Blockciphers Modes, Authentication

CMSC 23200/33250, Autumn 2018, Lecture 4

David Cash

University of Chicago

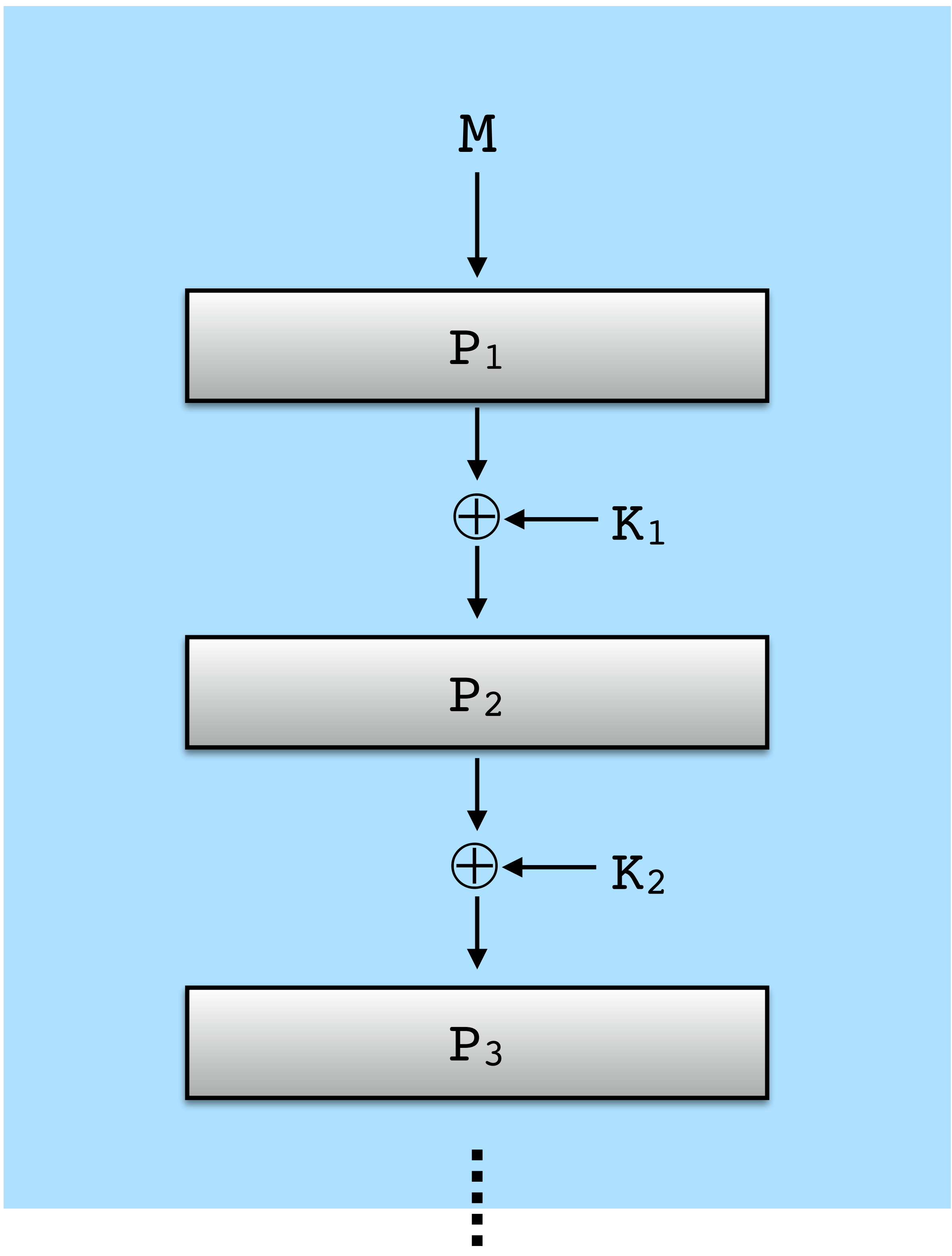
Plan

1. Blockciphers recall
2. Blockcipher modes (encrypting large messages)
3. Authentication: MACs
4. Authenticated Encryption
5. Padding Oracle Attacks

Advanced Encryption Standard (AES)

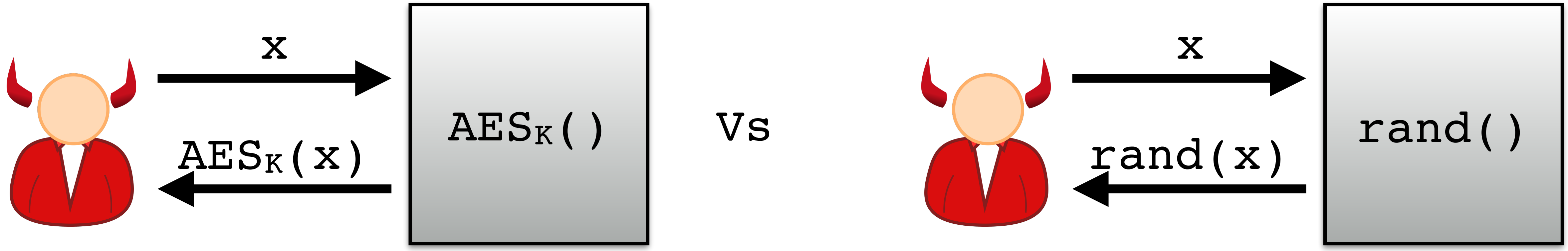
- Due to Rijmen and Daemen
  - Block length n = 128
  - Key length k = 128,192,256

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



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: 9500924ad9d1b7a28391887d95fcfbd<u>6</u>

