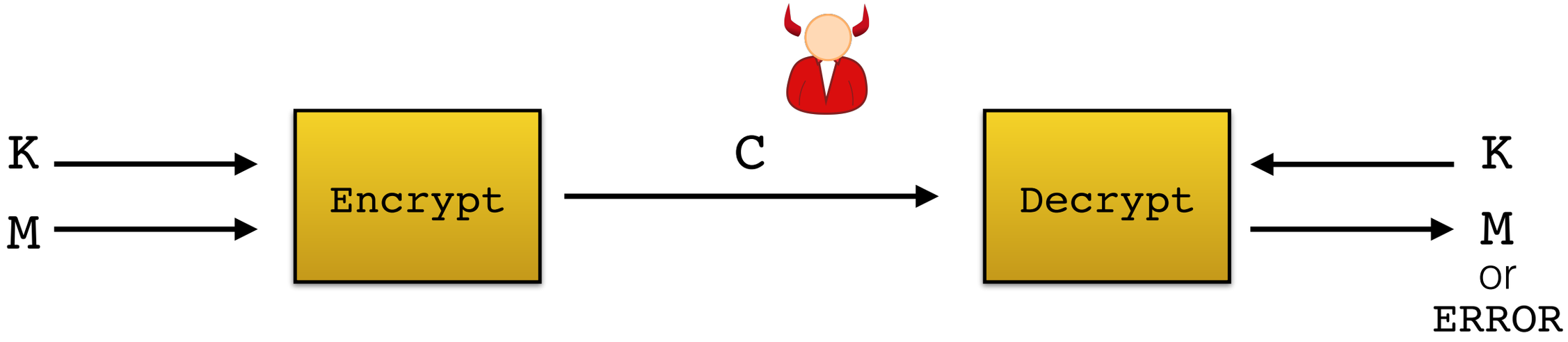
Security of Encryption

- "Security" is extremely subtle to define for encryption and other crypto tools
  - Most of CS 284 is about this

- We typically measure security by defining:
  1. **Adversary goals**: What are they trying to achieve?
  2. **Adversary capabilities**: How can they probe our system?
  3. **Adversary strength**: How much compute can they afford?

- We will always assume:
  - Adversary knows our algorithm (so they can run it too)
  - Adversary does not know the key

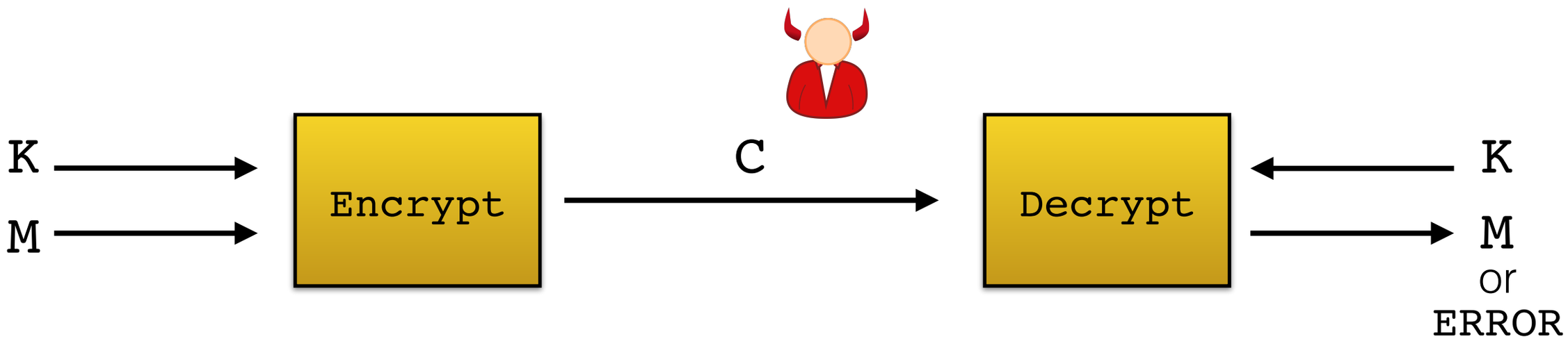
1. Adversary Goal: Breaking Confidentiality



- **Break confidentiality**: Want to recover non-trivial information about m.

- Recovering all of message certainly counts...
- ... but recovering only part of the message usually enough (even one bit)
- Example: `m = http://site.com?password=`▮▮▮▮▮▮▮▮

2. Adversary Capability: Eavesdropping



- Adversary passively sees ciphertexts as they are sent, but cannot change them
- Later: We will allow adversary to modify/insert ciphertexts and plaintexts

3. Adversary Strength

- We coarsely measure strength in “number of computations steps” or “cycles”

# Steps	Who can do that many?
2^{56}	Strong computer with GPUs
2^{80}	All computers on Bitcoin network in under 1 hour
2^{128}	Very large quantum computer? (Ask Diana,Fred,Bill,Robert)*
2^{192}	Nobody?
2^{256}	Nobody?

*Not directly comparable but this is an estimate of equivalent power. Quantum computers are most effective against public-key crypto, but they also speed up attacks on the crypto I’m doing today. (More next time.)

Historical Example 2: Substitution Cipher

- Key is any permutation π from alphabet to itself
- To encrypt, apply π to each character

```
Encrypt(K,M):
  C = ""
  For each character x in M:
    Append π(x) to C
  Output C
```

Example: P: ATTACKATDAWN
K: π
C: ZKKZAMZKYZGT

x	π(x)
A	Z
B	U
C	A
D	Y
E	R
F	E
G	X
H	B
I	D
J	C
K	M
L	Q
M	H
N	T
O	I
P	S
Q	V
R	N
S	P
T	K
U	O
V	F
W	G
X	W
Y	L
Z	J

Brute Force Attacks Against Encryption

- Any attacker can always run a **brute force attack**:

```
BruteForceAttack(C):
  For every possible key K:
    M = Decrypt(K,C)
    If M looks correct:
      Output M
```

- Time scales with number of keys
- Encryption schemes must have lots of keys to defeat brute force
  - Typically 2^{128} or 2^{256}

Brute Force Against the Substitution Cipher?

```
Encrypt(K,M):
  C = ""
  For each character x in M:
    Append π(x) to C
  Output C
```

- How many possible keys are there? (CS 271 time!)
  - 26! ... is that big?
  - $\approx 2^{88}$
  - Too many for almost any adversary

x	π(x)
A	Z
B	U
C	A
D	Y
E	R
F	E
G	X
H	B
I	D
J	C
K	M
L	Q
M	H
N	T
O	I
P	S
Q	V
R	N
S	P
T	K
U	O
V	F
W	G
X	W
Y	L
Z	J

The Substitution Cipher Does Not Hide Partial Info

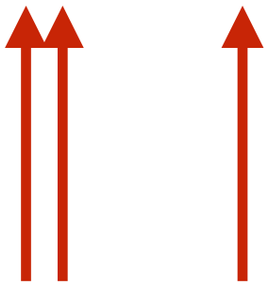
```
Encrypt(K,m):
  C = ""
  For each character x in m:
    Append π(x) to C
  Output C
```

x	π(x)
A	Z
B	U
C	A
D	Y
E	R
F	E
G	X
H	B
I	D
J	C
K	M
L	Q
M	H
N	T
O	I
P	S
Q	V
R	N
S	P
T	K
U	O
V	F
W	G
X	W
Y	L
Z	J

Example: P: ATTACKATDAWN

K: π

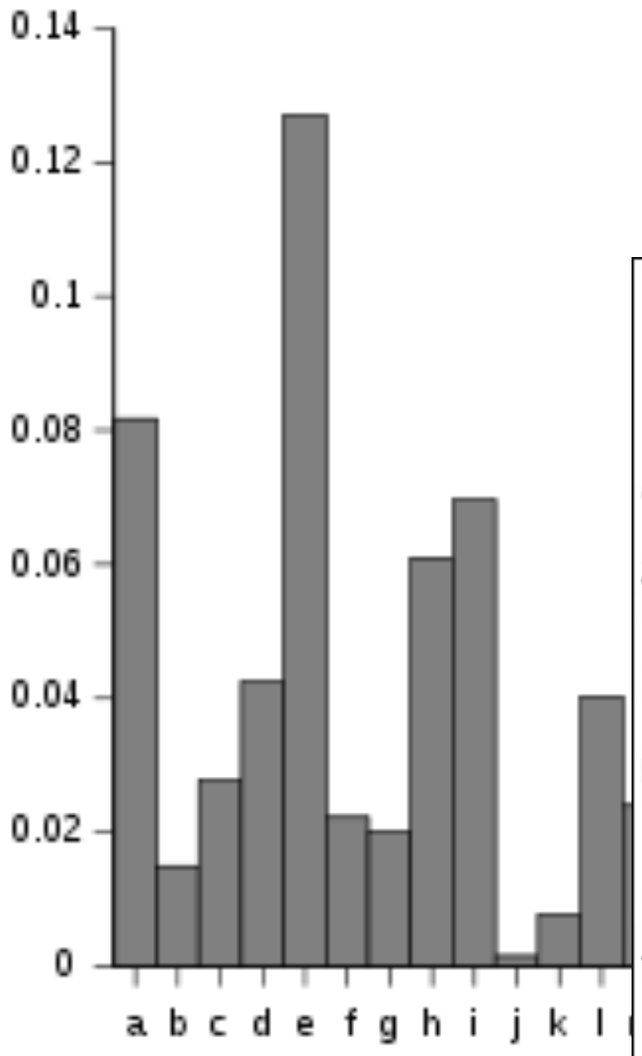
C: ZKKZAMZKYZGT



Repeats in ciphertext indicate repeats in plaintext.

Cryptanalysis of Substitution Cipher

- Partial information leakage can be boosted to recover entire plaintext



CELEBRITY CIPHER

by Luis Campos

Celebrity Cipher cryptograms are created from quotations by famous people, past and present. Each letter in the cipher stands for another.

"U P X E G T H W Z H F X Y L F H O S L N P F X. H M T P J S X E P O X V P G A V G P O S L E E B X X O A P M L N P F X, T P J 'C X Z P W E E B X V B P Z X E B H O S." — V. W. Y X G V H O

Previous Solution: "Time is the cruelest teacher; first she gives the test, then teaches the lesson." — Leonard Bernstein

TODAY'S CLUE: ſ ʃɹǝnbǝ N

Outline of Lecture 5

1. Encryption Basics
2. **One-Time Pad Encryption**
3. Practical Tool #1: Stream Ciphers
4. Practical Tool #2: Blockciphers
5. Message Authentication
6. Hash Functions

Quick recall: Bitwise-XOR operation

We will use bit-wise XOR:

$$\begin{array}{r} 0101 \\ \oplus\ 1100 \\ \hline 1001 \end{array}$$

Some Properties:

- $X \oplus Y = Y \oplus X$
- $X \oplus X = 000\ldots0$
- $X \oplus Y \oplus X = Y$

Very Important Cipher Example: One-Time Pad

- Fixed parameter: message length **n**
- Key is any bitstring of length **n**
- Message is any bitstring of length **n**

```
Encrypt(K,M):
  C = K ⊕ M
  Output C
```

```
Decrypt(K,C):
  M = K ⊕ C
  Output M
```

- Decryption is correct because

$$\begin{aligned} K \oplus C &= K \oplus (K \oplus M) \\ &= (K \oplus K) \oplus M \\ &= 0^n \oplus M \\ &= M \end{aligned}$$

Security of One-Time Pad

Theorem: If adversary sees **only one** ciphertext under a random key, then any plaintext is equally likely, so it cannot recover any partial information besides plaintext length.

