Crypto
Part 2 of 3

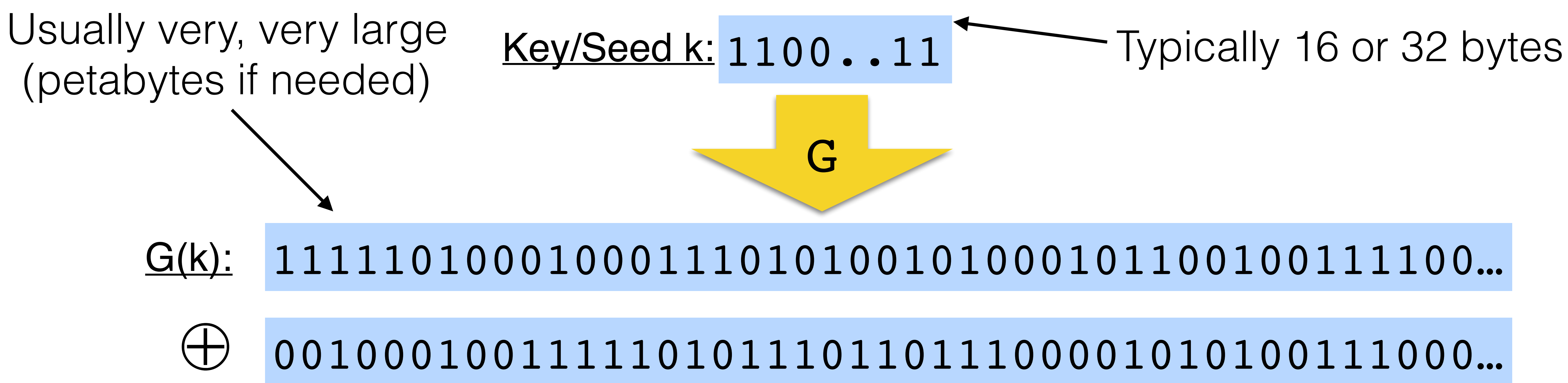
CMSC 23200/33250, Winter 2020, Lecture 4

David Cash & Blase Ur

University of Chicago

Tool to address key-length of OTP: Stream Ciphers

Stream cipher syntax: Algorithm G that takes one input and produces an very long bit-string as output.



Use G(seed) in place of pad.
Still malleable and still one-time, but key is shorter.

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.

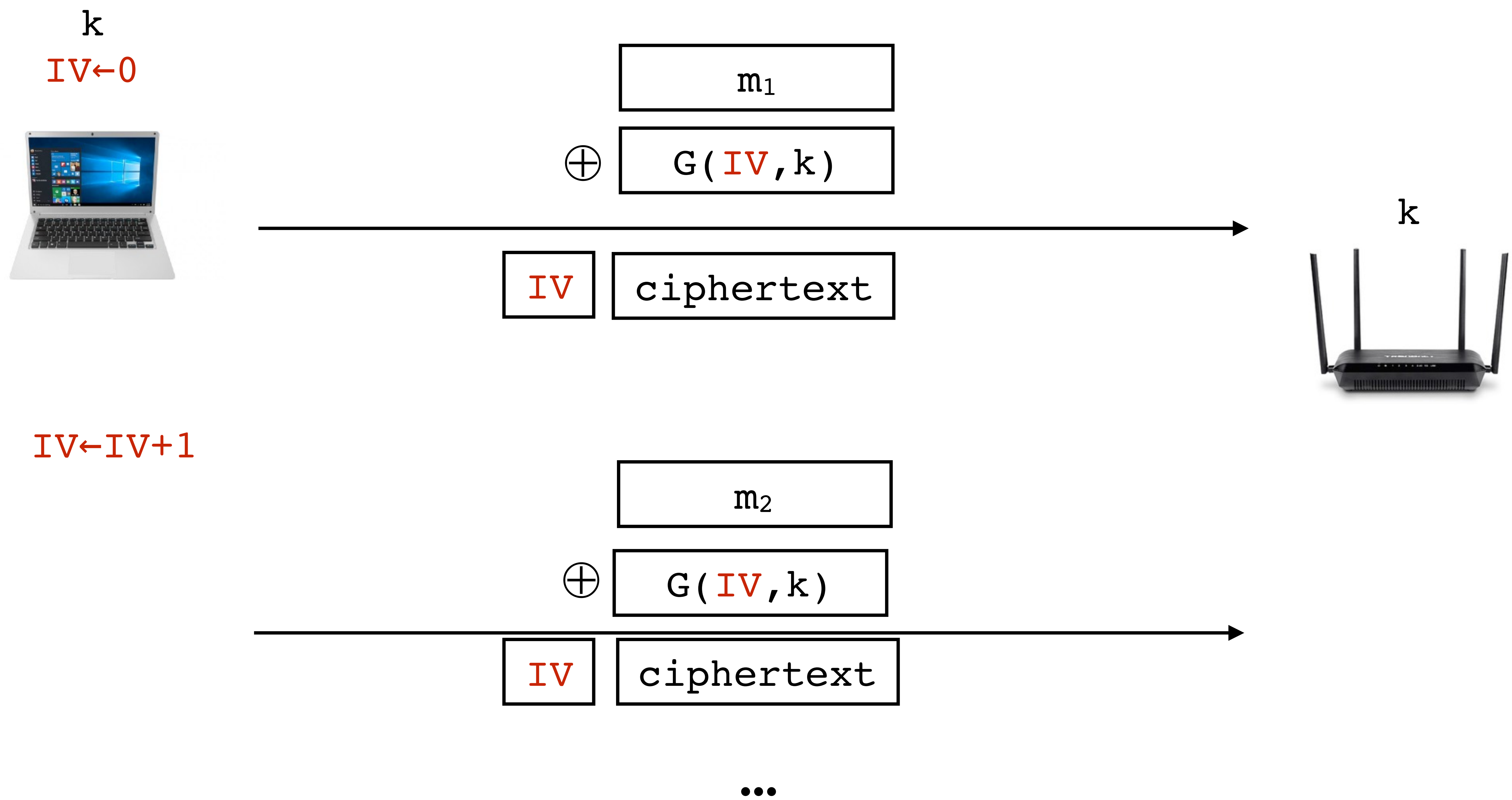
Nonce IV: `1100..11` Key/Seed k: `1100..11`

G(IV,k): `11111010001000111010100101000101100100111100…`

- “nonce” = “number once”.
- Usually denoted IV = “initialization vector”

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

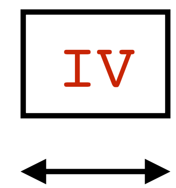


- If nonce repeats, then pad repeats

Example of Pad Re-use: WEP

Warning: Broken

<u>IEEE 802.11b WEP:</u> WiFi security standard '97-'03



IV is 24-bit wide counter

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

<u>Solutions: (WPA2 replacement)</u>

- Larger IV space, or force rekeying more often
- Set IV to combination of packet number, address, etc

Example of Pad Re-use: WEP

Warning: Broken

<u>IEEE 802.11b WEP:</u> WiFi security standard '97-'03

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

- Re
- Of

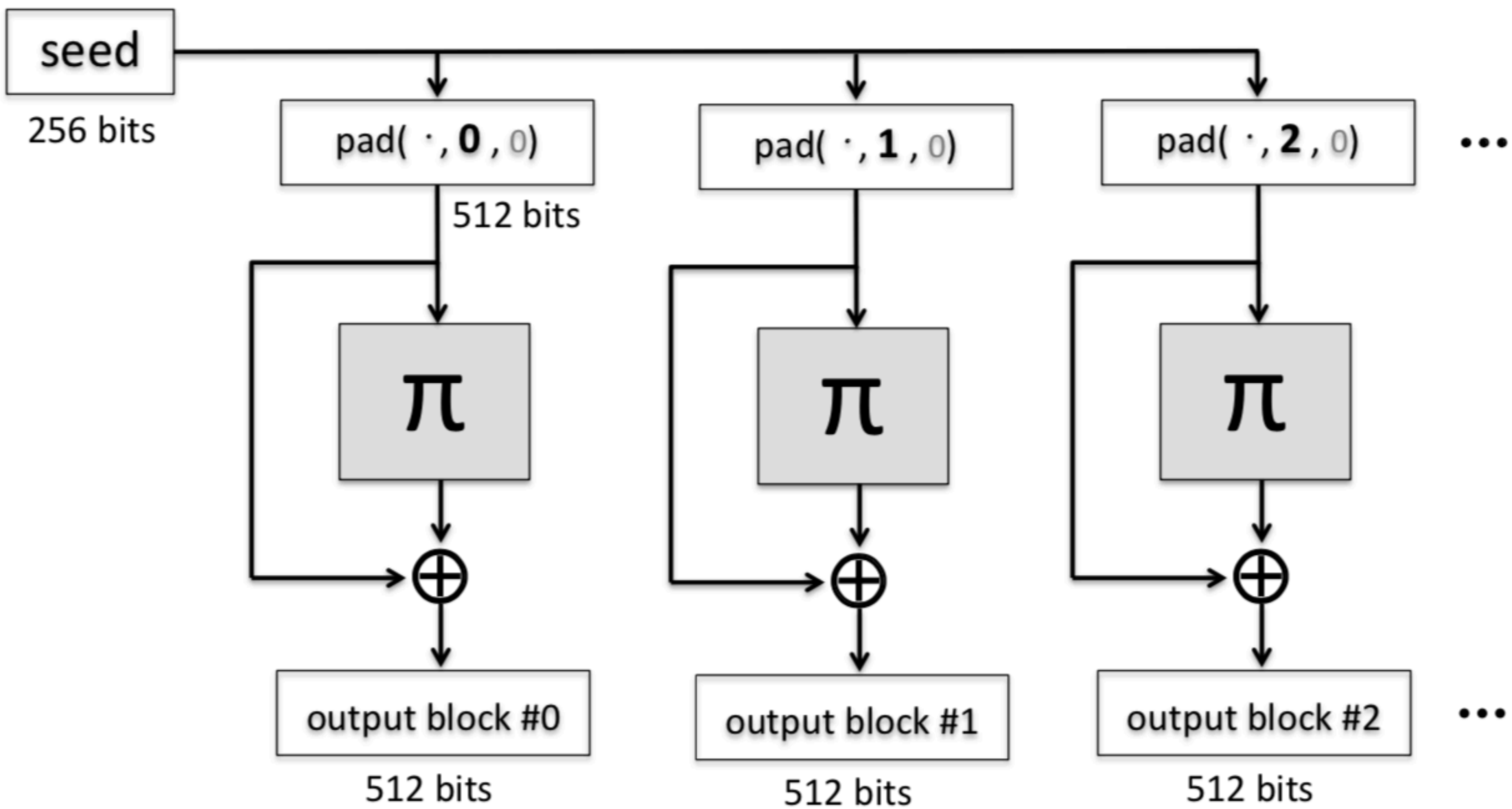
<u>Solutions: (W</u>

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.