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…

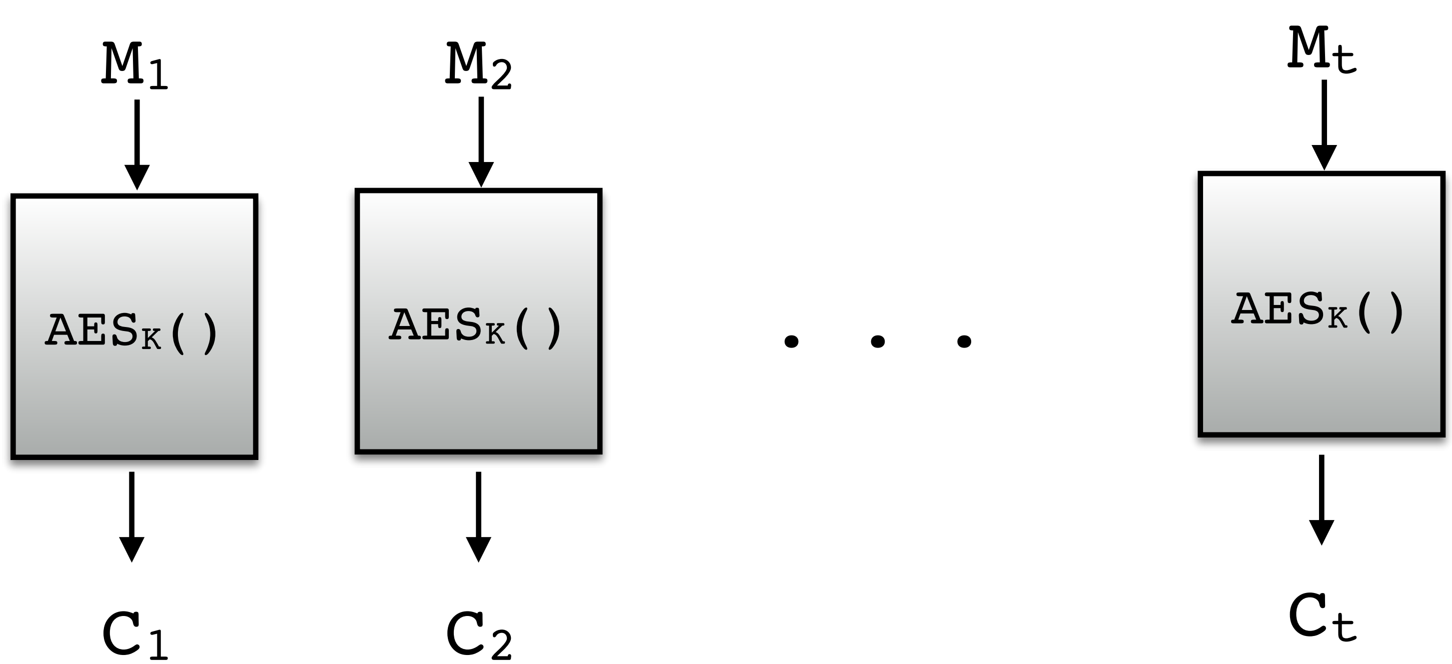
Encrypting large files: ECB

Warning: Broken

- ECB = "Electronic Code Book"

<u>$\texttt{AES-ECB}_k\texttt{(M)}$</u>

- 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 $i=1 \ldots t$:
  - $C_i \leftarrow AES_k(M_i)$
- Return $C_1, \ldots, C_t$