Ciphertext observed: `10111`
Possible plaintext: `00101`
⇒ Possible key: `10010`

Ciphertext observed: `10111`
Possible plaintext: `11111`
⇒ Possible key: `01000`

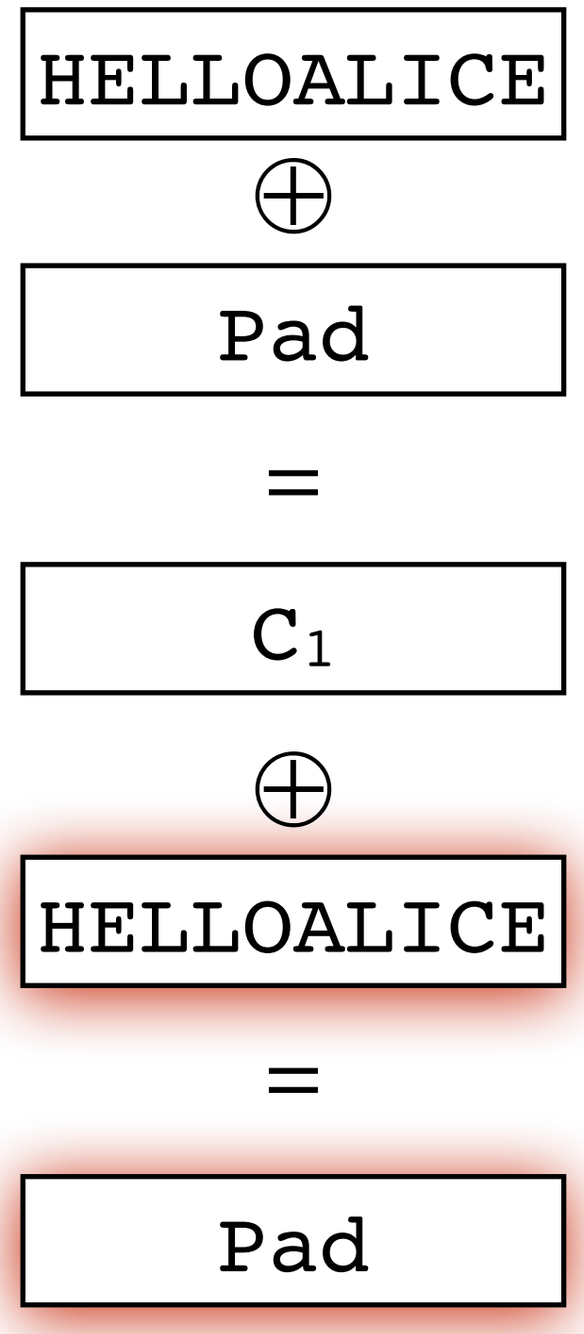
According to our criteria, we have security in the follow setting:

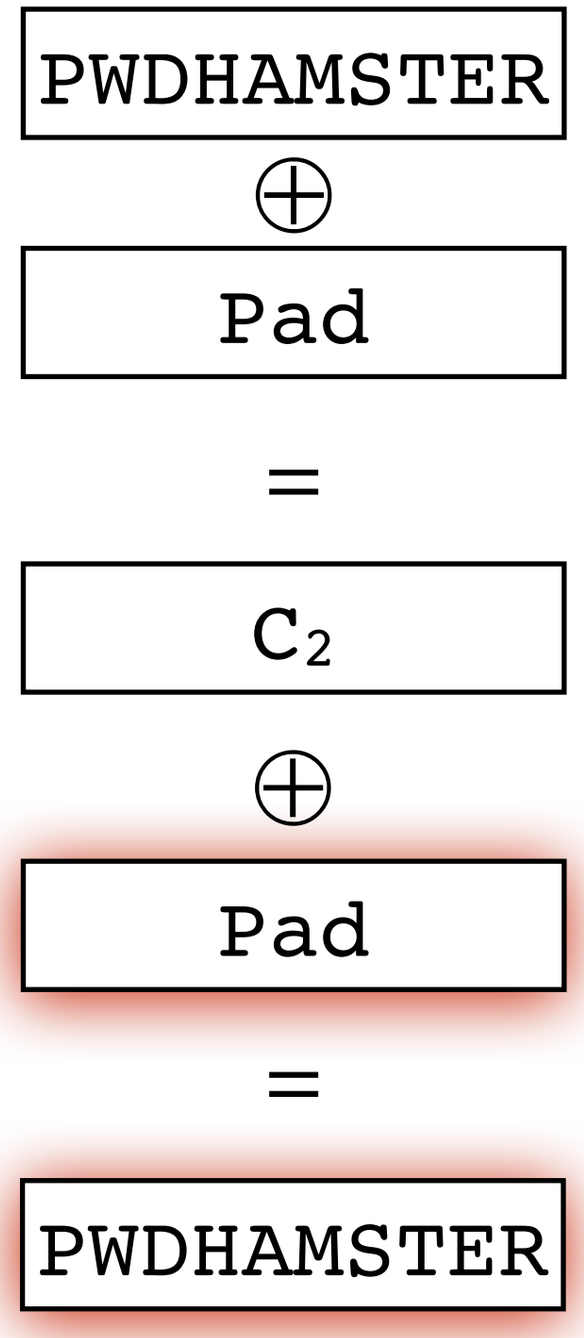
1. Adversary goal: Learn partial information from plaintext
2. Adversary capability: Observe a single ciphertext
3. Adversary compute resources: Unlimited time/memory (!)

Three Issues with One-Time Pad

1. Reusing a pad is insecure
2. One-Time Pad has a long key
3. One-Time Pad does not provide integrity/authenticity

Issue #1: Reusing a One-Time Pad is Insecure

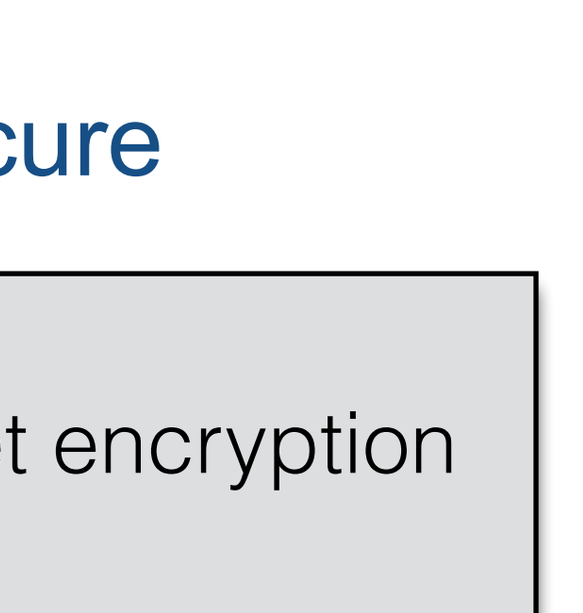




Issue #1: Reusing a One-Time Pad is Insecure

Has led to real attacks:
- Project Venona (1940s) attack by US on Soviet encryption
- MS Windows NT protocol PPTP
- WEP (old WiFi encryption protocol)
- Fortiguard routers! [link]
- Assignment 3

C_1 C_2



$C_1 \oplus C_2$

$= \texttt{S3CR3T1234} \oplus \texttt{3L33THXRRR}$

Issue #2: One-Time Pad Needs a Long Key

- The following is formalized and proved in the crypto class:

Theorem: Any encryption as secure as the OTP must have

Key-length ≥ Plaintext-length

- If plaintext is long, then the key must be long.
- Not realistic in practice to share long keys and use them once

Outline of Lecture 5

1. Encryption Basics
2. One-Time Pad Encryption
3. **Practical Tool #1: Stream Ciphers**
4. Practical Tool #2: Blockciphers
5. Message Authentication
6. Hash Functions

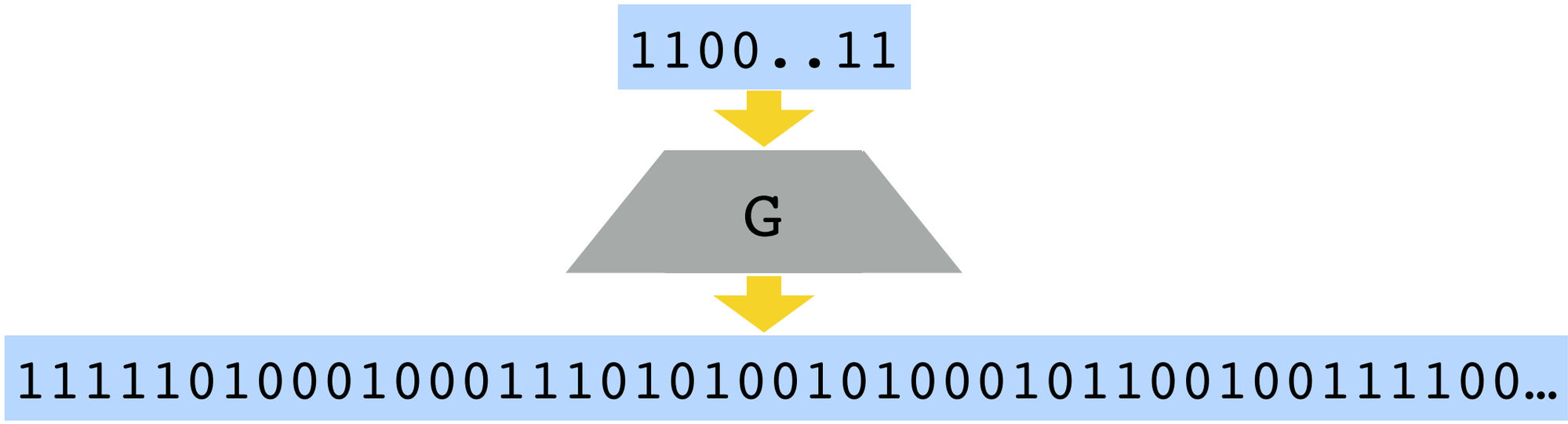
Stream Ciphers: Stretching Keys for the OTP

Key Idea: Starting from a short random key **K**, compute a very long pseudorandom string and use that with OTP.

- The practical tool for doing this is called a stream cipher.

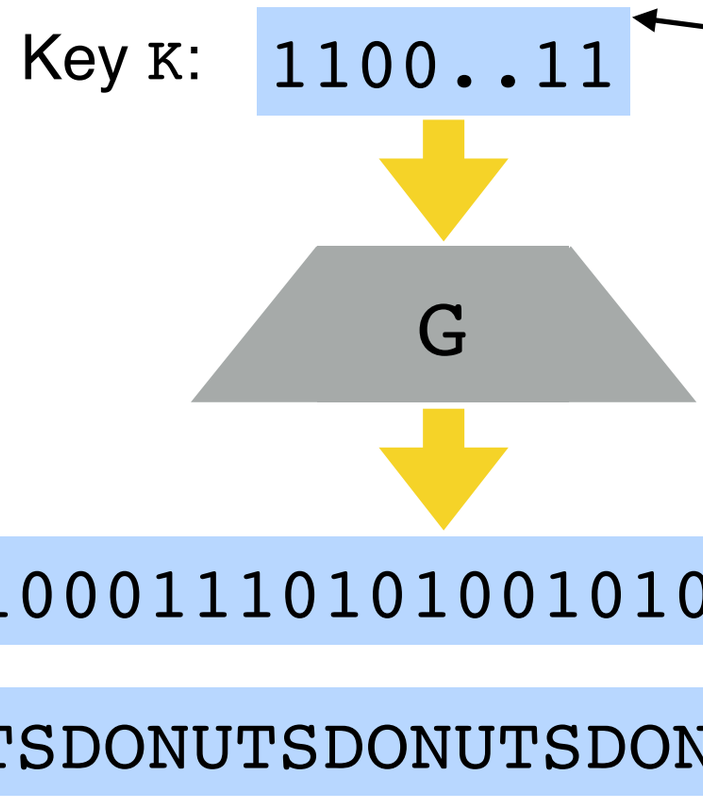
Definition: A stream cipher is an algorithm **G** such that:

1. **G** takes one short input string (16 or 32 bytes)
2. **G** produces a gigantic output string (petabytes if needed)
3. When **K** is chosen at random, **G(K)** looks random to someone who does not know **K**.



Encrypting with a Stream Cipher: The Pseudo-OTP

Usually very, very large (petabytes if needed)

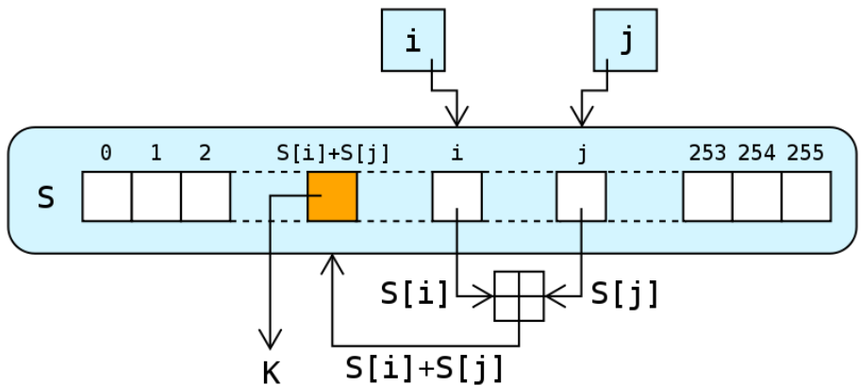


Use **G(K)** in place of pad.

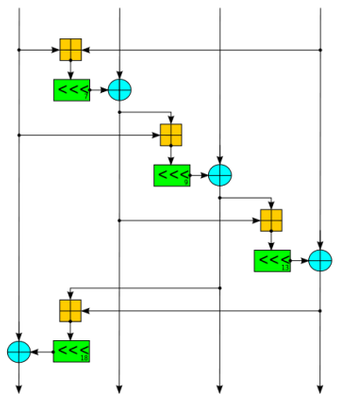
- Can now encrypt messages much longer than the key

Practical Stream Ciphers (Not covered in this class)

RC4 (1987)**:** “Ron’s Cipher #4”. Mostly retired by 2016.



ChaCha20 (2007)**:** Successfully deployed replacement.



- They both run very quickly (e.g. 10 cycles per output byte)
- You should never need to write your own implementation

Stream Cipher Security: Pseudorandomness

- Security goal: When `K` is random and unknown, `G(K)` should "look" random, even to an adversary spending **a lot** of computation.