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

Example Stream Cipher w/ Nonces: ChaCha20

- Key-length: 256 bits
- Generates stream by applying a fixed permutation to seed and counter
- Uses "feed-forward" to break up permutation structure



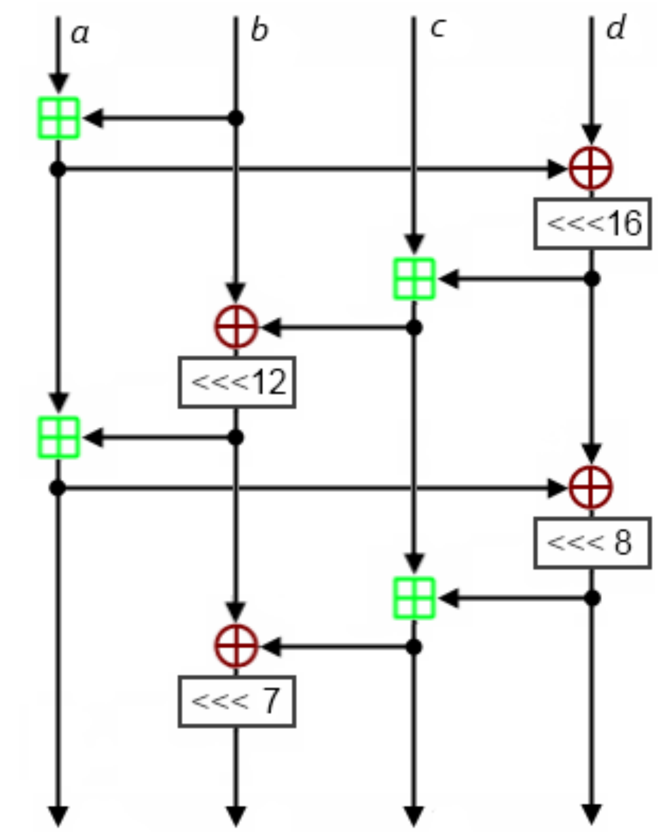
ChaCha20 Block Permutation

```c
#define ROTL(a,b) (((a) << (b)) | ((a) >> (32 - (b))))
#define QR(a, b, c, d) (            \
    a += b,  d ^= a,  d = ROTL(d,16),  \
    c += d,  b ^= c,  b = ROTL(b,12),  \
    a += b,  d ^= a,  d = ROTL(d, 8),  \
    c += d,  b ^= c,  b = ROTL(b, 7))
#define ROUNDS 20

void chacha_block(uint32_t out[16], uint32_t const in[16])
{
    int i;
    uint32_t x[16];

    for (i = 0; i < 16; ++i)
        x[i] = in[i];
    // 10 loops × 2 rounds/loop = 20 rounds
    for (i = 0; i < ROUNDS; i += 2) {
        // Odd round
        QR(x[0], x[4], x[ 8], x[12]); // column 0
        QR(x[1], x[5], x[ 9], x[13]); // column 1
        QR(x[2], x[6], x[10], x[14]); // column 2
        QR(x[3], x[7], x[11], x[15]); // column 3
        // Even round
        QR(x[0], x[5], x[10], x[15]); // diagonal 1 (main diagonal)
        QR(x[1], x[6], x[11], x[12]); // diagonal 2
        QR(x[2], x[7], x[ 8], x[13]); // diagonal 3
        QR(x[3], x[4], x[ 9], x[14]); // diagonal 4
    }
    for (i = 0; i < 16; ++i)
        out[i] = x[i] + in[i];
}
```

QR(a, b, c, d):



???

In Assignment 2: Develop attack when a weak "statistical" stream cipher is used.

Issues with One-Time Pad

1. Reusing a pad is insecure ✓ *Use unique nonces*
2. One-Time Pad is *malleable*
3. One-Time Pad has a long key ✓ *Use stream cipher with short key*

More difficult to address; We will return to this later.

Next Up: Blockciphers

Blockciphers are a ubiquitous crypto tool applied to many different problems.

Informal definition: A <u>blockcipher</u> is essentially a substitution cipher with a very large alphabet and a very compact key. Require that efficient algorithms for forward and backward directions.

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

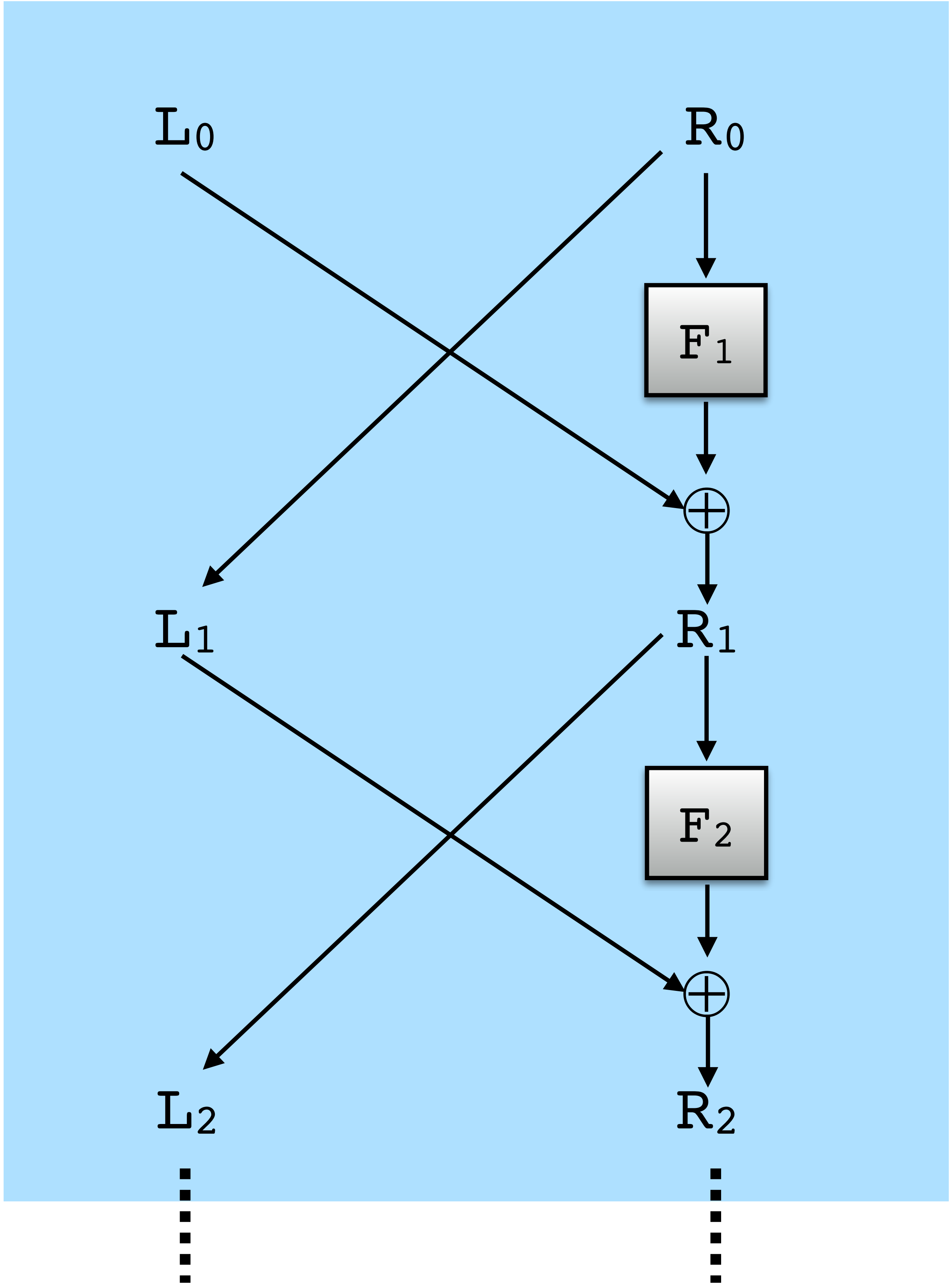
<u>Plan: Build many higher-level protocols from a good blockcipher.</u>

<u>Now: Two example blockciphers, DES and AES.</u>

Data Encryption Standard (DES)

- Originally a designed by IBM
- Parameters adjusted by NSA
- NIST Standard in 1976
    - Block length n = 64
    - Key length k = 56

Parses input block into 32-bit chunks and applies 16 rounds of a “Feistel Network”



DES is Broken

Warning: Broken

Attack	Complexity	Year
Biham&Shamir	2^{47} encrypted blocks	1992
DESCHALL	41 days	1997
EFF Deepcrack	4.5 days	1998
EFF Deepcrack	22 hours	1999

- 3DES (“Triple DES”) is still used by banks
- 3DES encrypts three times (so key length is 118)
- 3DES is not known to be broken but should be avoided

GET CRACKING

NOTE: There are currently extremely high wait times.
We're in the process of adding capacity to speed things up.

QUEUE WAIT TIME:
Standard 46.2 Days, ASAP 1.0 Days

These are the types of DES cracking jobs that we support:

Windows LM/NTLMv1 Authentication
PPTP VPNs WPA-Enterprise
des_crypt() Hashes
DES Kerberos5 Known Plaintext DES

SUBMIT A JOB!

Token:

Priority: Enter Token For Pricing

PAY WITH CARD

WARNING: Charges will show up on your credit card statement as from "crack.sh" and processed through Stripe. We've experienced a high number of our charges being reported as fraudulent, so we'll be blacklisting any accounts that contest charges for jobs submitted. If you wish to cancel a job or have any issues, please email david@toorcon.org and we'll be happy to cancel and refund any charges.