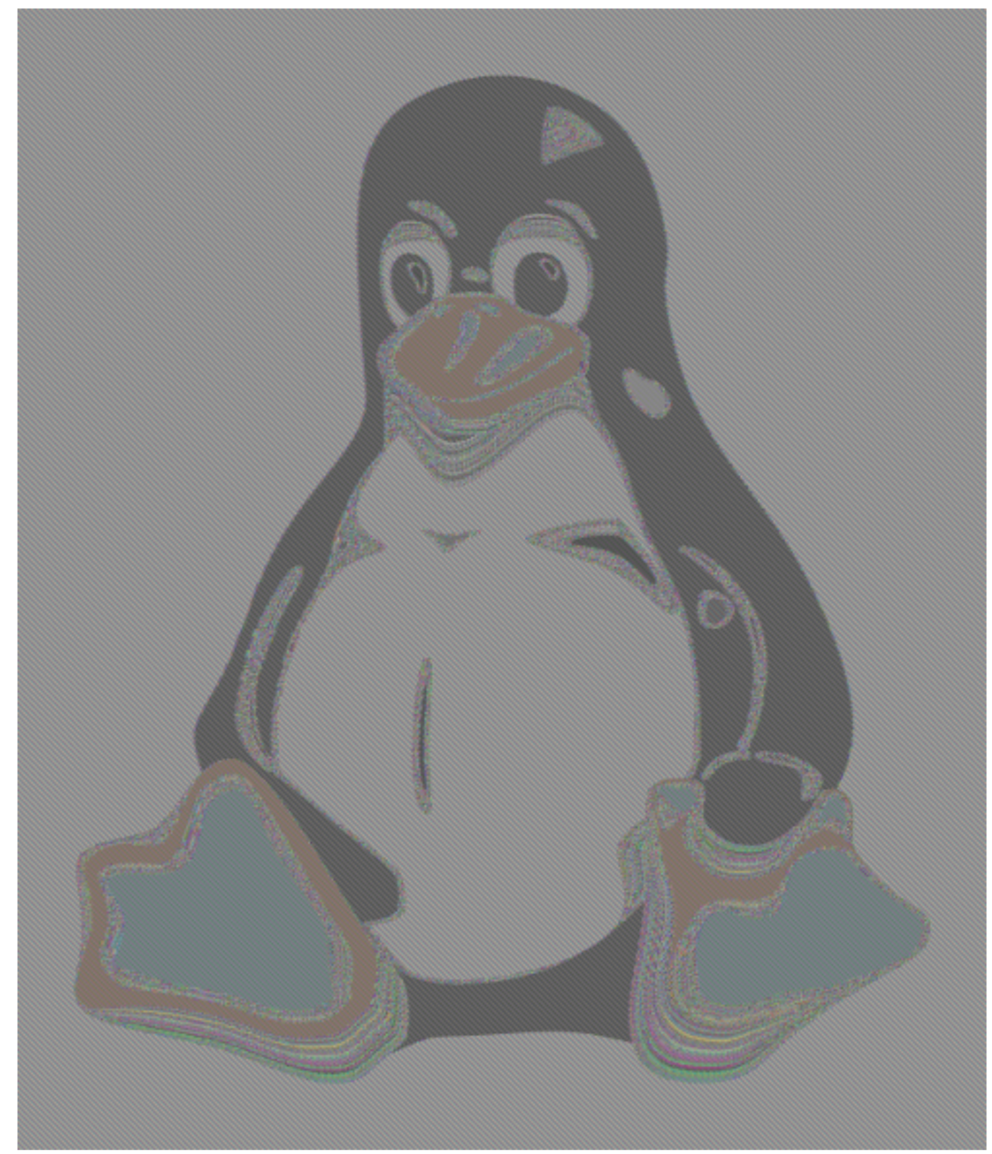
The ECB Penguin

Warning: Broken

- 16 byte chunks are consecutive pixels

Plaintext

ECB Ciphertext

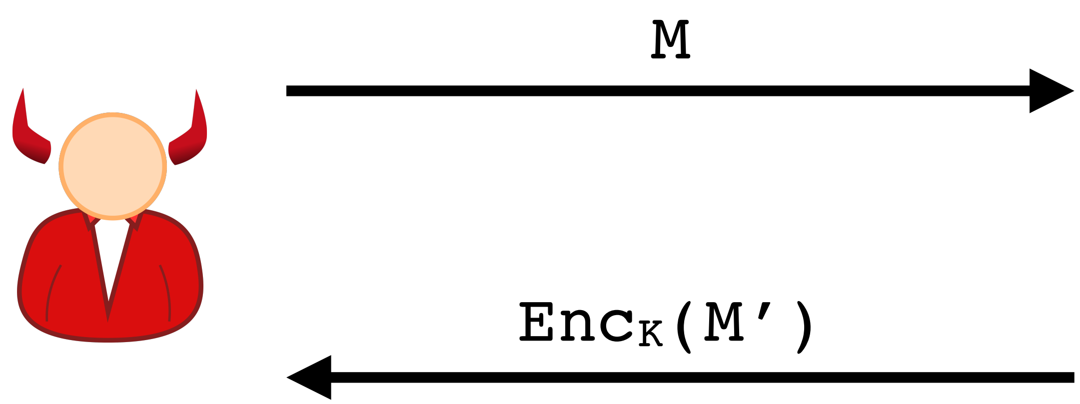


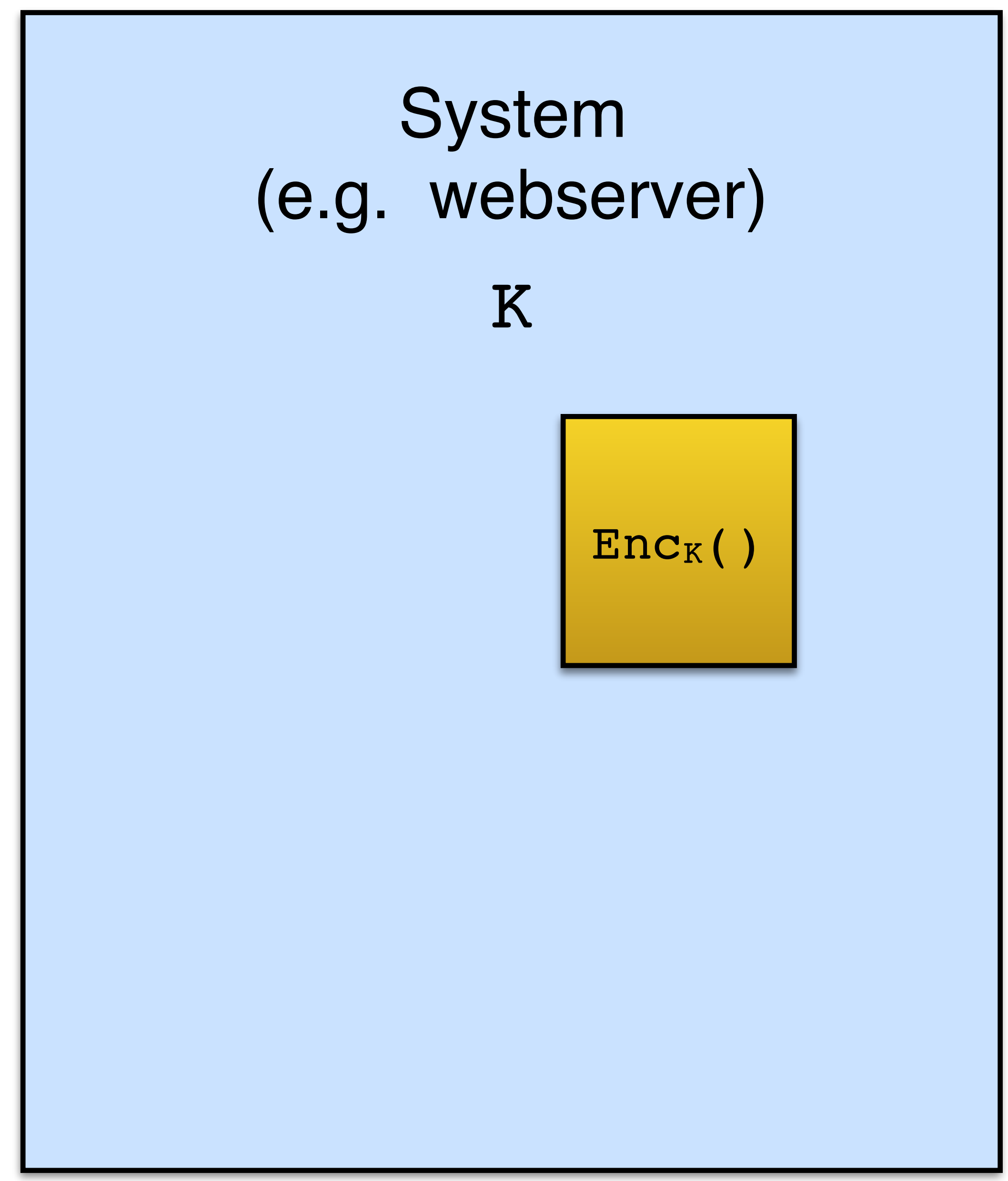
- It gets even worse…

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 decrypt *any* ciphertext.