- Basic properties `G` must have:
  - Outputs must have about the same number of 0s and 1s
  - Outputs should have each possible byte about 1/256th of time
  - ...

- But those are simple properties. We require resistance to extremely clever and well-resourced statistical tests, like brute force attacks:

```
BruteForceAttack(y):
  For every possible key K:
    y' = G(K)
    If y'==y: guess "not random"
```

- Clarified goal: When `K` is random and unknown, `G(K)` should "look" random to anyone with less computational power needed for a brute force attack.

(Non-secure) Pseudorandom Generators vs Stream Ciphers

- The pseudorandom generators are frequently used in “random” packages
  - Example: Mersenne twister in the **`python random`** package
- These algorithms pass lots of statistical tests ....
- ... but they are **completely insecure** when used as a stream cipher

Aside: Fundamental Physical Property of the Universe*

There exist functions (say on bitstrings) that are:
1) Very fast to evaluate
2) Invertible, but computationally infeasible to invert

The disparity can be almost arbitrarily large!

Evaluating **y = f(x)** may only take a few cycles….

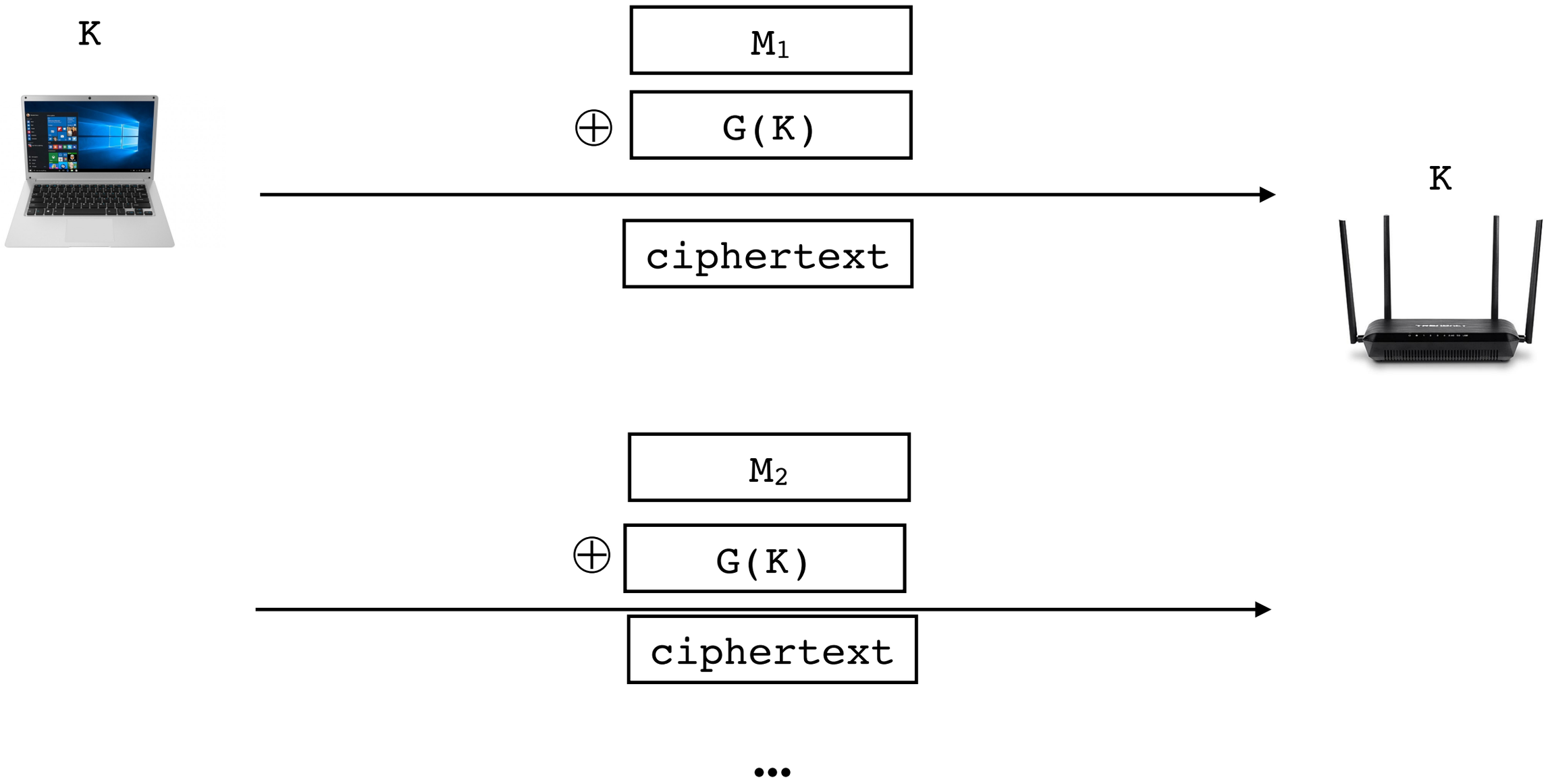
… and finding **x** from **y** within the lifetime of the universe may not be possible, even with a computer made up of every particle in the universe.

**depending on who you ask*

Stream Ciphers: Summary

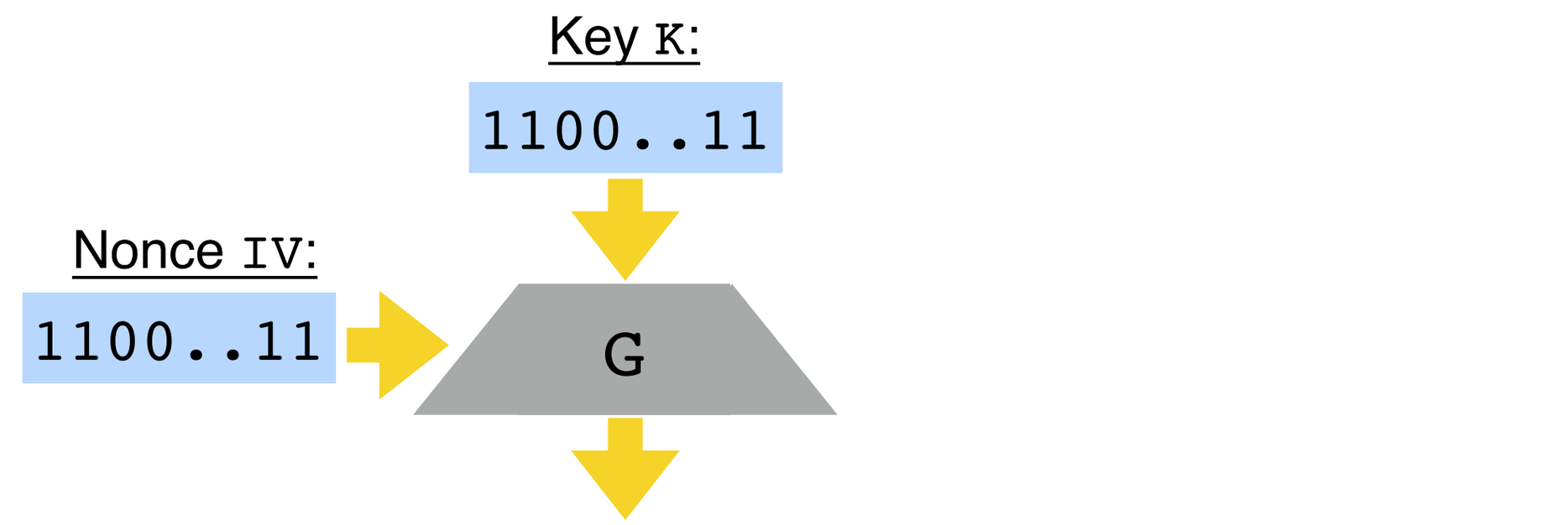
- Stream ciphers like **`chacha20`** are extremely efficient to evaluate and have very strong pseudorandomness properties
- We can get a “secure enough” version of the OTP by using **`G(K)`** instead of **`K`** as the pad

Pad reuse can still happen with stream ciphers



Addressing pad reuse: Stream cipher with a nonce

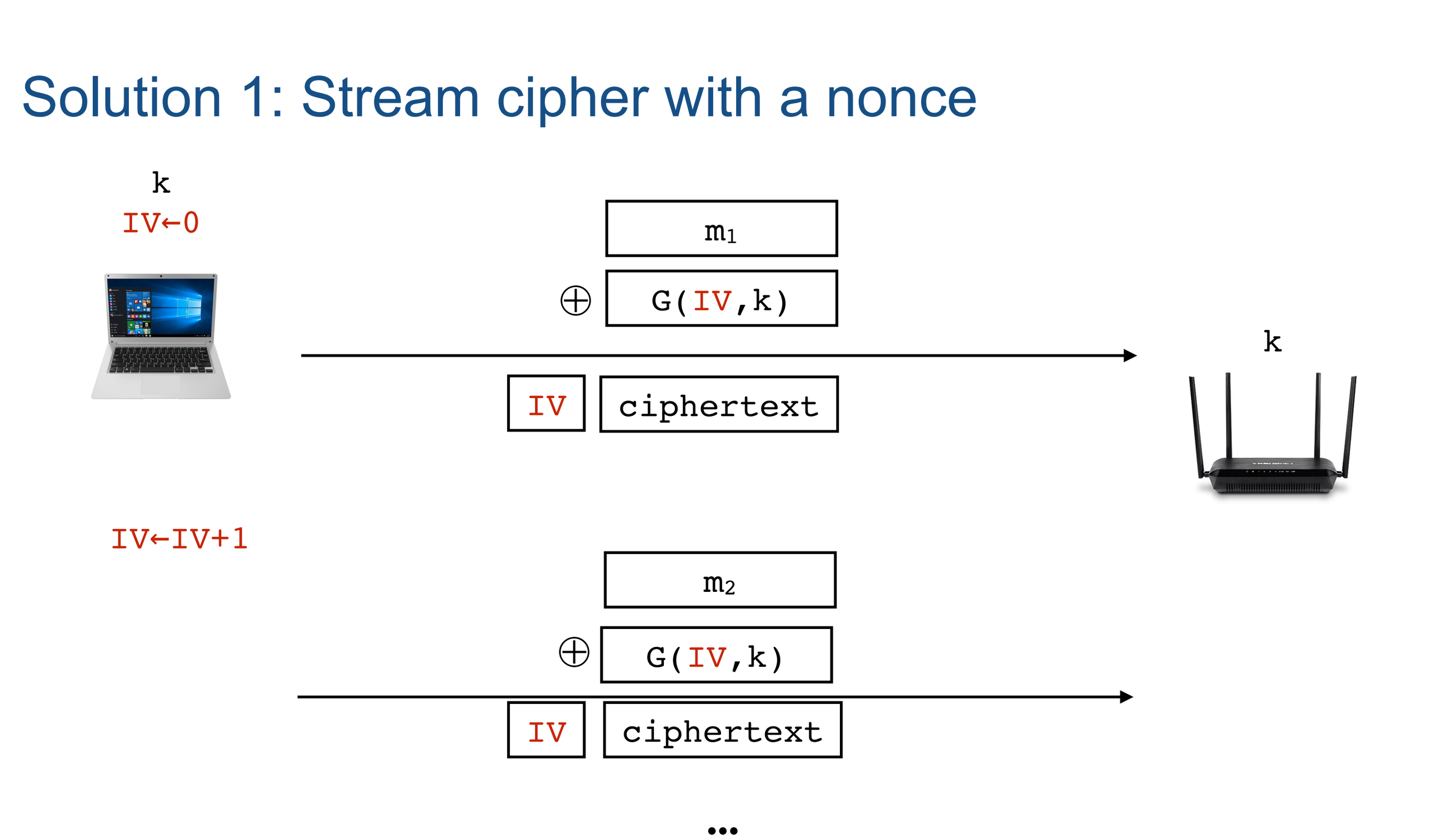
Stream cipher with a nonce: Algorithm `G` that takes **two inputs** and produces a very long bit-string as output.



- "nonce" = "number once".
- Often denoted `IV` = "initialization vector"
- `IV` is short (e.g. 12 bytes)

Security goal: When `K` is random and unknown, `G(IV,K)` should "look" random and independent for each value of `IV`.

Solution 1: Stream cipher with a nonce

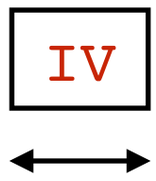


- If nonce repeats, then pad repeats
- Random nonces also possible, but allow for repeats with probability that must be carefully estimated

Example of Pad Re-use: WEP

Warning: Broken

IEEE 802.11b WEP: WiFi security standard ’97-‘03



`IV` is 24-bit wide counter

- Repeats after 2^{24} frames (≈16 million)
- `IV` is often set to zero on power cycle

Solutions: (WPA2 replacement)

- Larger `IV` space, rekeys more often
- Set `IV` to combination of packet number, address, etc

Example of Pad Re-use: WEP

Warning: Broken

IEEE 802.11b WEP: WiFi security standard '97-'03

- Re
- IV

ars TECHNICA

BIZ & IT TECH SCIENCE POLICY CARS GAMING & CULTURE FORUMS

BIZ & IT —

Serious flaw in WPA2 protocol lets attackers intercept passwords and much more

KRACK attack is especially bad news for Android and Linux users.