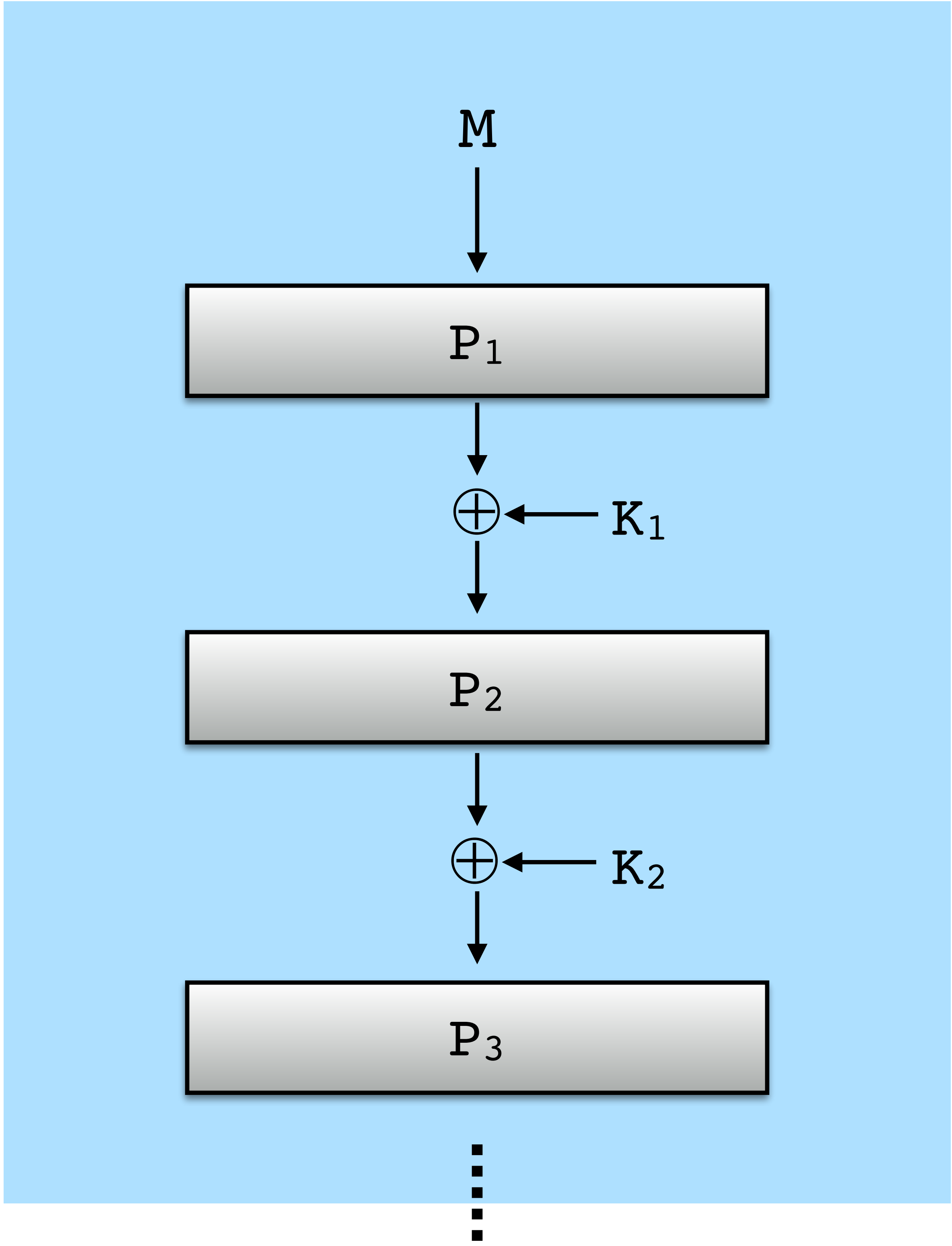
Advanced Encryption Standard (AES)

- NIST ran competition to replace DES starting in 1997
- Several submissions, *Rijndael* chosen and standardized
- AES is now the gold standard blockcipher
- Very fast; Intel chips even have AES instructions

Advanced Encryption Standard (AES)

- Due to Rijmen and Daemen
  - Block length n = 128
  - Key length k = 128,192,256

- Different structure from DES.
- 10 rounds of "substitution-permutation network"



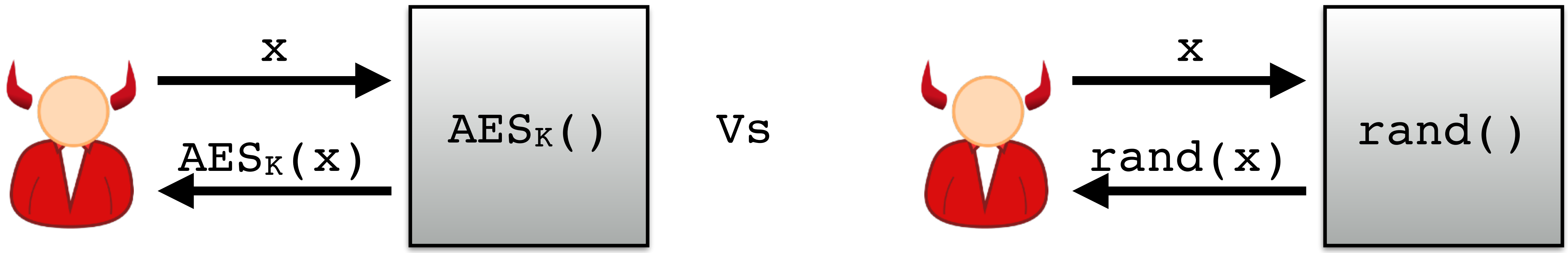
AES is not (know to be) broken

Attack	Complexity	Year
Bogdanov et al.	$\approx 2^{126.1}$	2011

- Compare to trying all keys: $2^{126.1} \approx 2^{128}/4$
- Always prefer AES for a blockcipher if setting can support it (i.e. everything except low-power hardware)

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)=9918e60f2a20b1b8167464 6dceebdb51

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.