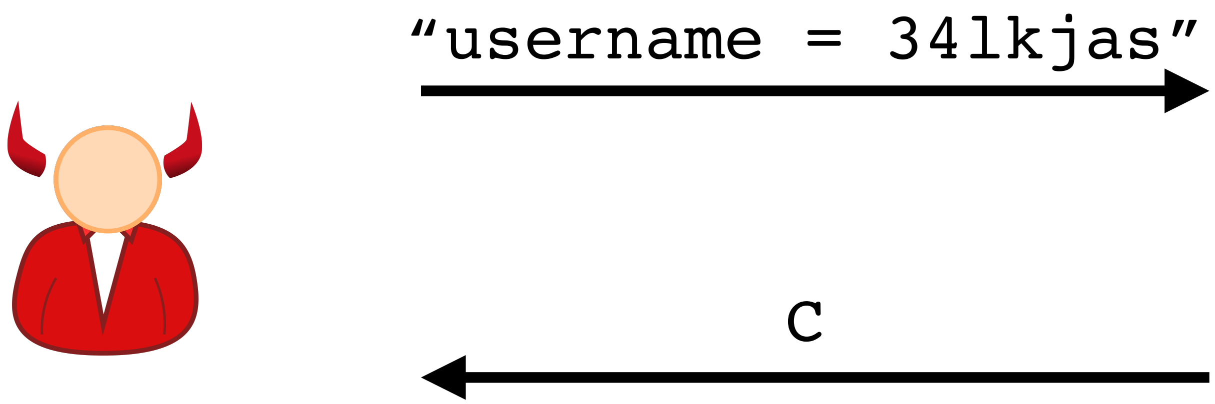
Chosen-Plaintext Attacks (CPA) against Encryption

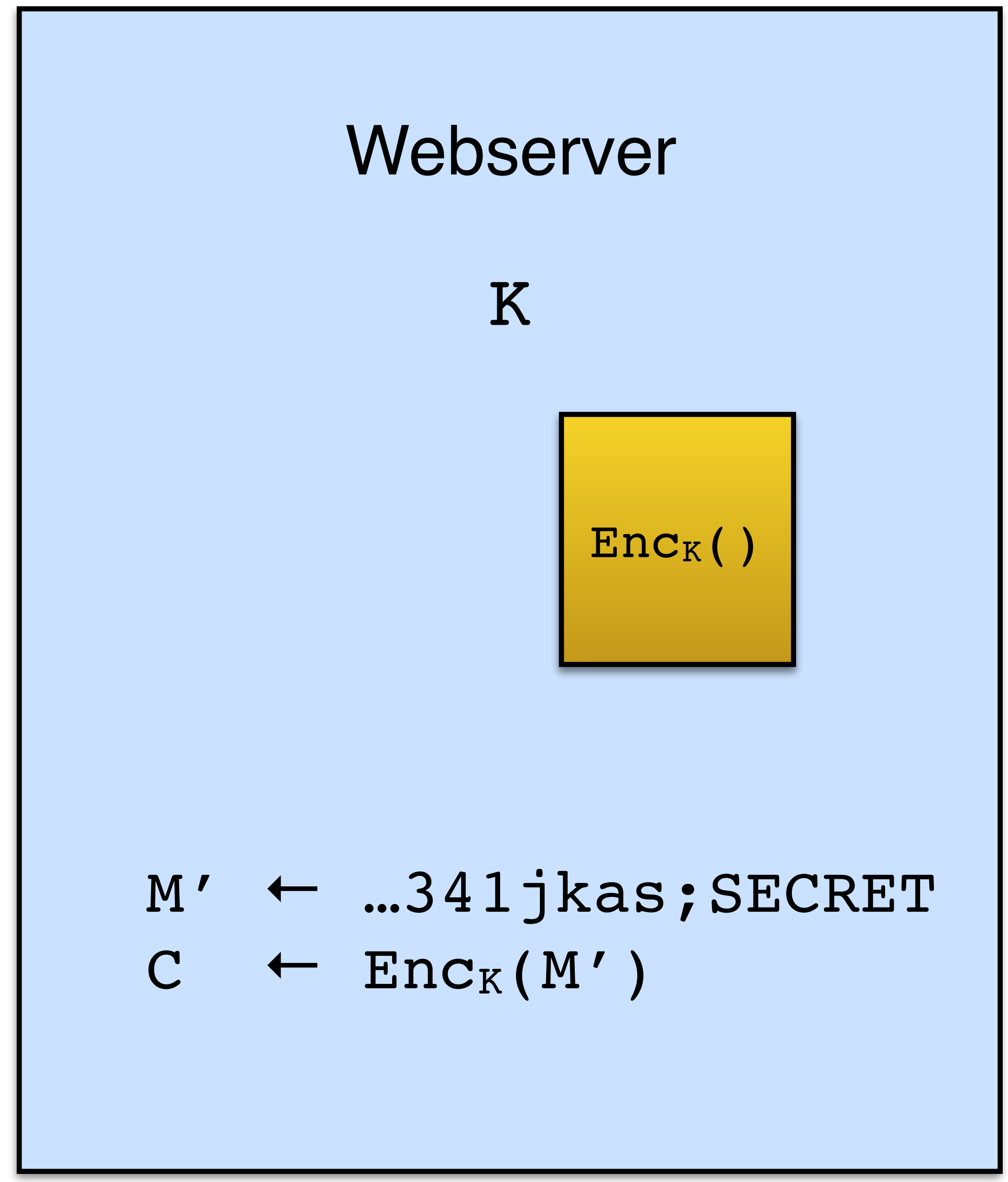




- Adversary provides inputs to system
- Obtains encryption of message that depends on its inputs
- Sometimes **M=M'**

CPA Example: Encrypted Cookies





- More later in web security module

Assignment 1 preview: ECB is totally insecure in this setting. You will attack it and recover SECRET.