DAN GOODIN - 10/15/2017, 11:37 PM

Solutions: (W

- Larger IV s

parameters to their initial values. KRACK forces the nonce reuse in a way that allows the encryption to be bypassed. Ars Technica IT editor Sean Gallagher has much more about KRACK here.

- Set IV to combination of packet number, address, etc

Three Issues with One-Time Pad

1. Reusing a pad is insecure ✓ *Use unique nonces*
2. One-Time Pad has a long key ✓ *Use stream cipher with short key*
3. One-Time Pad does not provide integrity/authenticity

Outline of Lecture 5

1. Encryption Basics
2. One-Time Pad Encryption
3. Practical Tool #1: Stream Ciphers
4. **Practical Tool #2: Blockciphers**
5. Message Authentication
6. Hash Functions

Block Ciphers

Block ciphers are a common crypto tool applied to many problems.

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

Typical parameters:

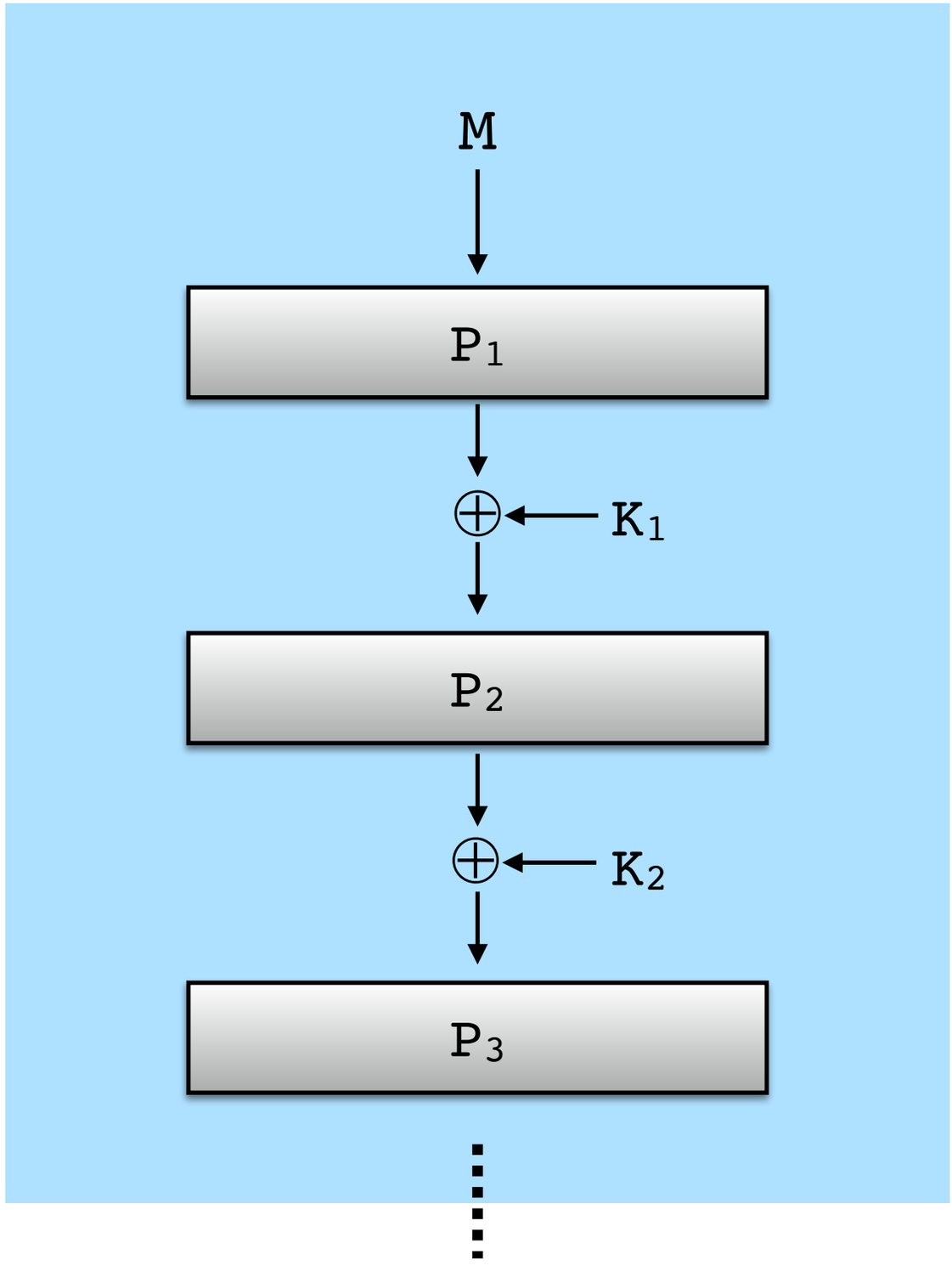
Alphabet = $\{0,1\}^{128}$ (16 bytes long)
Key length = 16 bytes.

Can build many higher-level protocols from a good blockcipher

Advanced Encryption Standard (AES)

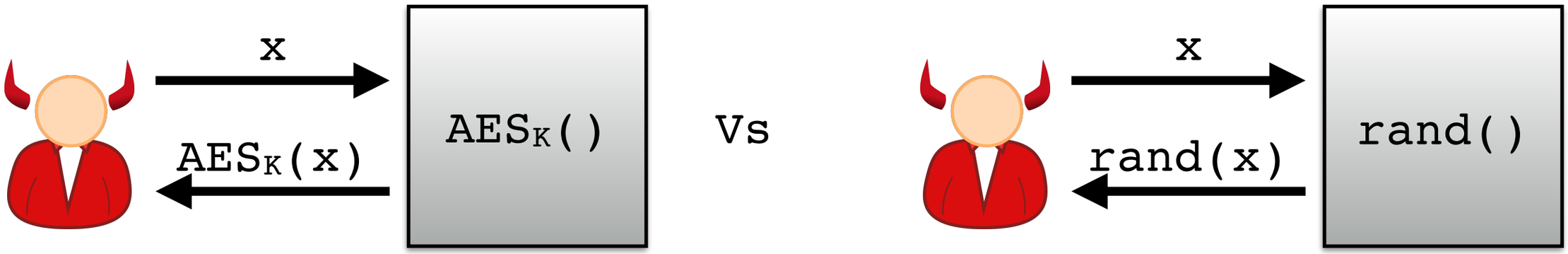
- NIST ran competition to develop standard starting in 1997
- From several submissions, *Rijndael* chosen and standardized as AES
- Very fast; Modern chips have AES instructions

- Block length n = 128
- Key length k = 128,192,256



Blockcipher Security

- 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.

Example - AES Input/Outputs

- Keys and inputs are 16 bytes = 128 bits

```
-K1: 9500924ad9d1b7a28391887d95fcfbd5
-K2: 9500924ad9d1b7a28391887d95fcfbd6
```

$AES_{K1}(00..00)$=8b805ddb39f3eee72b43bf95c9ce410f
$AES_{K1}(00..01)$=9918e60f2a20b1b81674646dceebdb51
$AES_{K2}(00..00)$=1303270be48ce8b8dd8316fdba38eb04
$AES_{K2}(00..01)$=96ba598a55873ec1286af646073e36f6

So we have a blockcipher…

- Now what?

  It only processes 16 bytes at a time, and I have a whole lot more data than that.

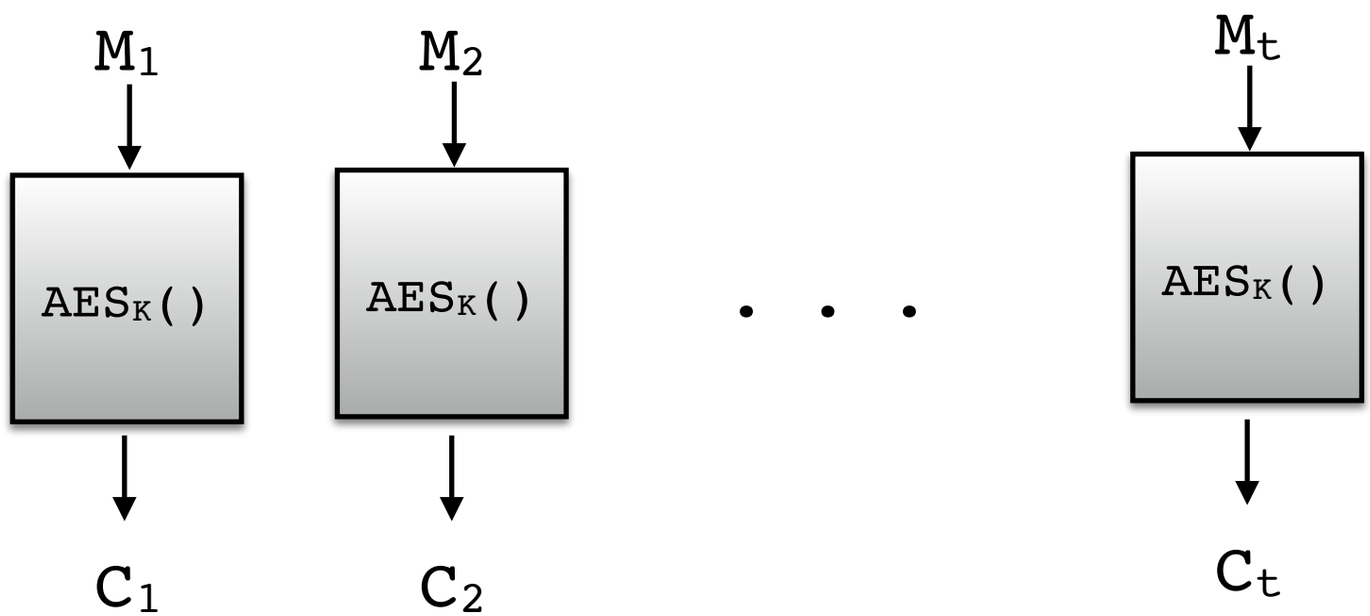
Encrypting large files: ECB

Warning: Broken

- ECB = “Electronic Code Book”

AES-ECB$_k$(M)