  This next step is where everything flies off the rails in implementations…

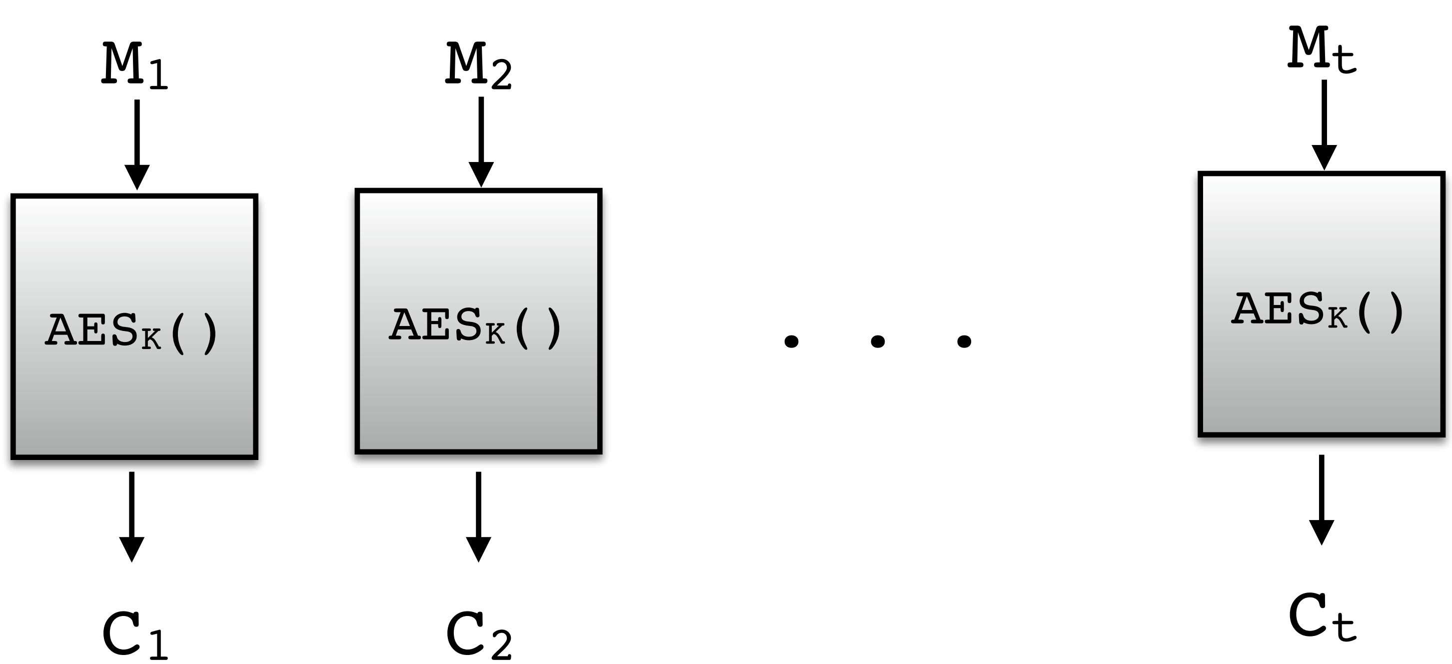
Encrypting large files: ECB

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 $C_1, \ldots, C_t$



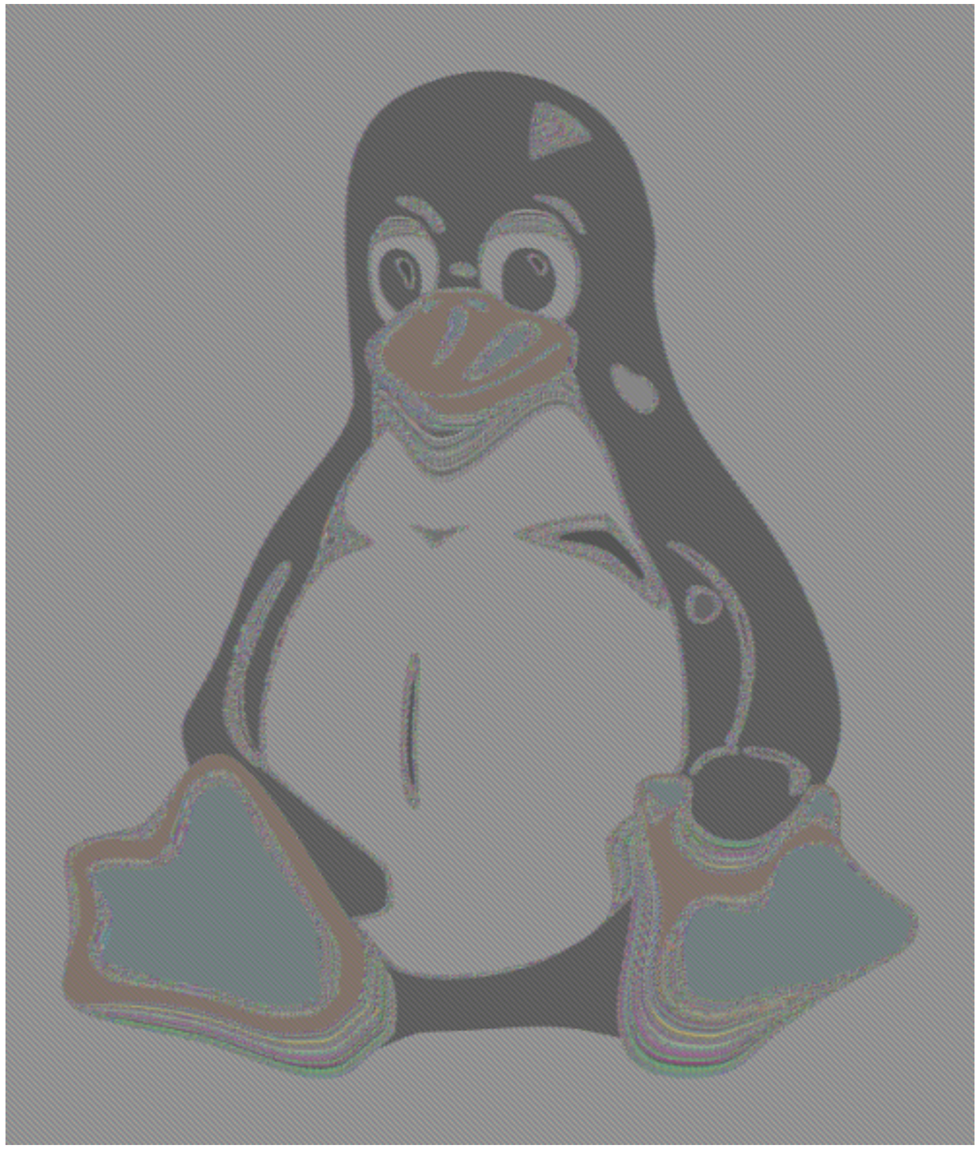
The ECB Penguin

Warning: Broken

- 16 byte chunks are consecutive pixels

Plaintext

ECB Ciphertext



- It gets even worse…

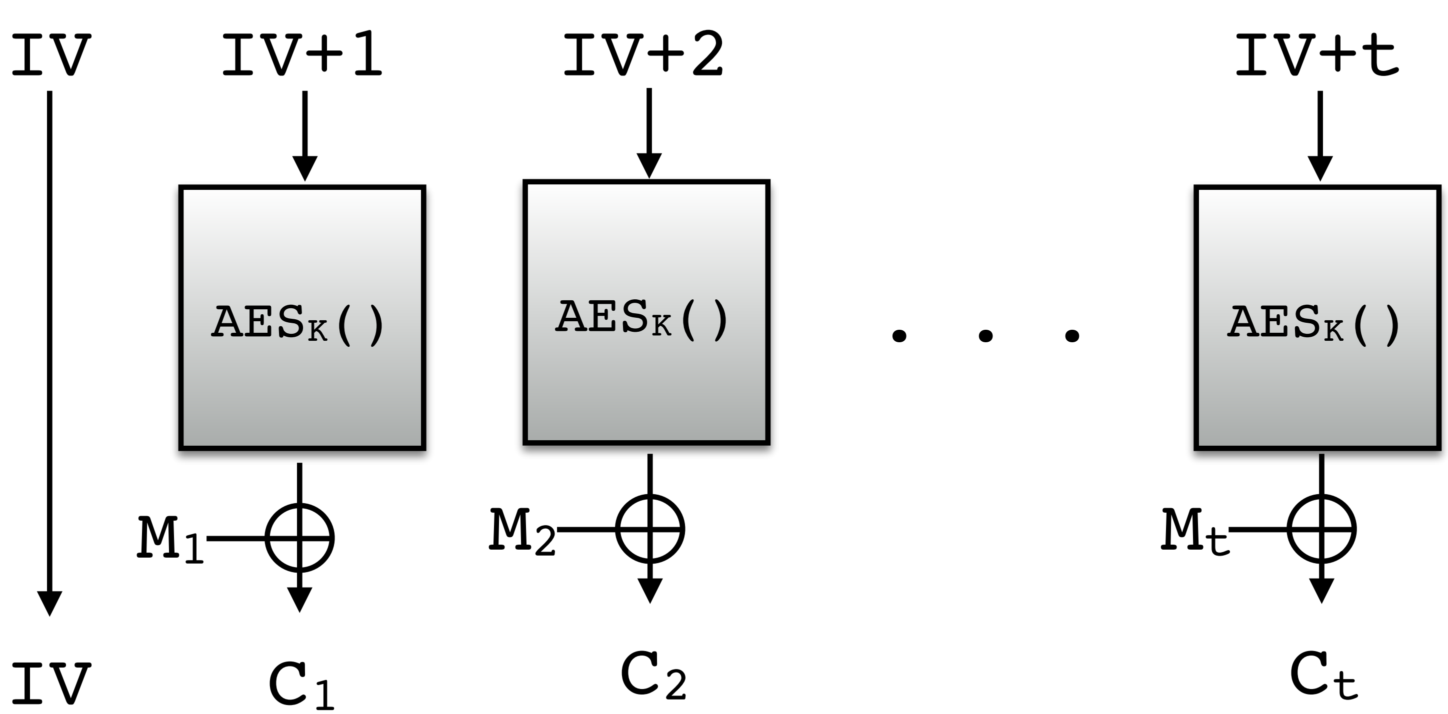
Encrypting large files, Attempt #2: CTR

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

$\text{AES-CTR}_k(\text{IV}, M)$

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

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

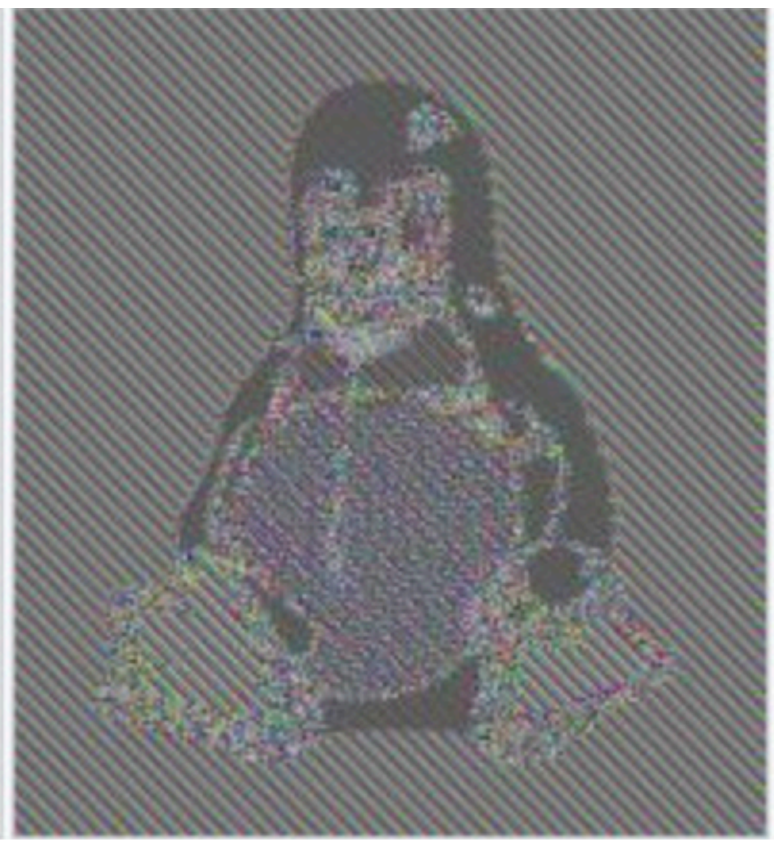


When combined with authentication, CTR is a good cipher.