Encrypting large files, Attempt #2: CTR

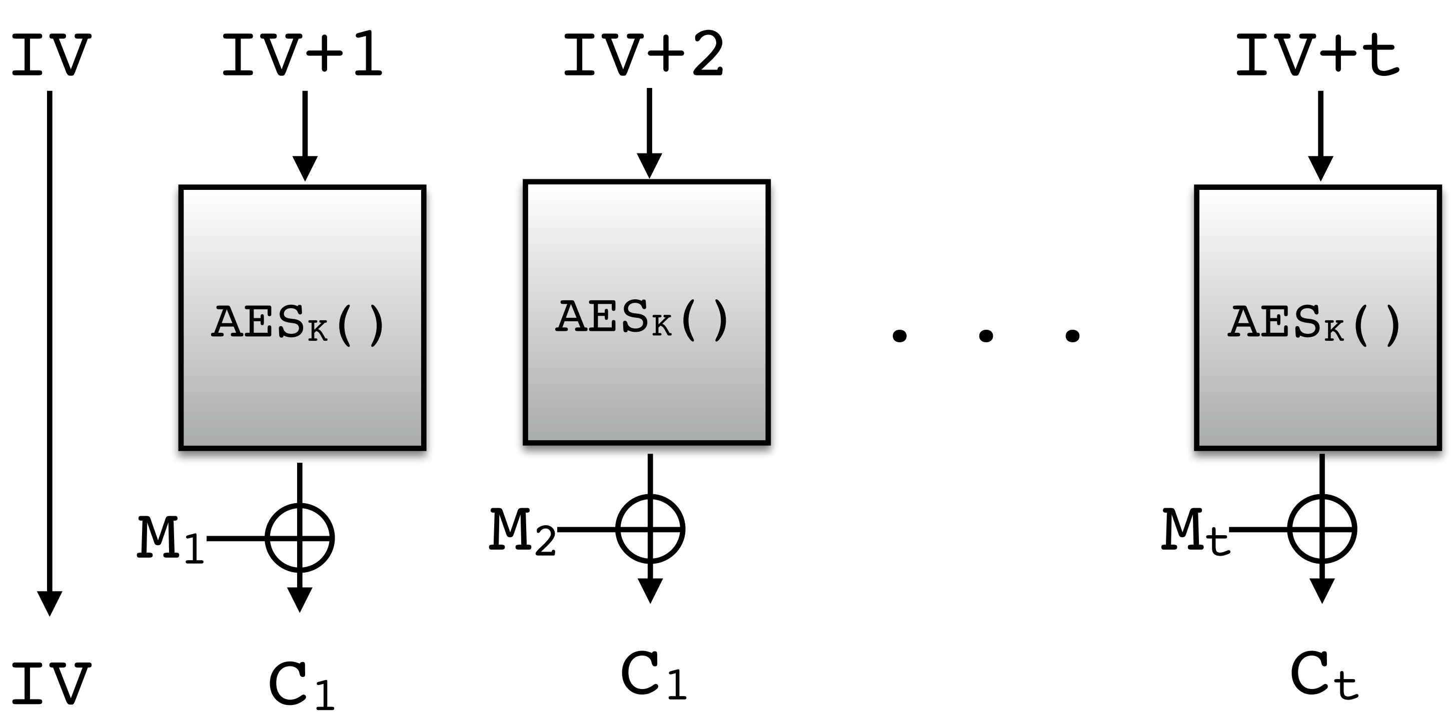
- CTR = "Counter Mode"
- Idea: Build a nonce-based stream cipher from AES

<u>$\texttt{AES-CTR}_k\texttt{(IV,M)}$</u>

- Parse $\texttt{M}$ into blocks $\texttt{M}_1, \texttt{M}_2, \ldots, \texttt{M}_t$
  // all blocks except M_t are 16 bytes
- For $\texttt{i=1…t}$:
  - $\texttt{C}_i \leftarrow \texttt{M}_i \oplus \texttt{AES}_k\texttt{(IV+i)}$
- Return $\texttt{IV}$, $\texttt{C}_1, \ldots, \texttt{C}_t$

Notes:

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



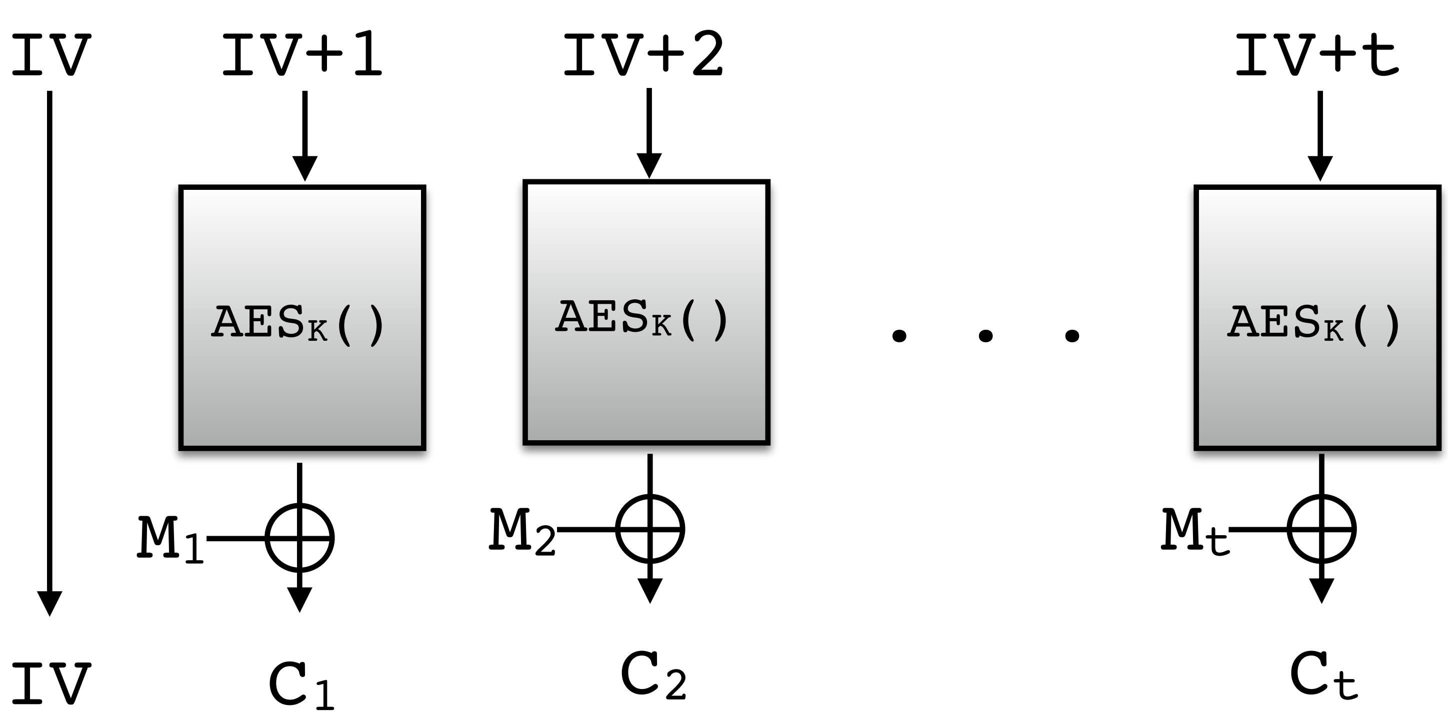
Encrypting large files, Attempt #2: CTR

- 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

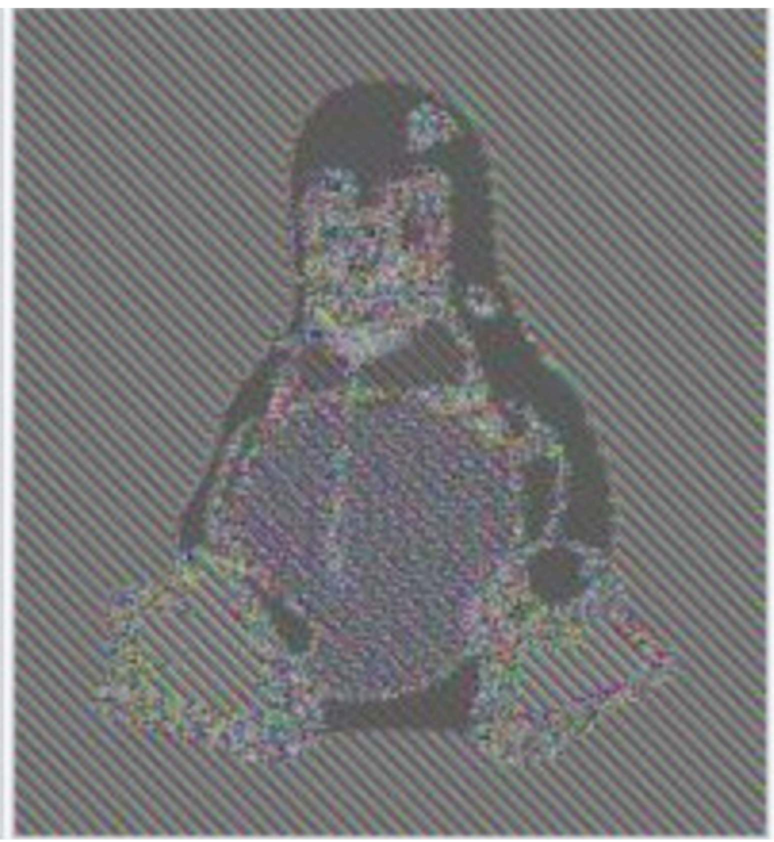


When combined with authentication, CTR is a good cipher.