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



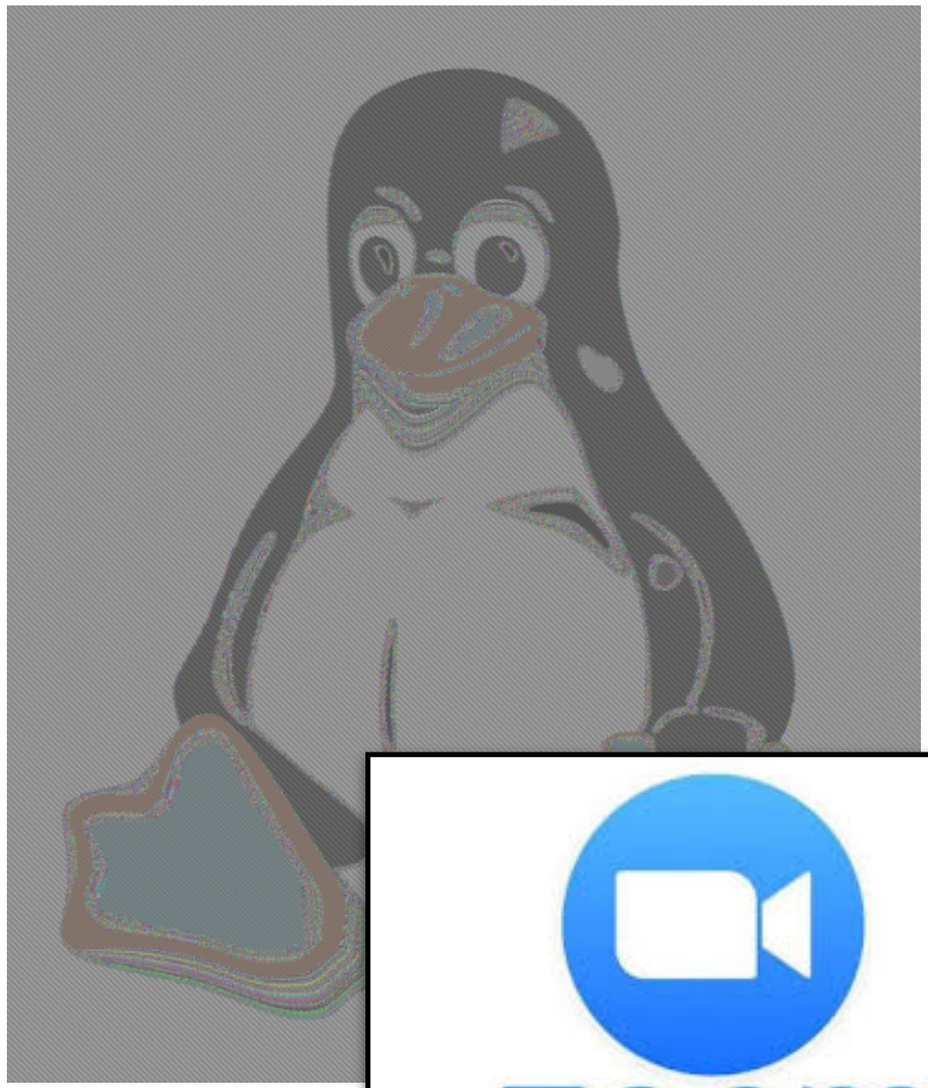
The ECB Penguin

Warning: Broken

- Treat pixels as one long string and encrypt

Plaintext

ECB Ciphertext

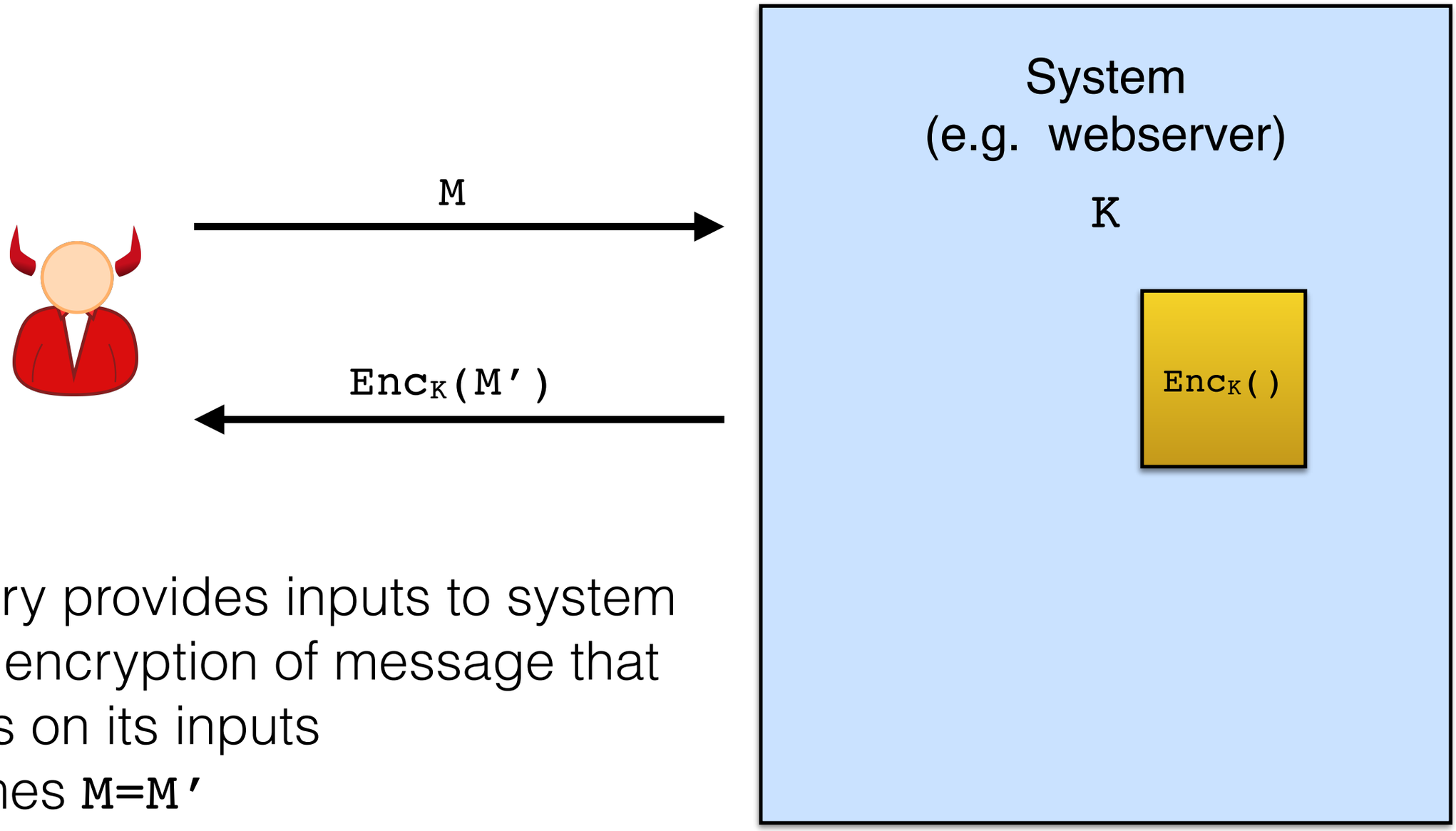


- It gets even worse…

ECB Security: It gets worse…

- Seeing penguins is bad, but it doesn't mean you can recover credit card numbers or passwords inside a ciphertext
- "Chosen Plaintext Attack" against ECB can often decrypt *any* ciphertext.

Chosen-Plaintext Attacks (CPA) against Encryption

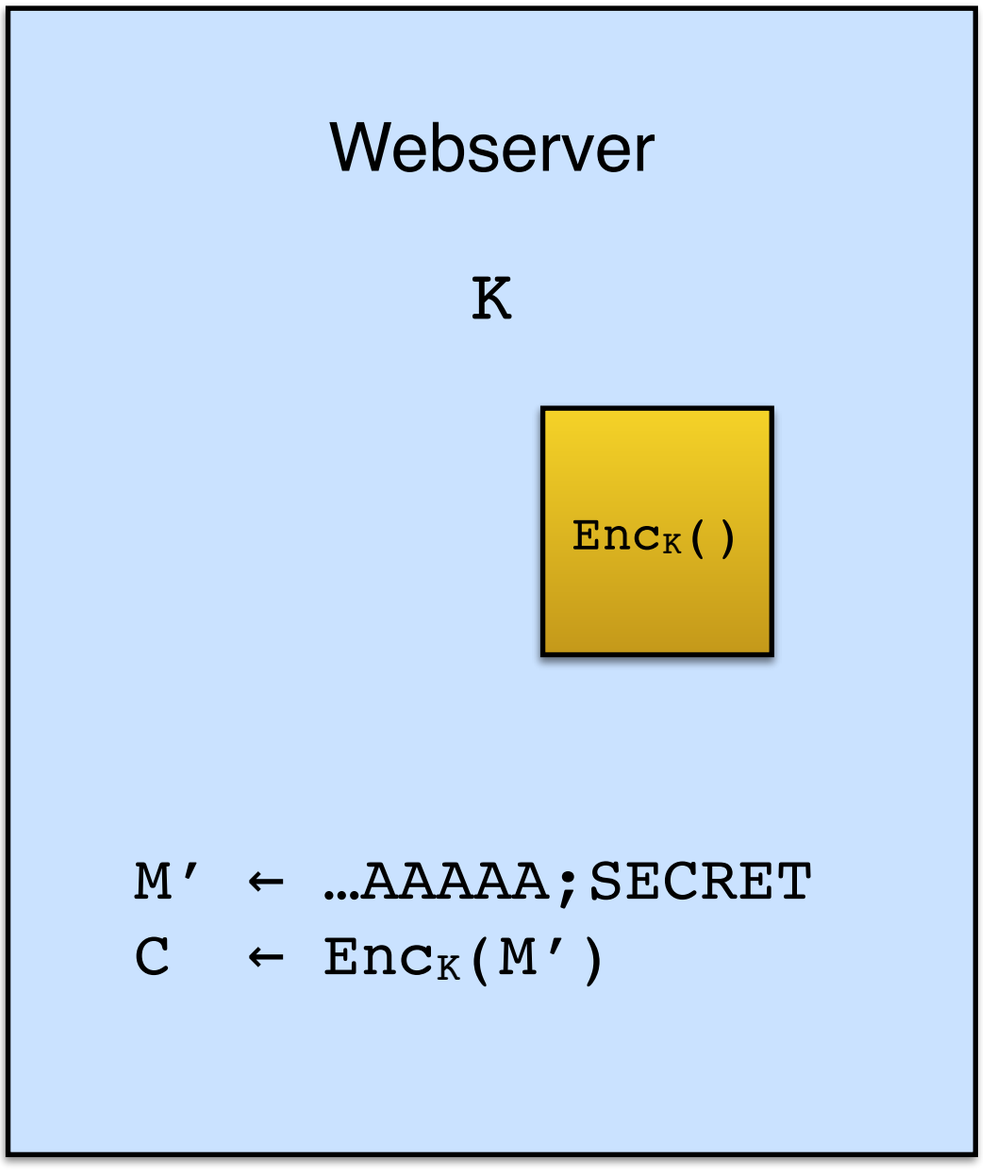


- Adversary provides inputs to system
- Obtains encryption of message that depends on its inputs
- Sometimes `M=M'`

CPA Example: Encrypted Cookies

"username = AAAAA"

C



- More later in web security module

Assignment 3!

AES-CTR Mode: A Secure Approach

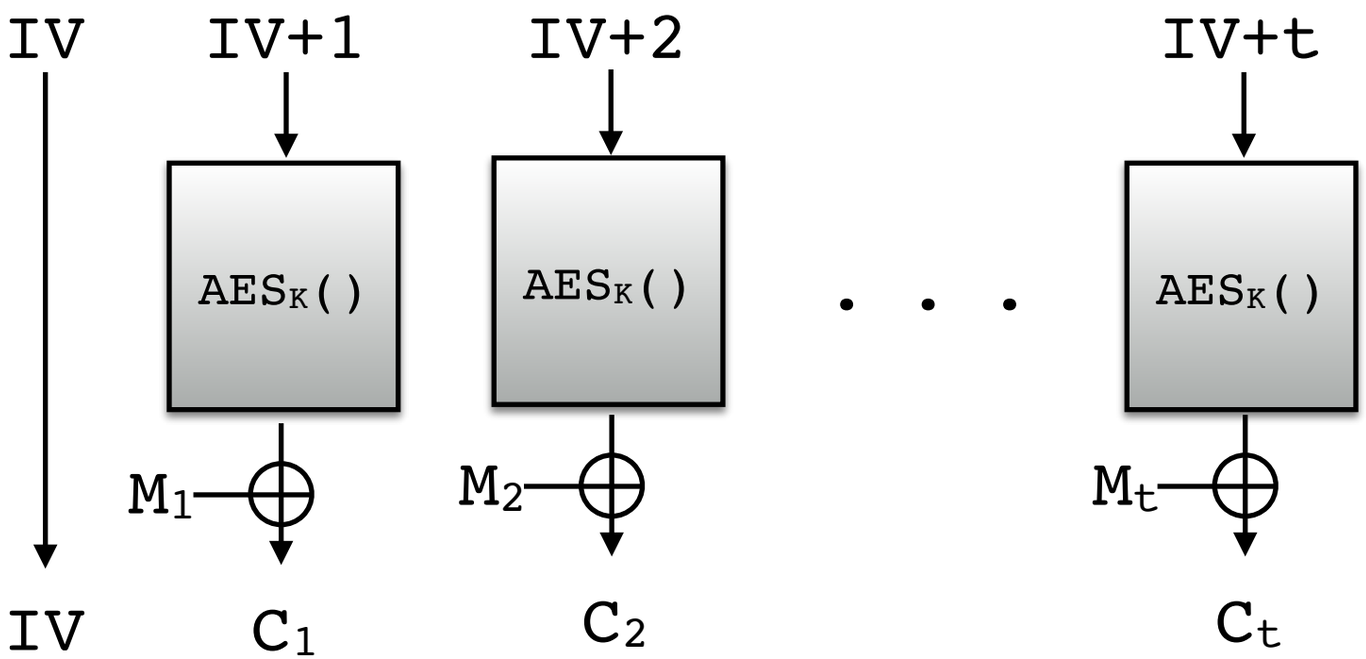
- CTR = “Counter Mode”
- Idea: Build a nonce-based stream cipher from AES

AES-CTR$_k$(IV,M)

- Parse M into blocks $M_1, M_2, \ldots, M_t$
  // all blocks except M_t are 16 bytes
- For i=1…t:
  - $C_i \leftarrow M_i \oplus AES_k(IV+i)$
- Return IV, $C_1, \ldots, C_t$

Notes:

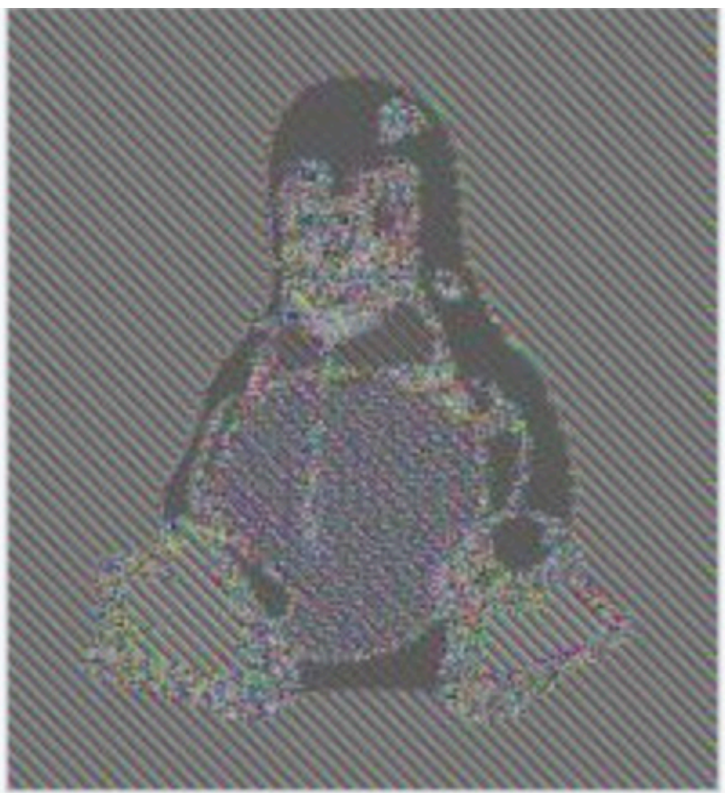
- No need to pad last block
- Must avoid reusing part of stream



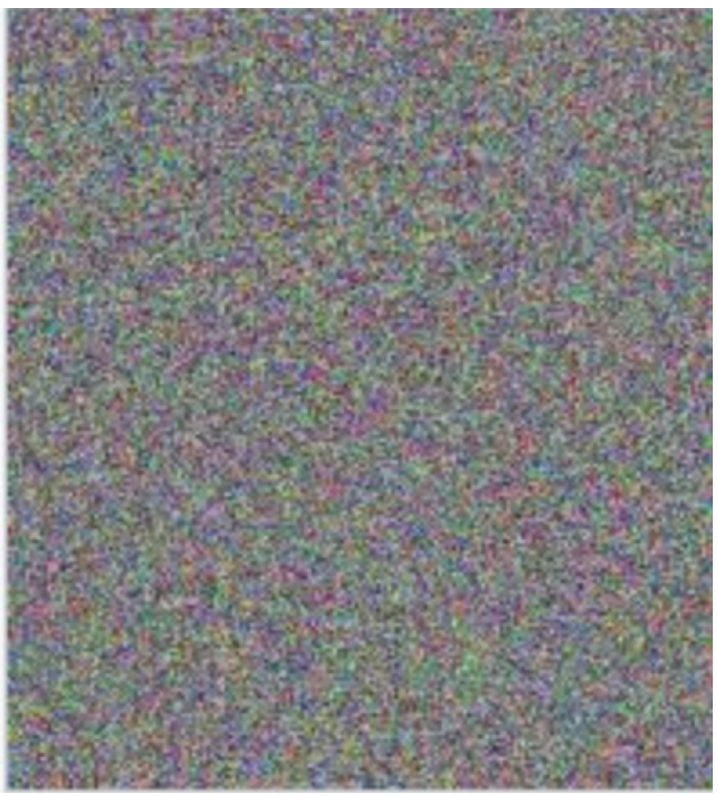
Penguin Sanity Check

Plaintext

ECB Ciphertext



CTR Ciphertext



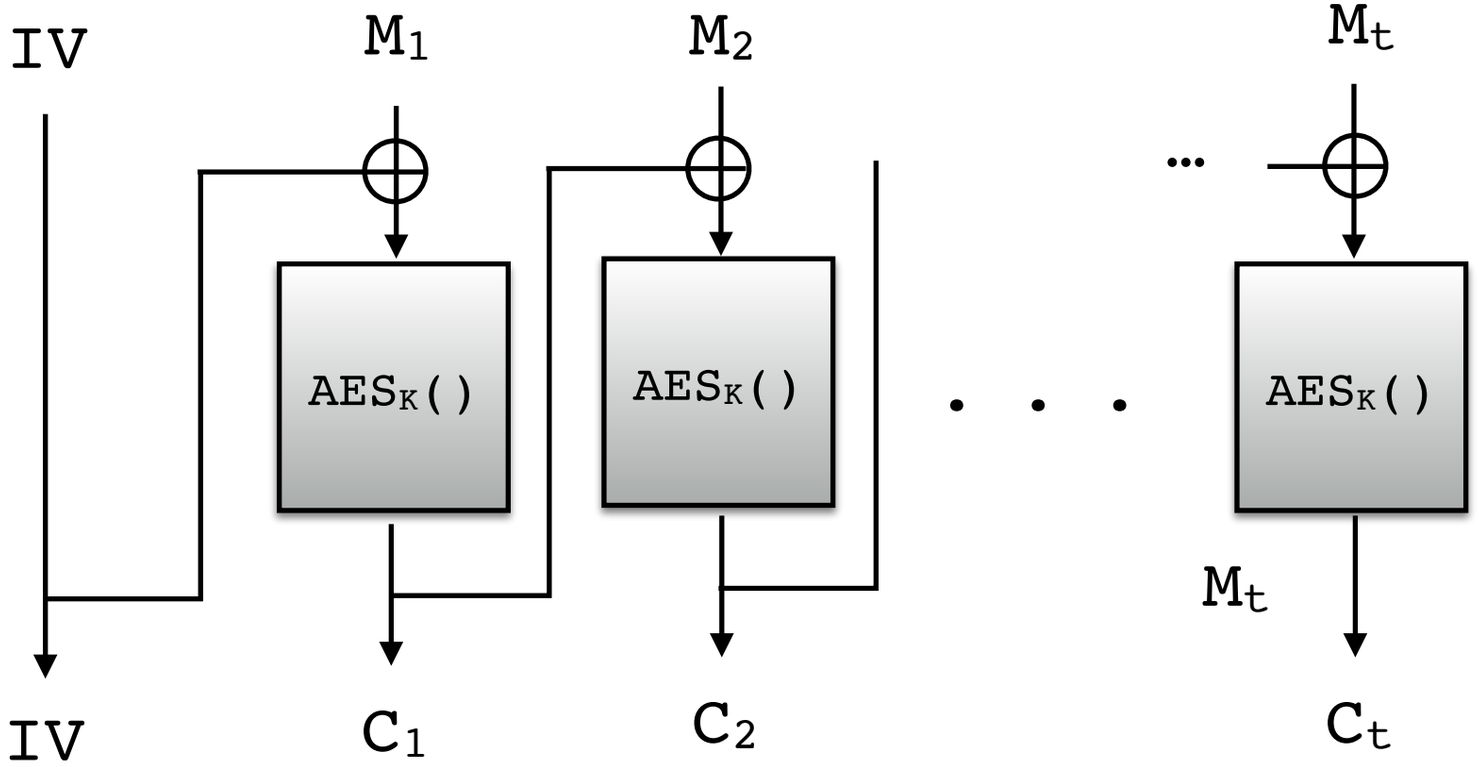
Looks random

Encrypting large files, Attempt #3: CBC

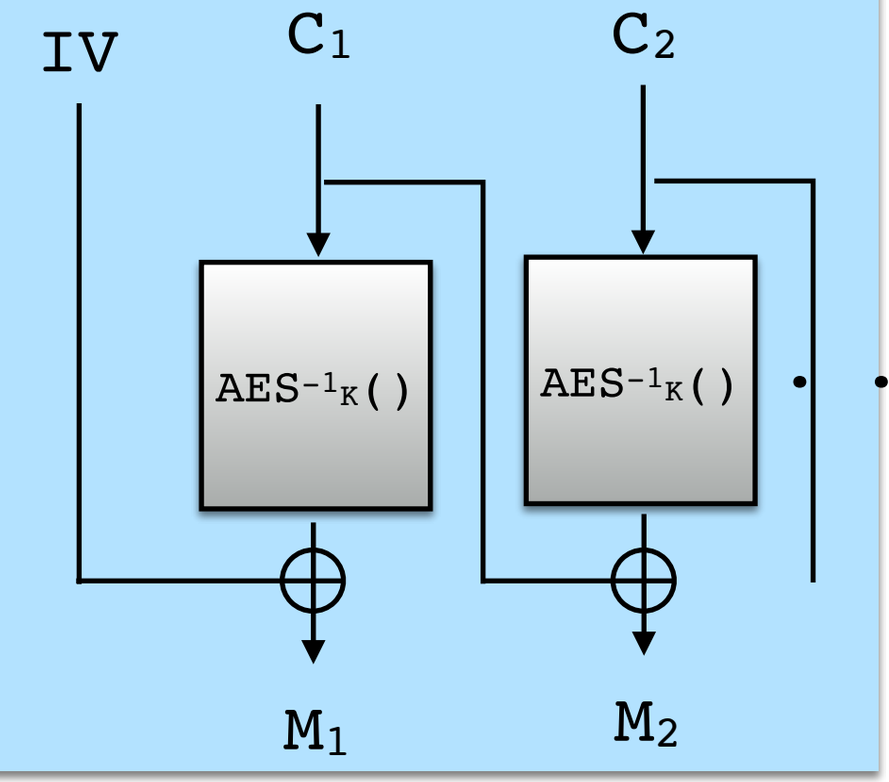
- CBC = “Cipher Block Chaining”
- Nonce-based, but not a stream cipher
- Historical option (sometimes used without nonce)

AES-CBC$_k$(IV,M)

- Parse M into blocks $M_1, M_2, \ldots, M_t$
  // all blocks except M_t are 16 bytes
- Pad M_t up to 16 bytes
- $C_0 \leftarrow IV$
- For i=1…t:
  - $C_i \leftarrow AES_k(M_i \oplus C_{i-1})$
- Return $C_0, C_1, \ldots, C_t$



Decryption

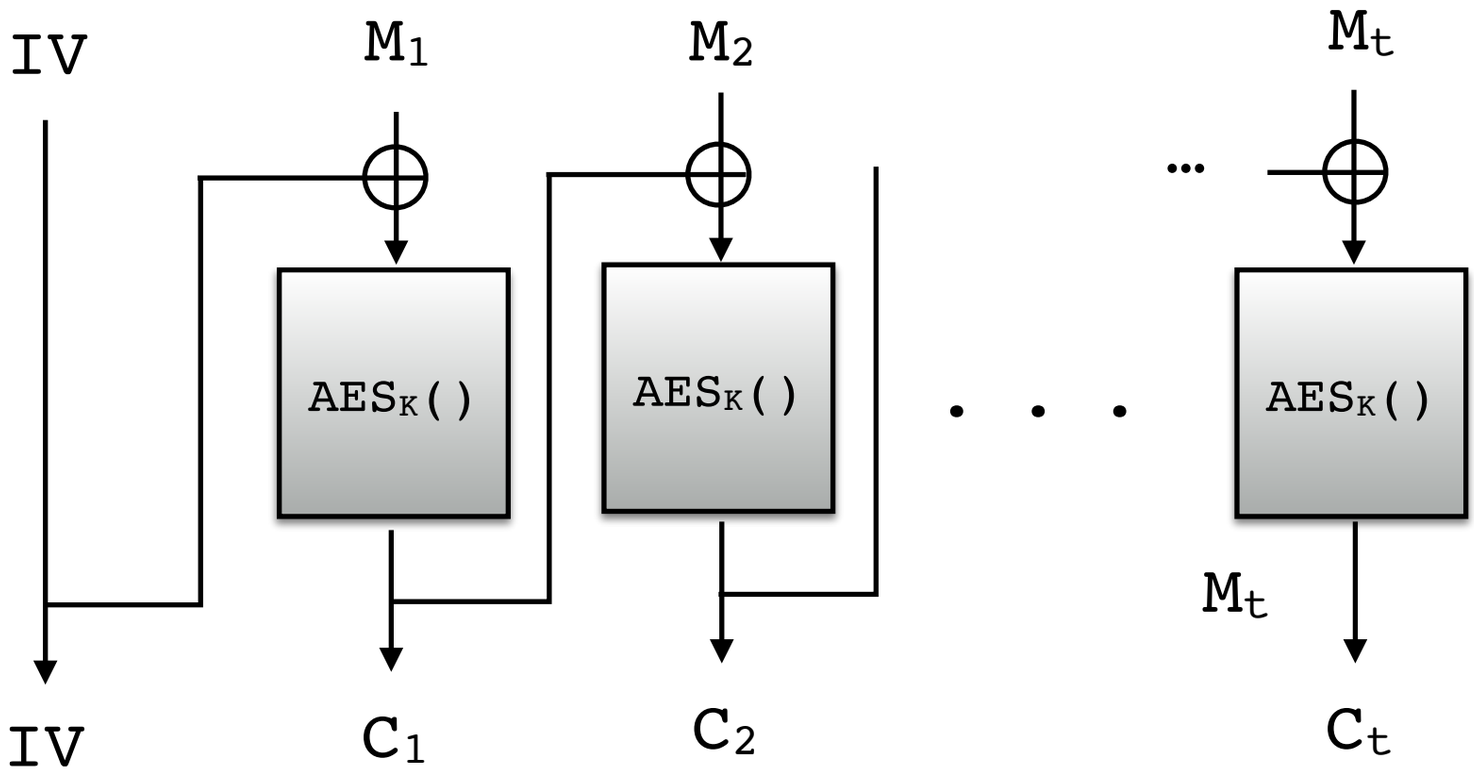


Encrypting large files, Attempt #3: CBC

- CBC = “Cipher Block Chaining”
- Nonce-based, but not a stream cipher
- Historical option (sometimes used without nonce)

AES-CBC$_k$(IV,M)

- Parse M into blocks $M_1, M_2, \ldots, M_t$
  // all blocks except M_t are 16 bytes
- Pad M_t up to 16 bytes
- $C_0 \leftarrow IV$
- For i=1…t:
  - $C_i \leftarrow AES_k(M_i \oplus C_{i-1})$
- Return $C_0, C_1, \ldots, C_t$



Warning: Padding creates havoc with authentication. Very difficult to implement.