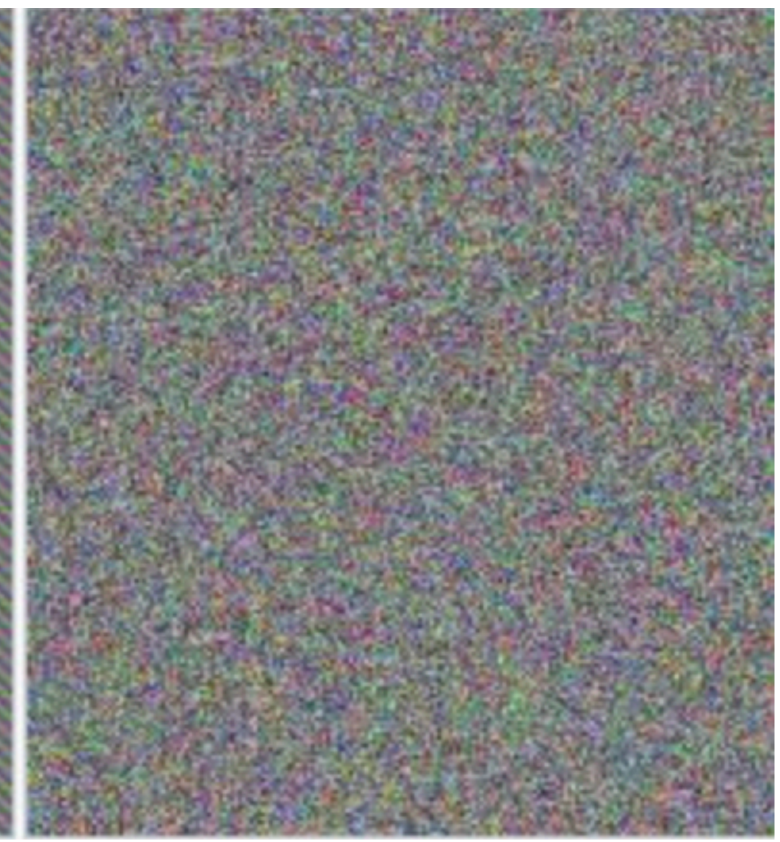
Penguin Sanity Check

Plaintext

ECB Ciphertext



CTR Ciphertext



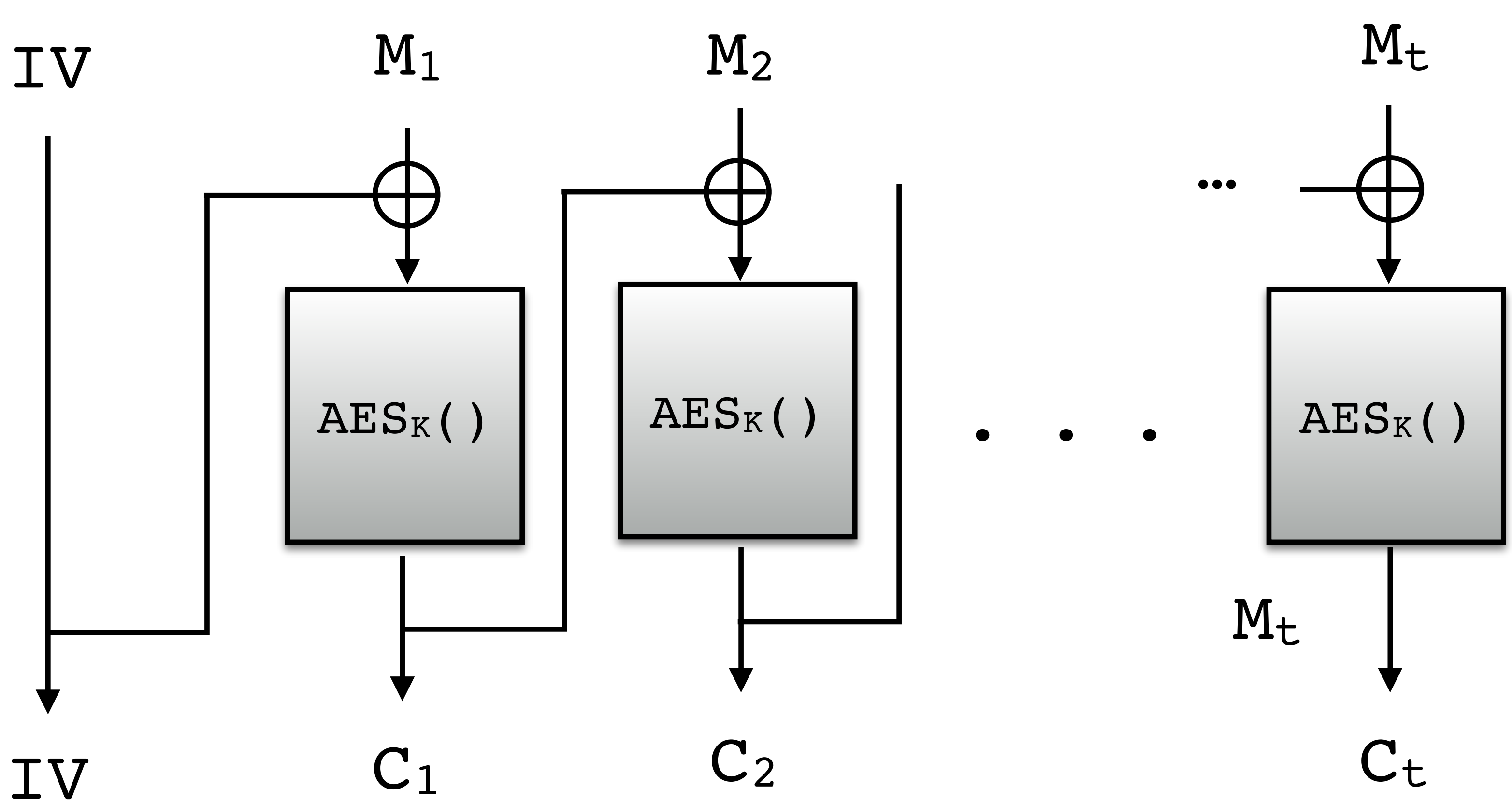
Looks random

Encrypting large files, Attempt #3: CBC

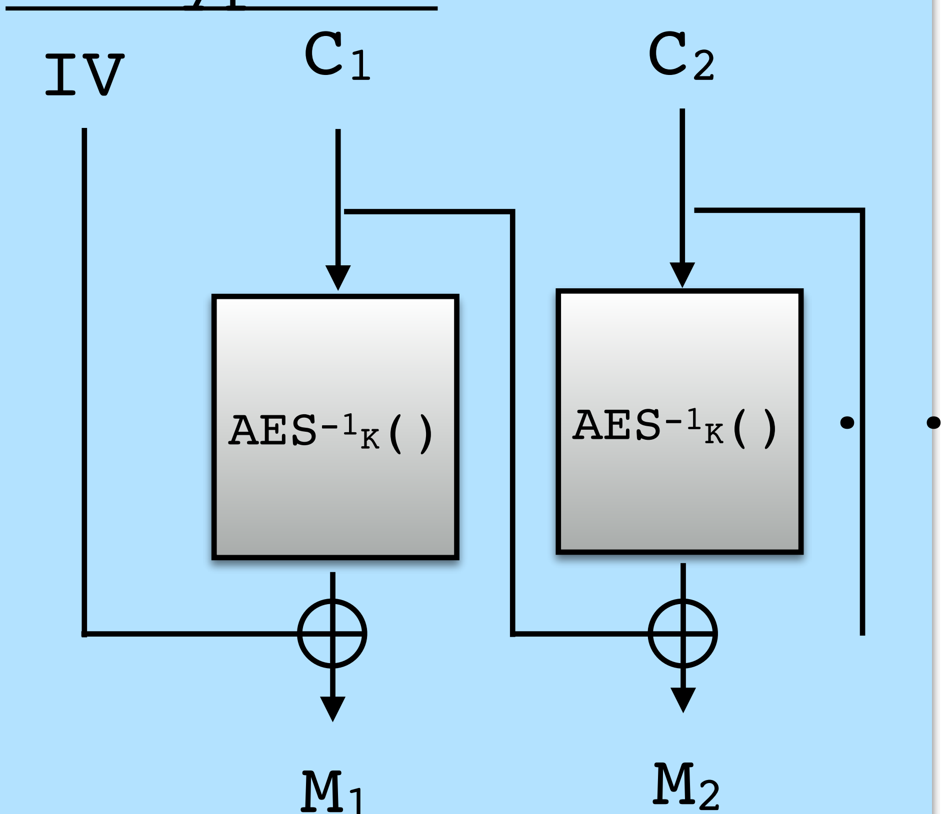
- CBC = "Cipher Block Chaining"
- Nonce-based, but not a stream cipher
- Historical option (sometimes used without nonce)

$\texttt{AES-CBC}_k\texttt{(IV,M)}$

- Parse $\texttt{M}$ 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 $i=1\ldots t$:
  - $C_i \leftarrow AES_k(M_i \oplus C_{i-1})$
- Return $C_0, C_1, \ldots, C_t$



Decryption

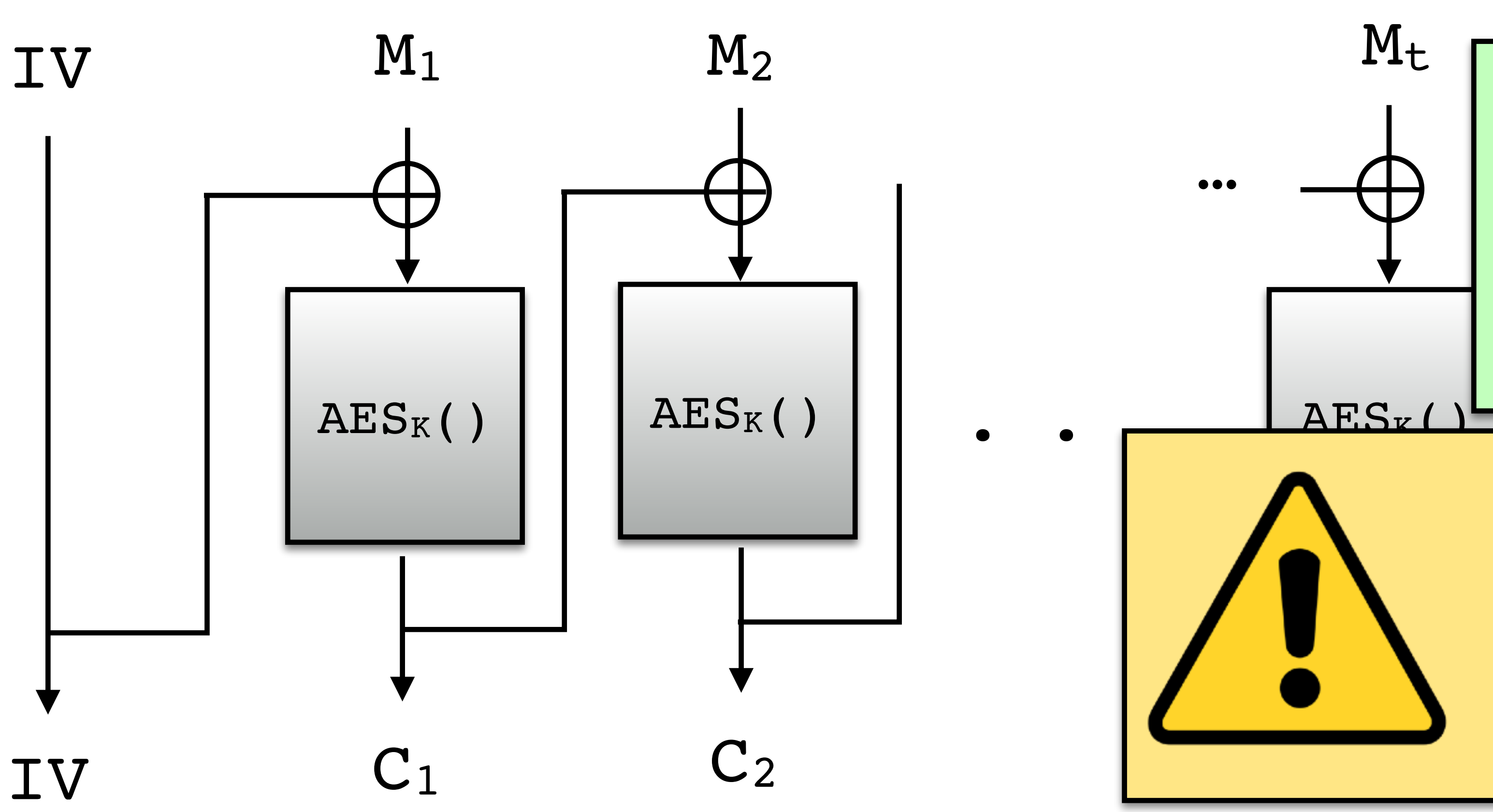


Encrypting large files, Attempt #3: CBC

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

$\underline{\texttt{AES-CBC}_K(\texttt{IV,M})}$

- 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 $i=1\ldots t$:
  - $C_i \leftarrow AES_K(M_i \oplus C_{i-1})$
- Return $C_0, C_1, \ldots, C_t$



<u>When combined with authentication</u>, CBC is a good cipher.

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

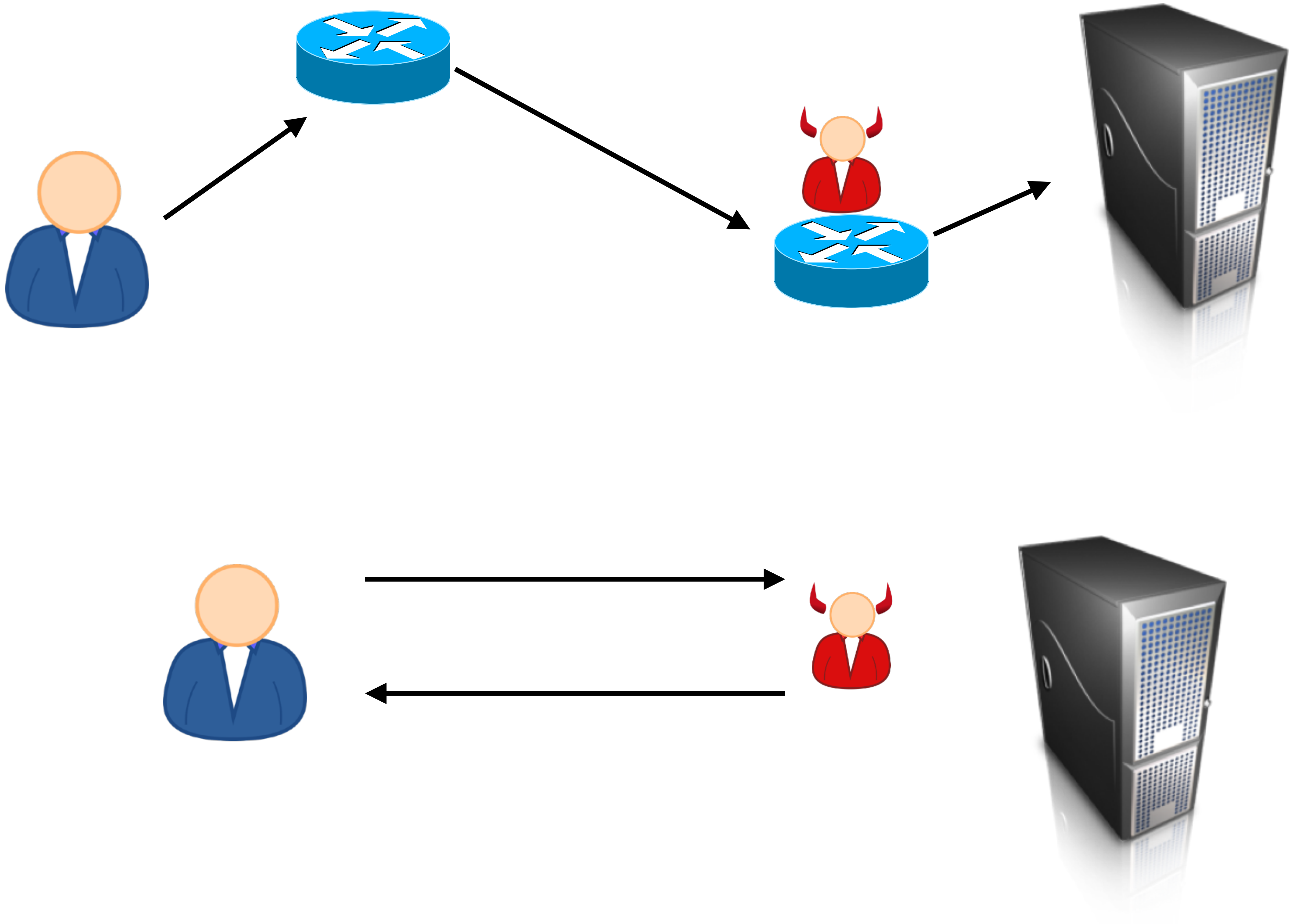
Blockcipher Encryption Summary

- 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.