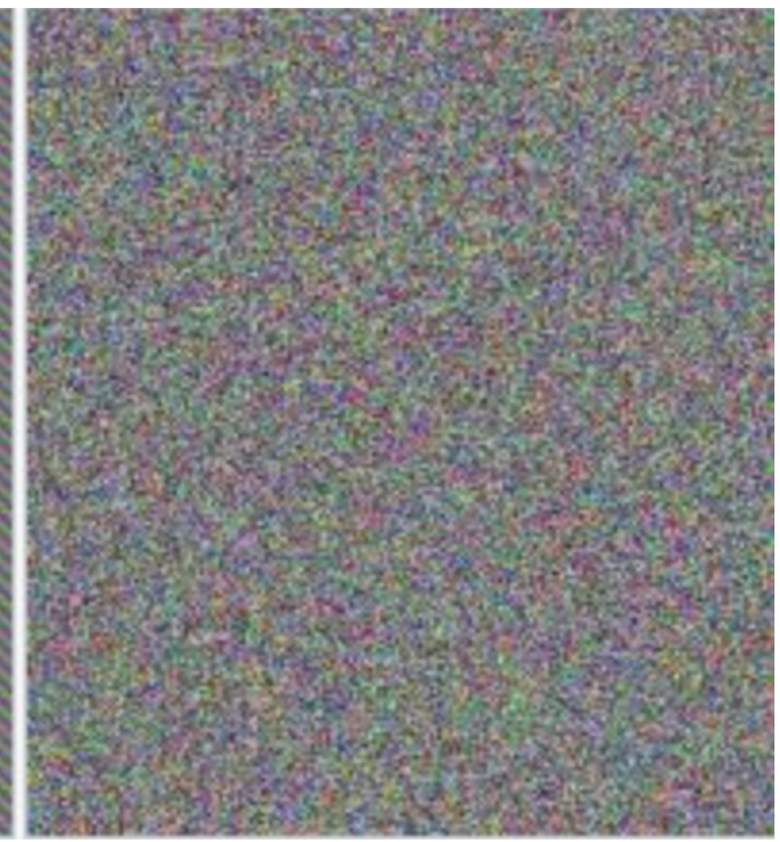
Penguin Sanity Check

Plaintext

ECB Ciphertext



CTR Ciphertext



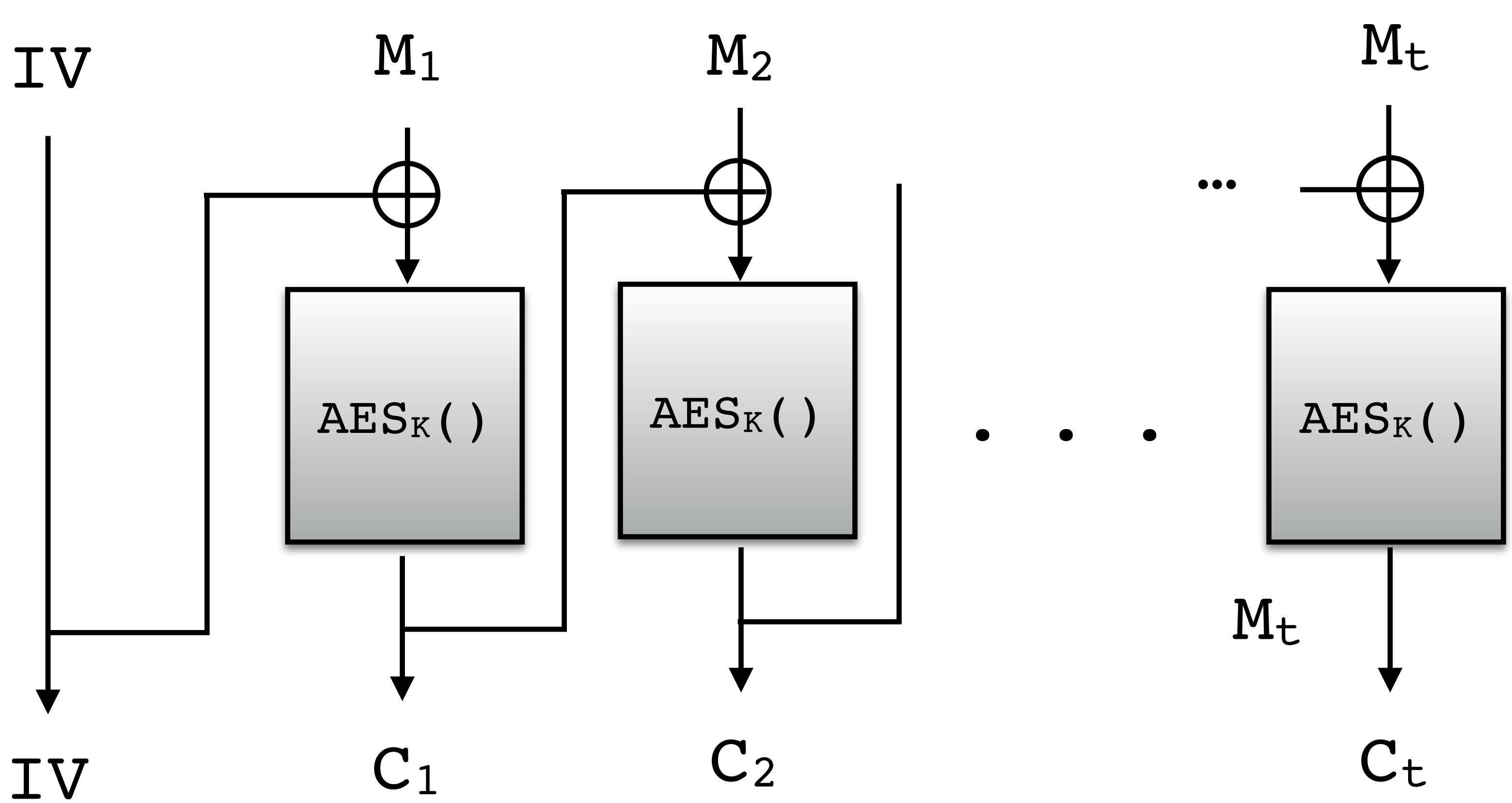
Looks random

Encrypting large files, Attempt #3: CBC

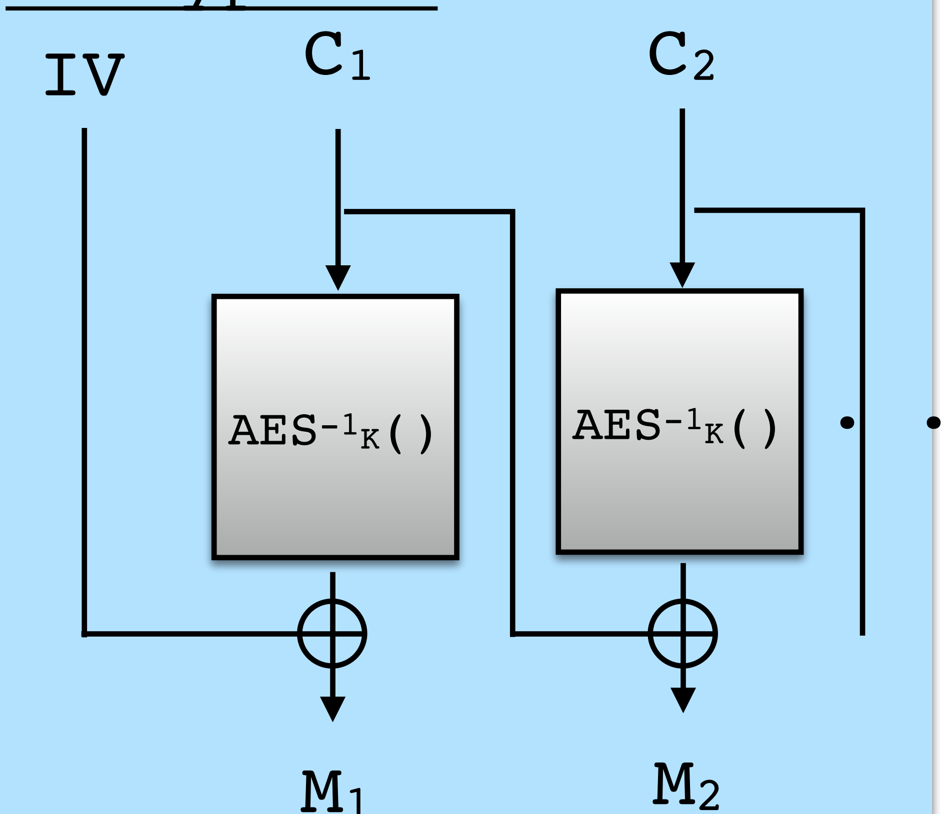
- CBC = "Cipher Block Chaining"
- Nonce-based, but not a stream cipher
- Historical option (sometimes