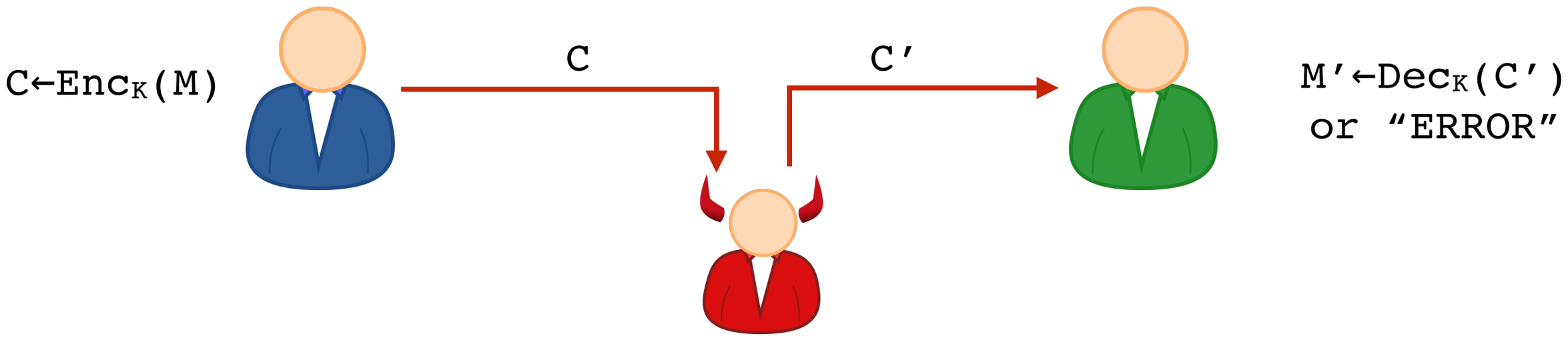
Blockcipher Encryption Summary

- AES is unbroken
- AES-CTR is most robust construction for confidentiality
- AES-CTR/AES-CBC do not provide authenticity/integrity and should almost never be used alone.

Outline of Lecture 5

1. Encryption Basics
2. One-Time Pad Encryption
3. Practical Tool #1: Stream Ciphers
4. Practical Tool #2: Blockciphers
5. **Message Authentication**
6. Hash Functions

Encryption Integrity: An abstract setting



Encryption satisfies **integrity** if it is infeasible for an adversary to send a new C' such that $\mathbf{Dec_K(C')\neq ERROR}$.

Stream ciphers do not give integrity

M = please pay ben 20 bucks

C = b0595fafd05df4a7d8a04ced2d1ec800d2daed851ff509b3e446a782871c2d

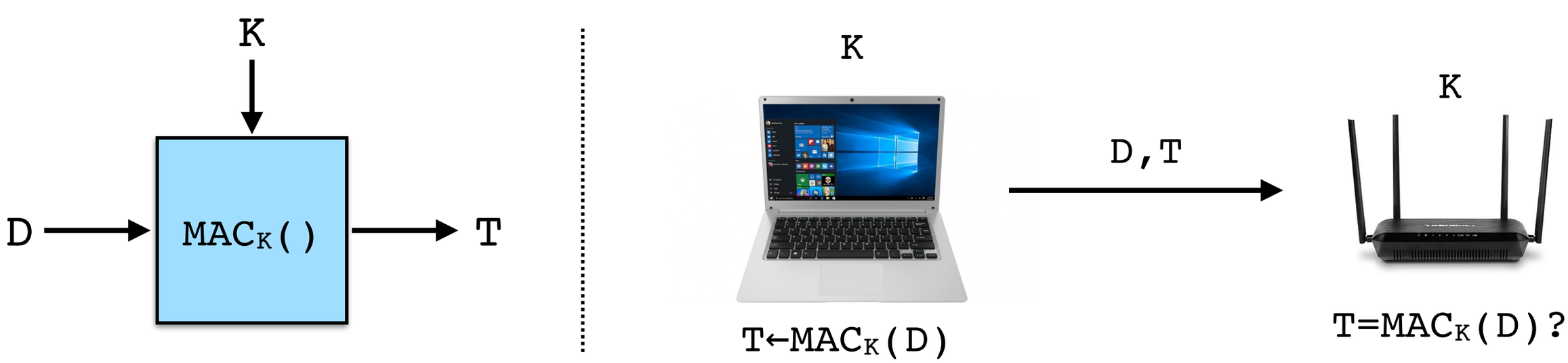
C’= b0595fafd05df4a7d8a04ced2d1ec800d2daed851ff509b3e546a782871c2d

M’ = please pay ben 21 bucks

Inherent to stream-cipher approach to encryption.

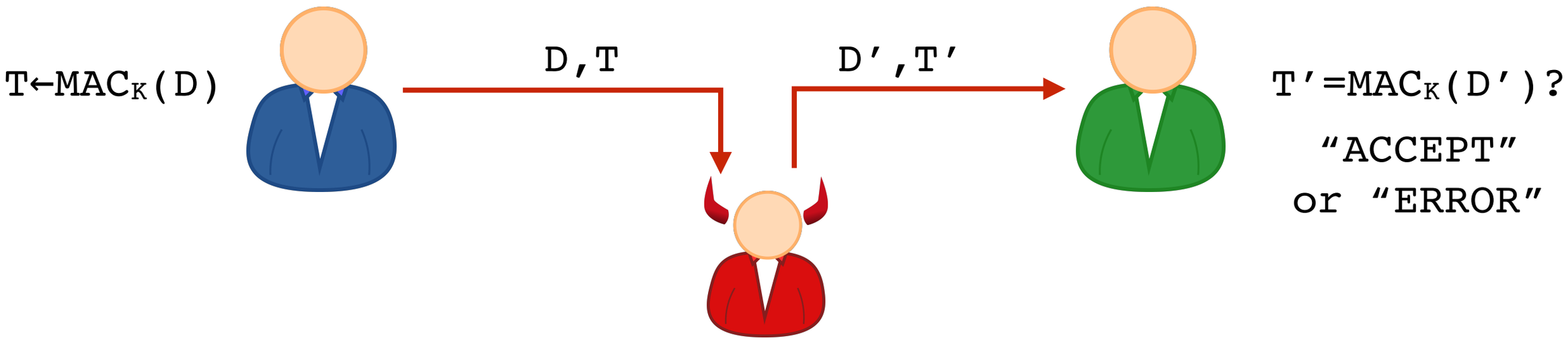
Message Authentication Code

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".
- Note: MACs are not block ciphers or encryption

MAC Security Goal: Unforgeability



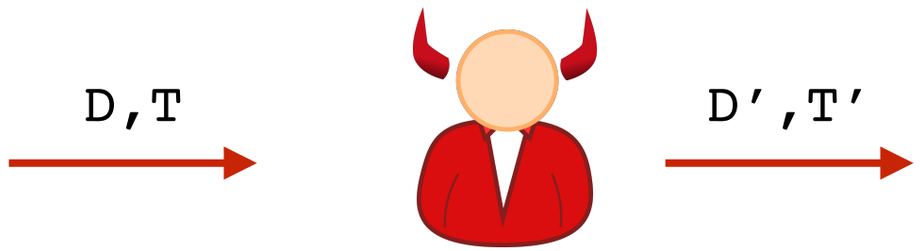
MAC satisfies **unforgeability** if it is infeasible for Adversary to fool Bob into accepting `D'` not previously sent by Alice.

MAC Security Goal: Unforgeability

Note: No encryption on this slide.

```
D = please pay ben 20 bucks

T = 827851dc9cf0f92ddcdc552572ffd8bc
```



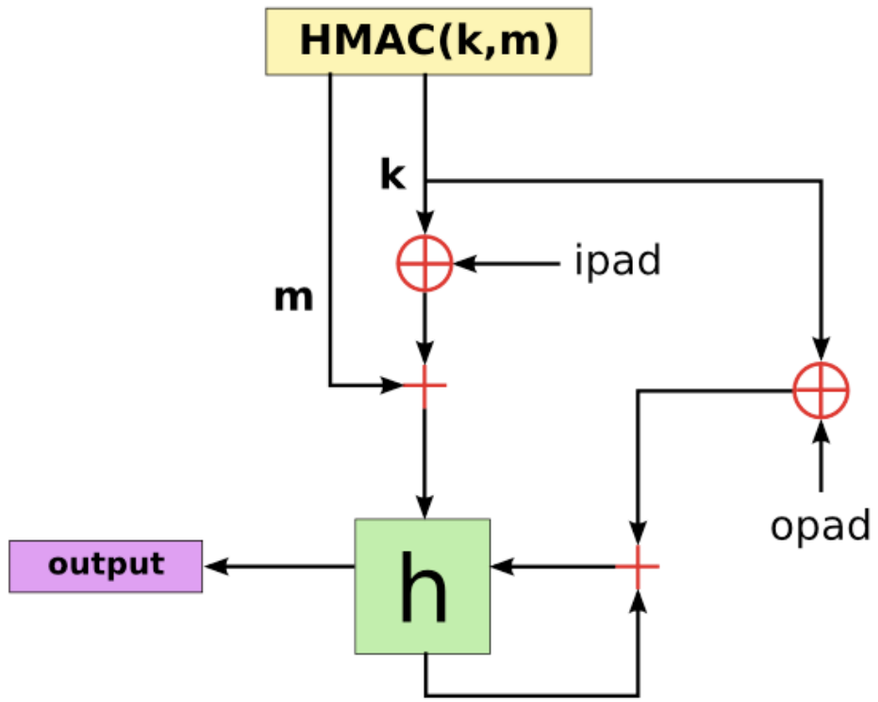

```
D’= please pay ben 21 bucks

T’= baeaf48a891de588ce588f8535ef58b6
```

Should be hard to predict **`T’`** for any new **`D’`**.

MACs In Practice: Use HMAC or Poly1305-AES

- More on how MACs are constructed in a moment.



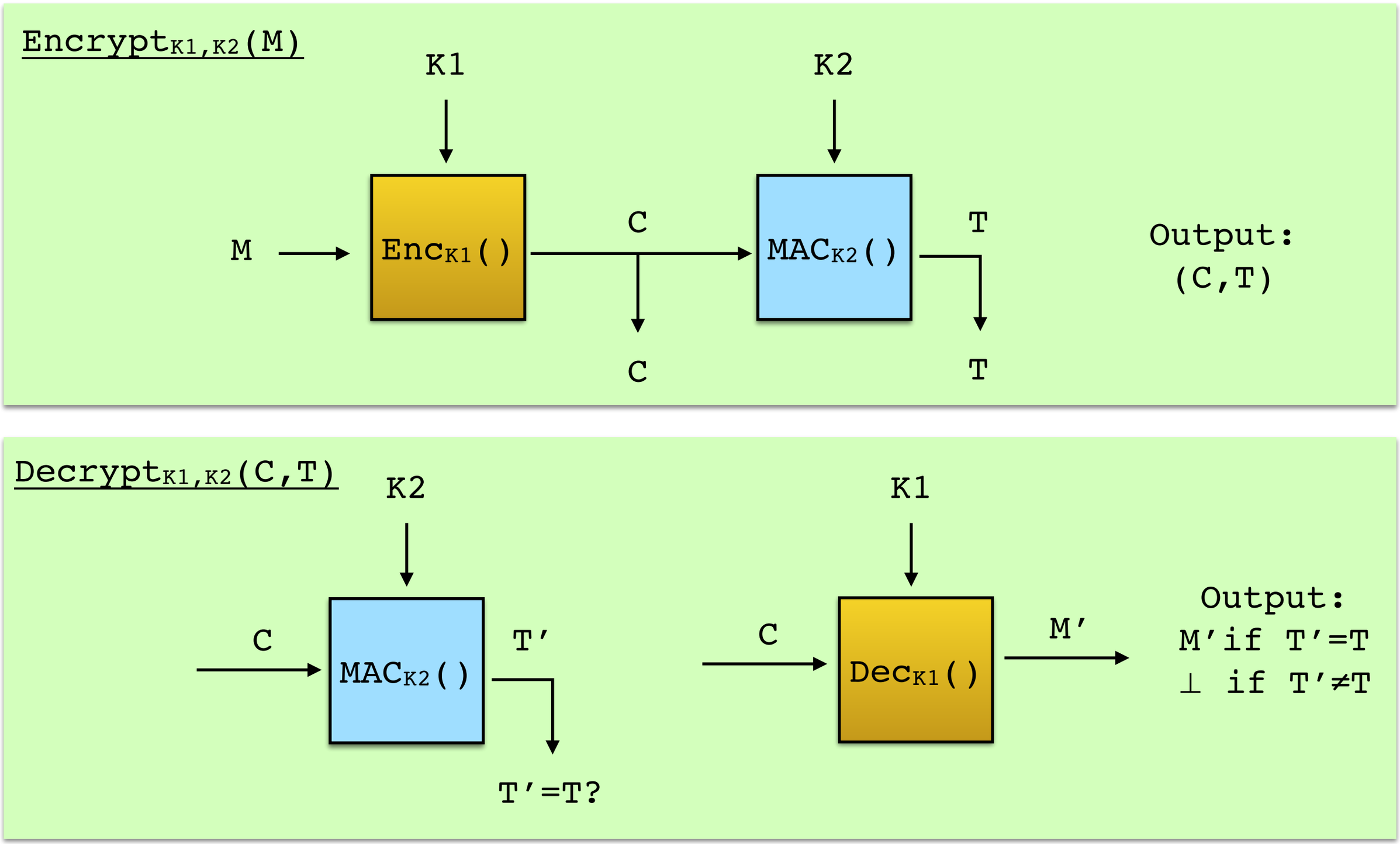
- Other, less-good option: AES-CBC-MAC (bug-prone)

Authenticated Encryption

Encryption that provides **confidentiality** and **integrity** is called **Authenticated Encryption**.