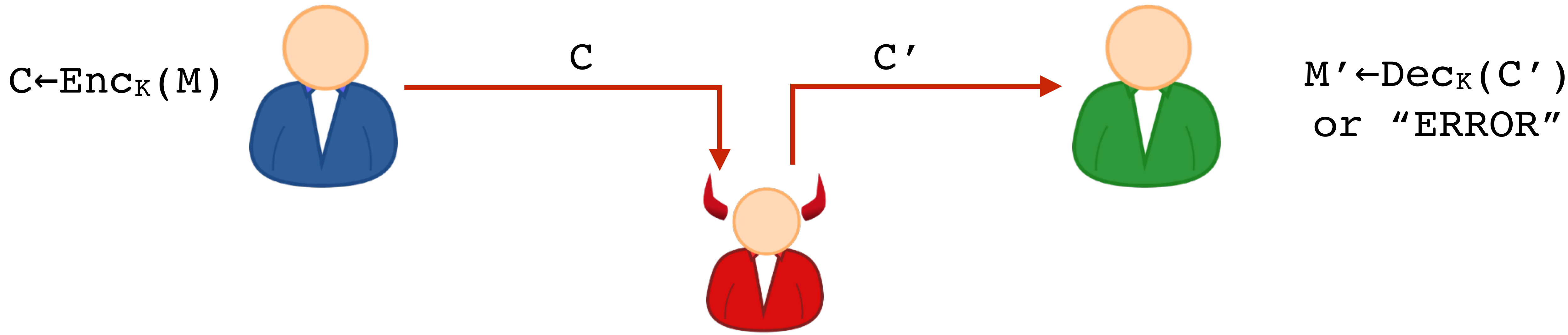
Next Up: Integrity and Authentication

- Authenticity: Guarantee that adversary cannot change or insert ciphertexts
- Achieved with MAC = “Message Authentication Code”

Integrity: Preventing message modification



Encryption Integrity: An abstract setting



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

AES-CTR does not satisfy integrity

M = please pay ben 20 bucks

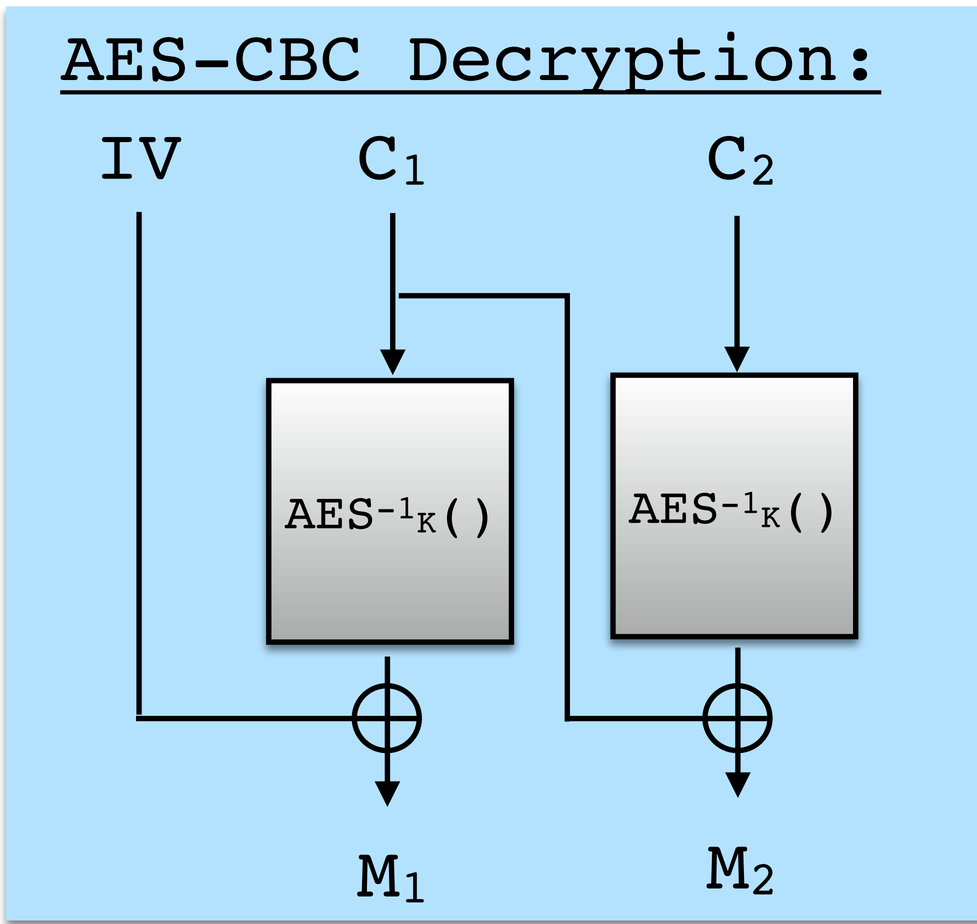
C = b0595fafd05df4a7d8a04ced2d1ec800d2daed851ff509b3e446a782871c2d

C’= b0595fafd05df4a7d8a04ced2d1ec800d2daed851ff509b3e546a782871c2d

M’ = please pay ben 21 bucks

Inherent to stream-cipher approach to encryption.

AES-CBC does not satisfy integrity



IV, C_1, C_2

$IV, C_1 \oplus X, C_2$

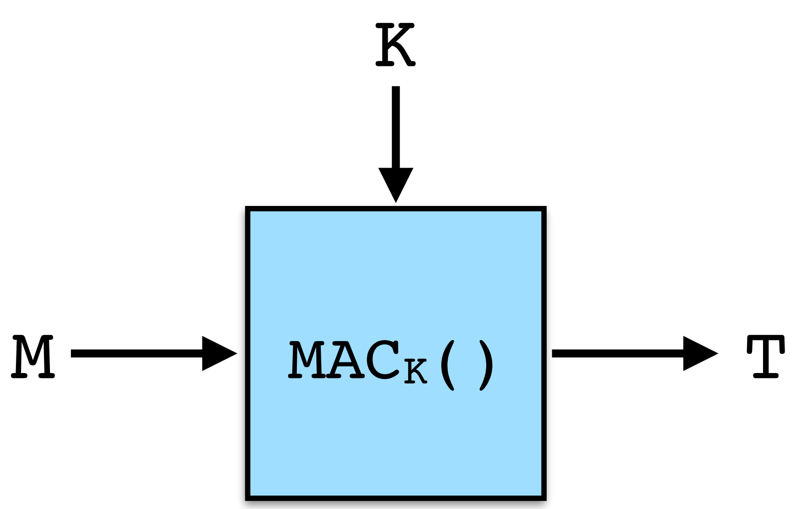
Decrypts to:

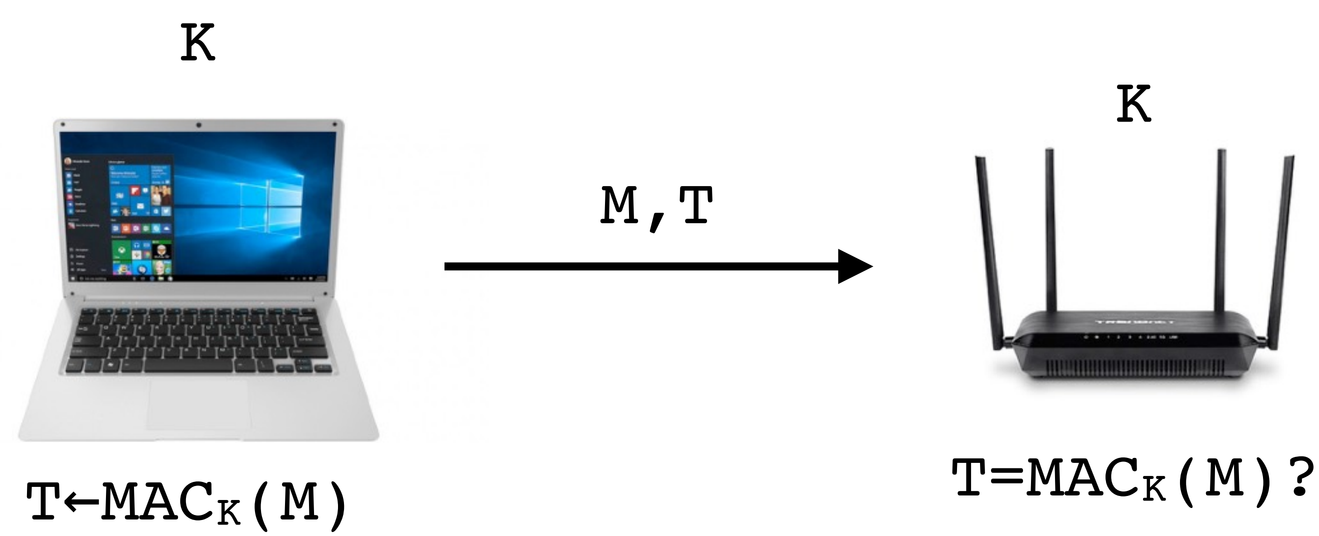
$R, M_2 \oplus X$

Where R is some unpredictable block.

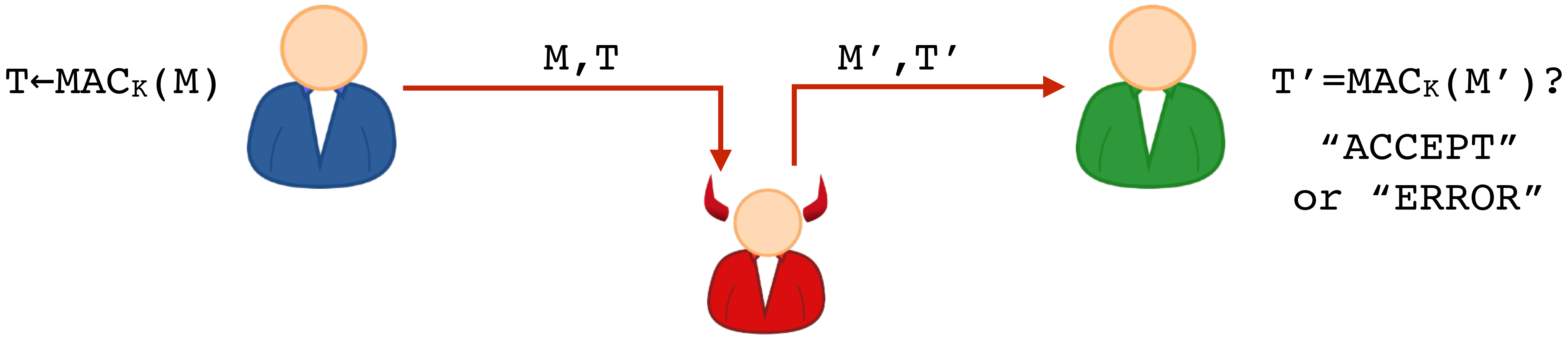
Message Authentication Code

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





MAC Security Goal: Unforgeability



MAC satisfies **unforgeability** if it is unfeasible for Adversary to fool Bob into accepting **M′** not previously sent by Alice.

MAC Security Goal: Unforgeability

Note: No encryption on this slide.

`M = please pay ben 20 bucks`

`T = 827851dc9cf0f92ddcdc552572ffd8bc`

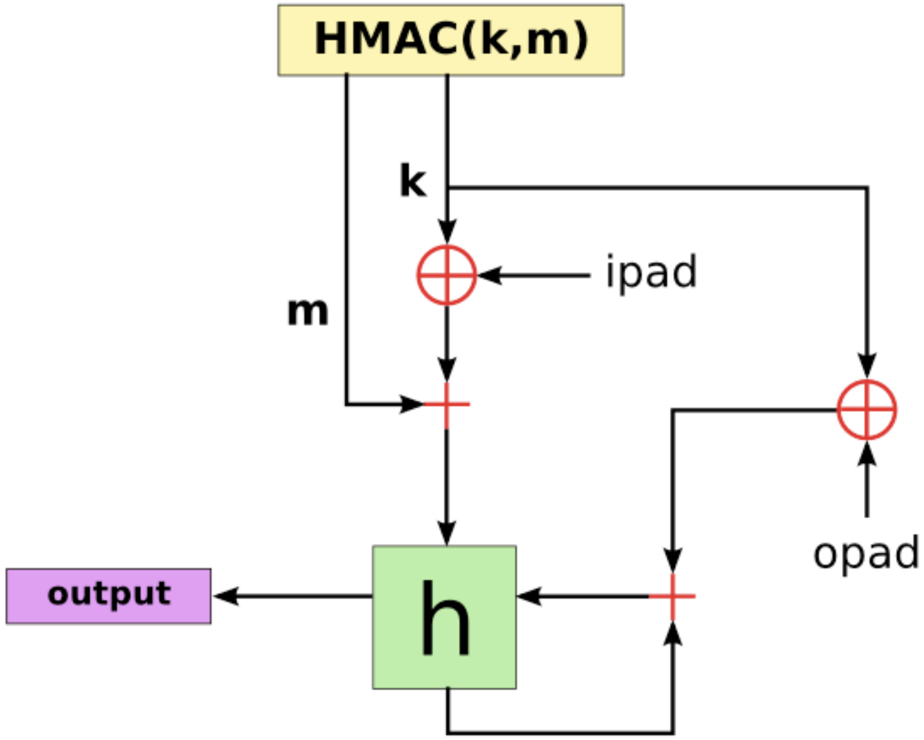
`M’= please pay ben 21 bucks`

`T’= baeaf48a891de588ce588f8535ef58b6`

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

MACs In Practice: Pretty much always use HMAC

- Don't worry about how it works.
- More precisely: Use HMAC-SHA2. More on hashes and MACs later.



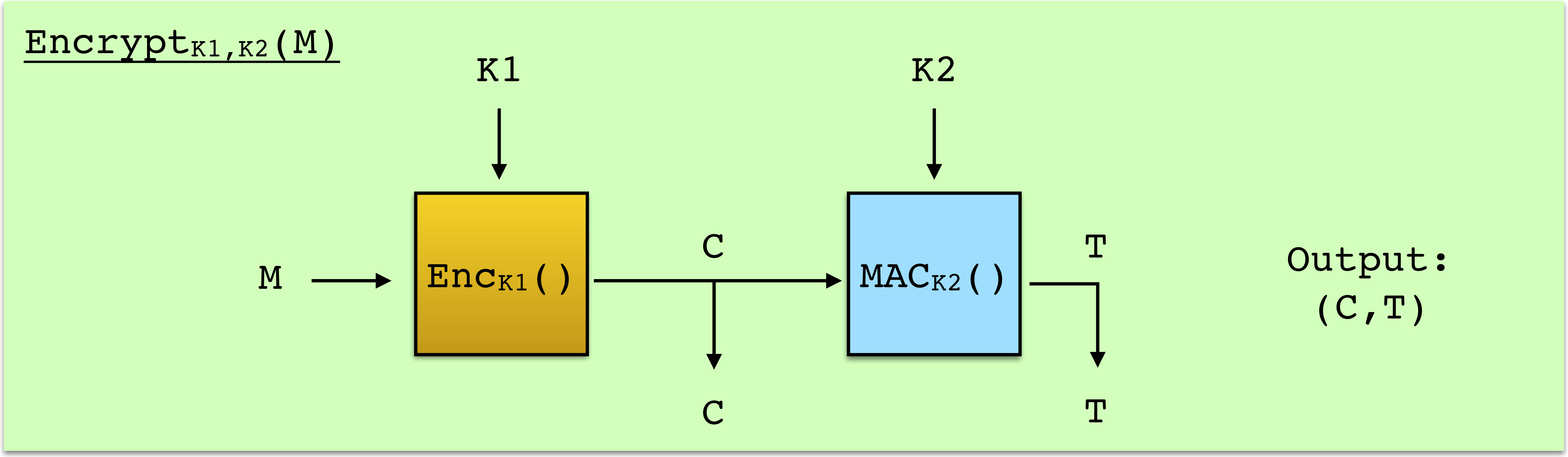
- Other options: Poly1305-AES or CBC-MAC (the latter is tricky)

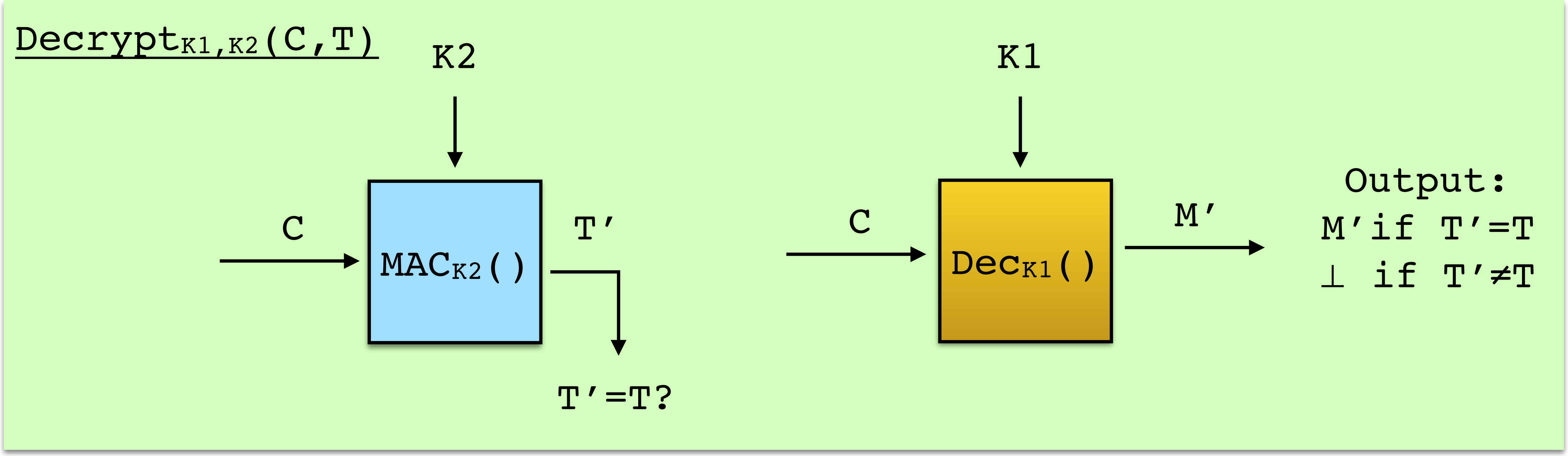
Authenticated Encryption

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

- Built using a good cipher and a MAC.
  - Ex: AES-CTR with HMAC-SHA2
- Best solution: Use ready-made Authenticated Encryption
  - Ex: AES-GCM is the standard