- Built using a good stream cipher and a MAC.
- Best solution: Use ready-made Authenticated Encryption
  - Examples:
    - Chacha20-Poly1305
    - AES-GCM

Building Authenticated Encryption



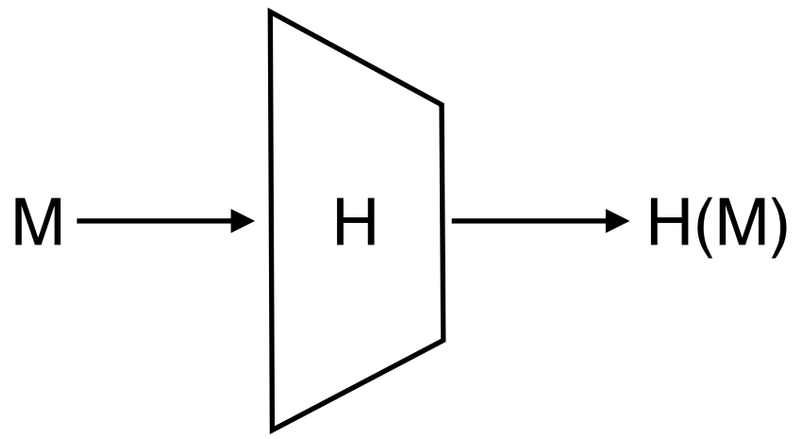
- Summary: Encrypt the message and then MAC the ciphertext

Outline of Lecture 5

1. Encryption Basics
2. One-Time Pad Encryption
3. Practical Tool #1: Stream Ciphers
4. Practical Tool #2: Blockciphers
5. Message Authentication
6. Hash Functions

Next Up: Hash Functions

Definition: A hash function is a deterministic function H that reduces arbitrary strings to fixed-length outputs.

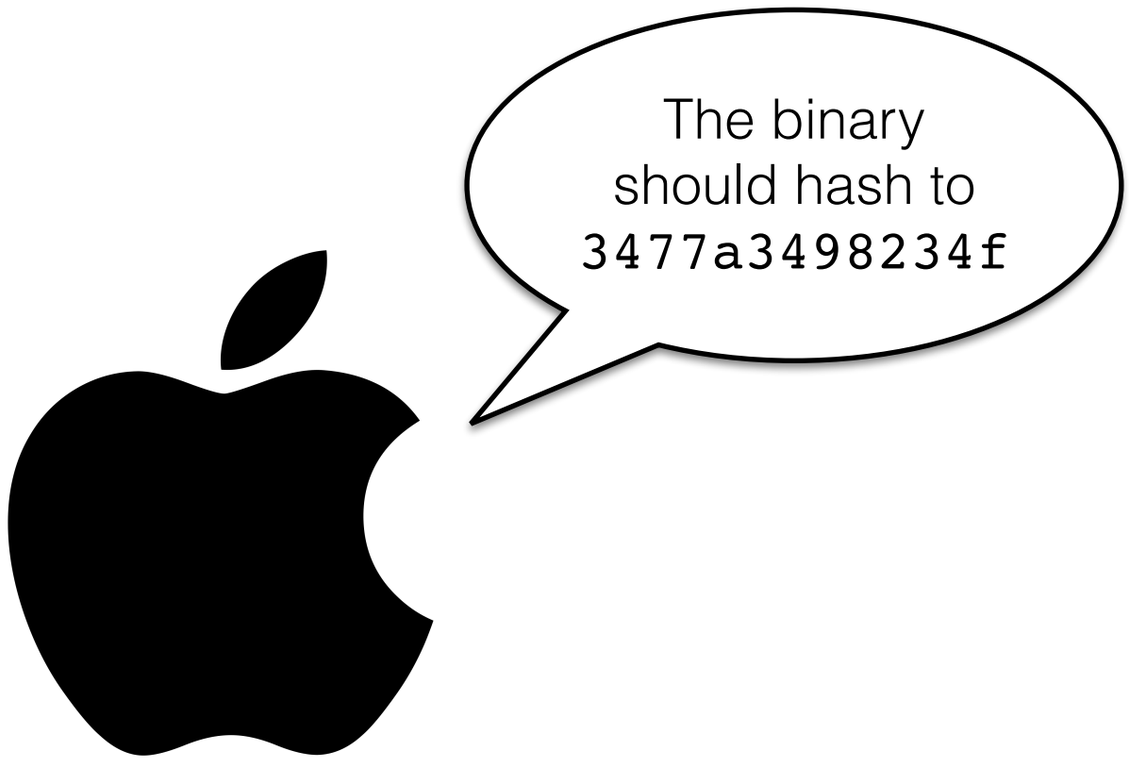


Primary security goal:

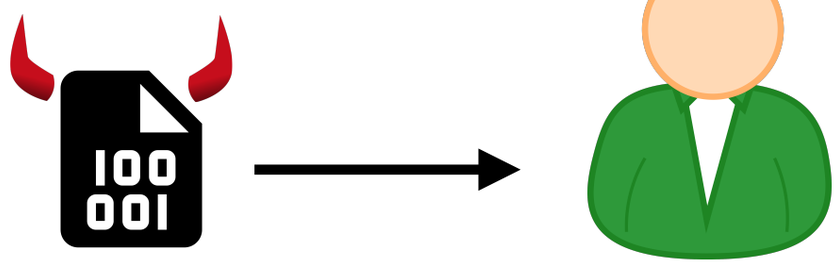
- **collision resistance**: can't find `M != M'` such that `H(M) = H(M')`

Note: Very different from hashes used in data structures!

Why are collisions bad?



MD5()=3477a3498234f



Hashes to
3477a3498234f,
so let's install!

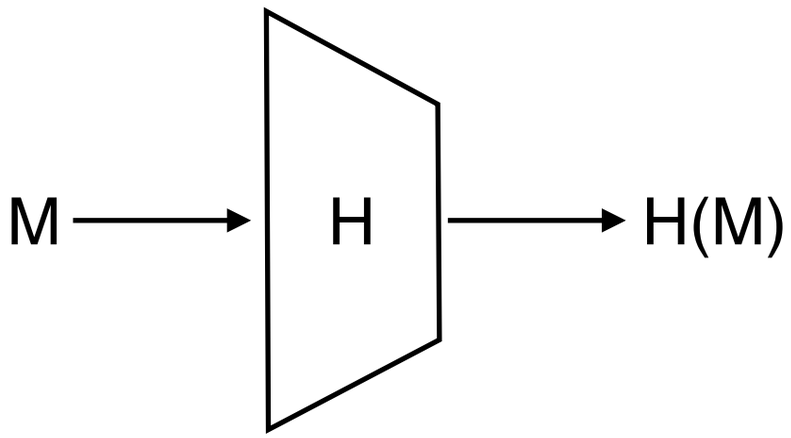
MD5()=3477a3498234f

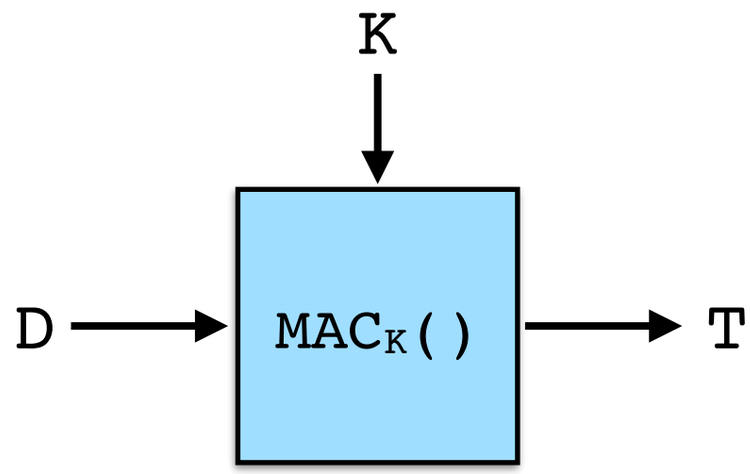
Practical Hash Functions

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

Confusion over “SHA” names leads to vulnerabilities.

Hash Functions are not MACs





Both map long inputs to short outputs… but a hash function does not take a key.

Intuition: a MAC is like a hash function, that only the holders of key can evaluate.

Hash Function Security History

- MD5 (1992) was broken in 2004 - can now find collisions very quickly.
- SHA-1 (1995) was broken in 2017 - A big computer can find collisions
- SHA-256/SHA-512 (2001) are not broken
- SHA-3 (2015) is new and not broken

MD5(d131dd02c5e6eec4693d9a0698aff95c 2fcab58712467eab4004583eb8fb7f89
55ad340609f4b30283e488832571415a 085125e8f7cdc99fd91dbdf280373c5b
d8823e3156348f5bae6dacd436c919c6 dd53e2b487da03fd02396306d248cda0
e99f33420f577ee8ce54b67080a80d1e c69821bcb6a8839396f9652b6ff72a70)

= MD5(d131dd02c5e6eec4693d9a0698aff95c 2fcab50712467eab4004583eb8fb7f89
55ad340609f4b30283e4888325f1415a 085125e8f7cdc99fd91dbd7280373c5b
d8823e3156348f5bae6dacd436c919c6 dd53e23487da03fd02396306d248cda0
e99f33420f577ee8ce54b67080280d1e c69821bcb6a8839396f965ab6ff72a70)

Xiaoyun Wang (Tsinghua University), 2004

MACs from Hash Functions

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

Construction: `MAC(K,D) = H(K+D)`

Warning: Broken

- Here, “+” means string concatenation
- Totally insecure if H = MD5, SHA1, SHA-256, SHA-512
- May be secure with SHA-3 (but don’t do it)

Construction: `MAC(K,D) = H(D+K)`

Just don’t

Upshot: Use HMAC; It’s designed to avoid this and other issues.

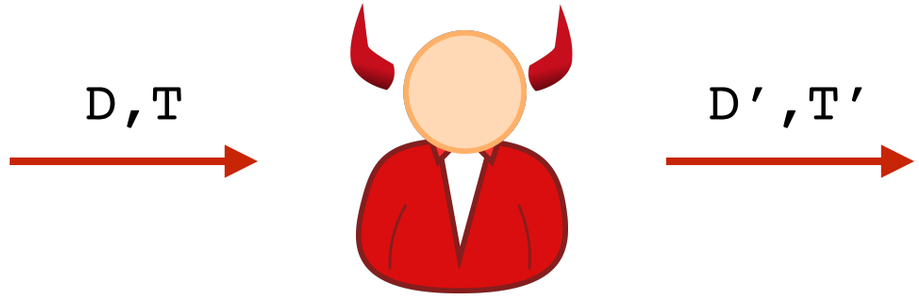
Later: Other applications of hash functions

Length Extension Attack

Construction: `MAC(K,D) = H(K+D)`

Warning: Broken

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



Need to find: Given `T=H(K+D)`, find `T'=H(K+D')` without knowing `K`.

In Assignment 3: Break this construction!

Review of Lecture 5

1. Encryption Basics
2. One-Time Pad Encryption
3. Practical Tool #1: Stream Ciphers
4. Practical Tool #2: Blockciphers
5. Message Authentication
6. Hash Functions

The End