Building Authenticated Encryption

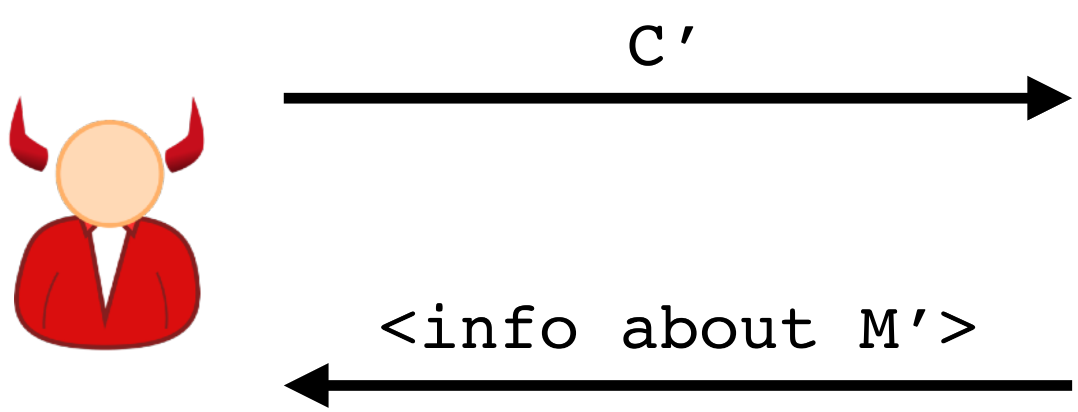


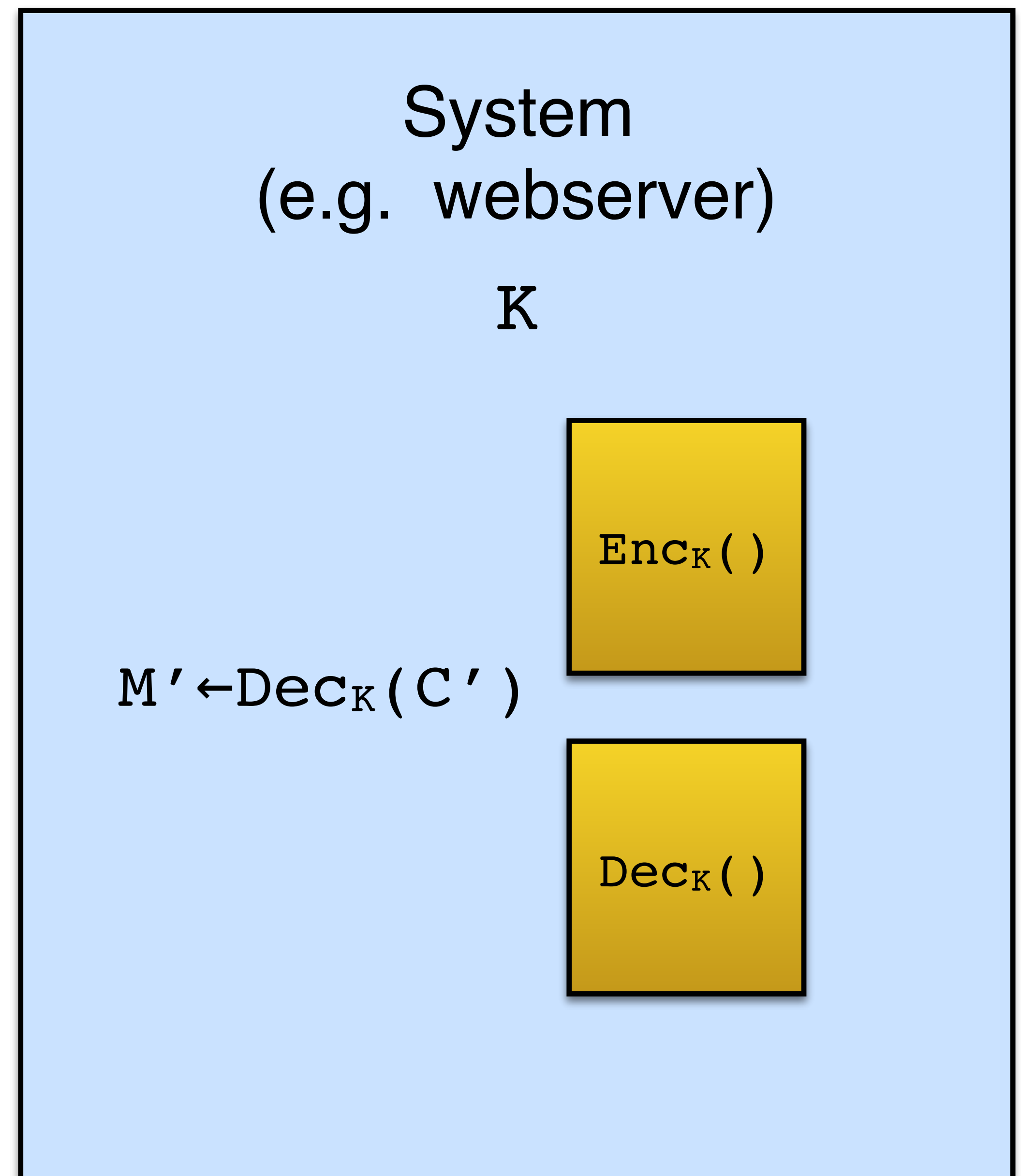


- Summary: MAC the ciphertext, not the message

Chosen-Ciphertext Attacks (CCA) against Encryption

- Integrity + Confidentiality = security against CCAs

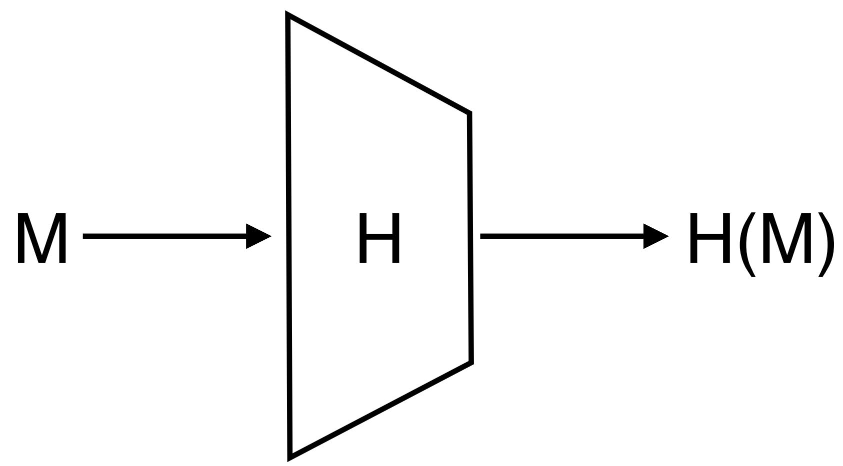




- Adversary provides ciphertext inputs to system
- Obtains info about decryptions of its ciphertexts

Next Up: Hash Functions

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



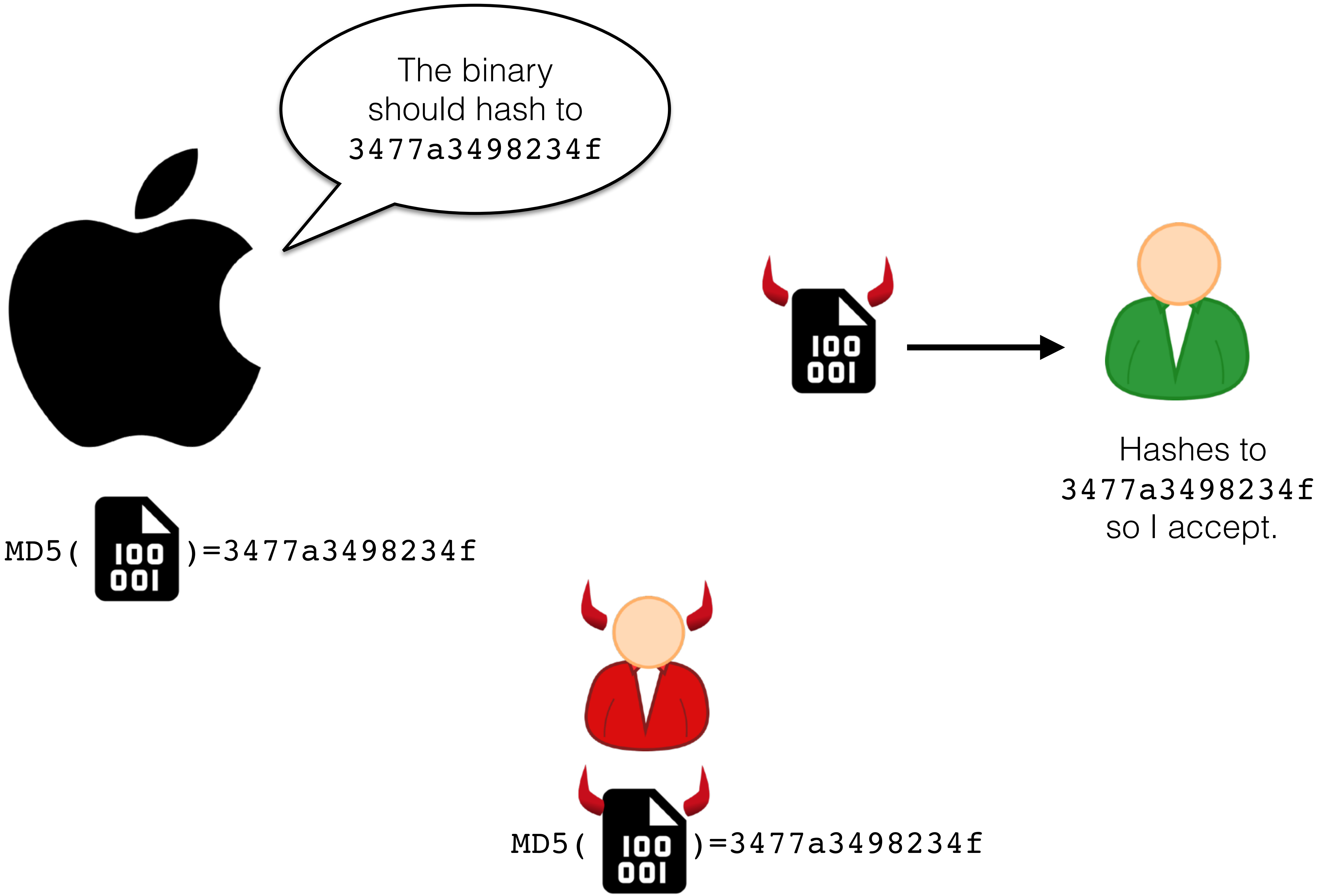
	Output length
MD5:	m = 128 bits
SHA-1:	m = 160 bits
SHA-256:	m = 256 bits
SHA-512:	m = 512 bits
SHA-3:	m >= 224 bits

Some security goals:

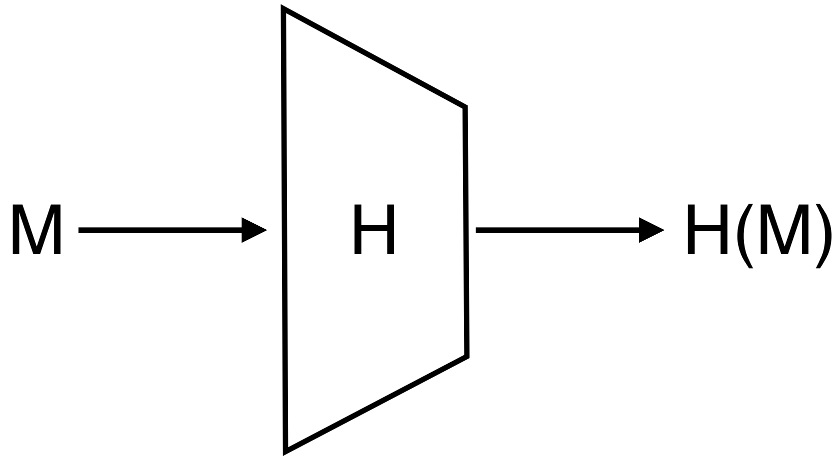
- collision resistance: can't find M != M' such that H(M) = H(M')
- preimage resistance: given H(M), can't find M
- second-preimage resistance: given H(M), can't find M' s.t. H(M') = H(M)

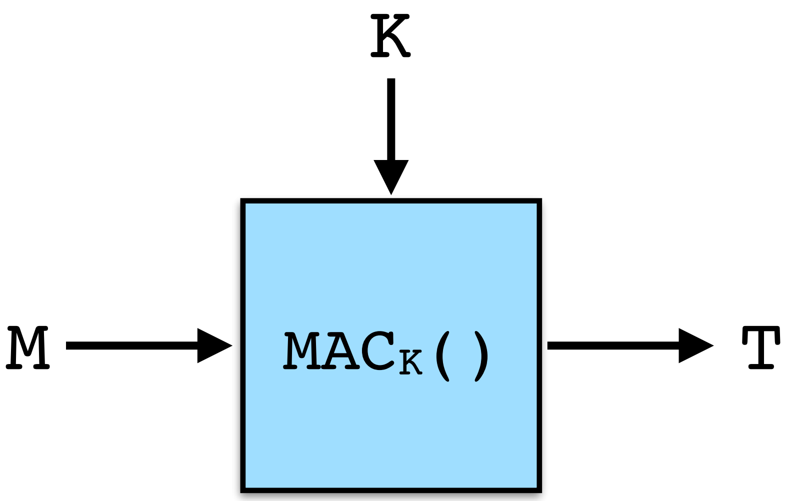
Note: Very different from hashes used in data structures!

Why are collisions bad?



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

- Can always find a collision in $2^{m/2}$ time ($\ll 2^m$ time). “Birthday Attack”
- 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

- Broken with clever techniques
- Compare to DES (broken b/c key too short)

In Assignment 2: Install and use actual attack code to see how MD5 can be abused.

MACs from Hash Functions

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

In Assignment 2: Break this construction!

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

- Totally insecure if H = MD5, SHA1, SHA-256, SHA-512
- Is secure with SHA-3

Construction: $MAC(K, M) = H(M \| K)$ **Just don't**

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

Later: Hash functions and certificates

The End