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

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



Decryption

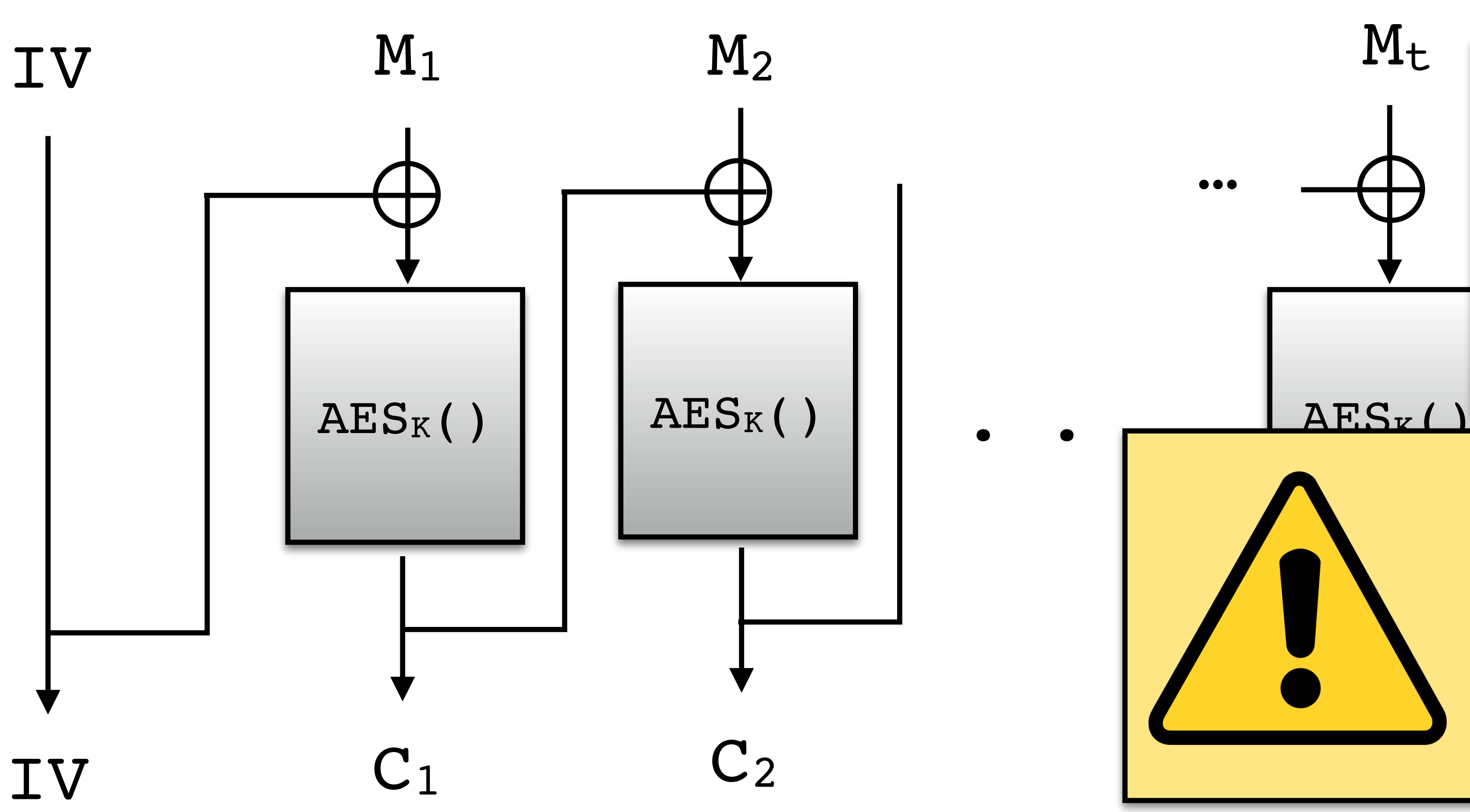


Encrypting large files, Attempt #3: CBC

- CBC = "Cipher Block Chaining"
- Nonce-based, but not a stream cipher
- Historical option (sometimes

<u>**$\texttt{AES-CBC}_k\texttt{(IV,M)}$**</u>

- Parse $\texttt{M}$ into blocks $\texttt{M}_1, \texttt{M}_2, \ldots, \texttt{M}_t$
  // all blocks except $\texttt{M}_t$ are 16 bytes
- Pad $\texttt{M}_t$ up to 16 bytes
- $\texttt{C}_0 \leftarrow \texttt{IV}$
- For $\texttt{i=1...t:}$
  - $\texttt{C}_i \leftarrow \texttt{AES}_k(\texttt{M}_i \oplus \texttt{C}_{i-1})$
- Return $\texttt{C}_0, \texttt{C}_1, \ldots, \texttt{C}_t$



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

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

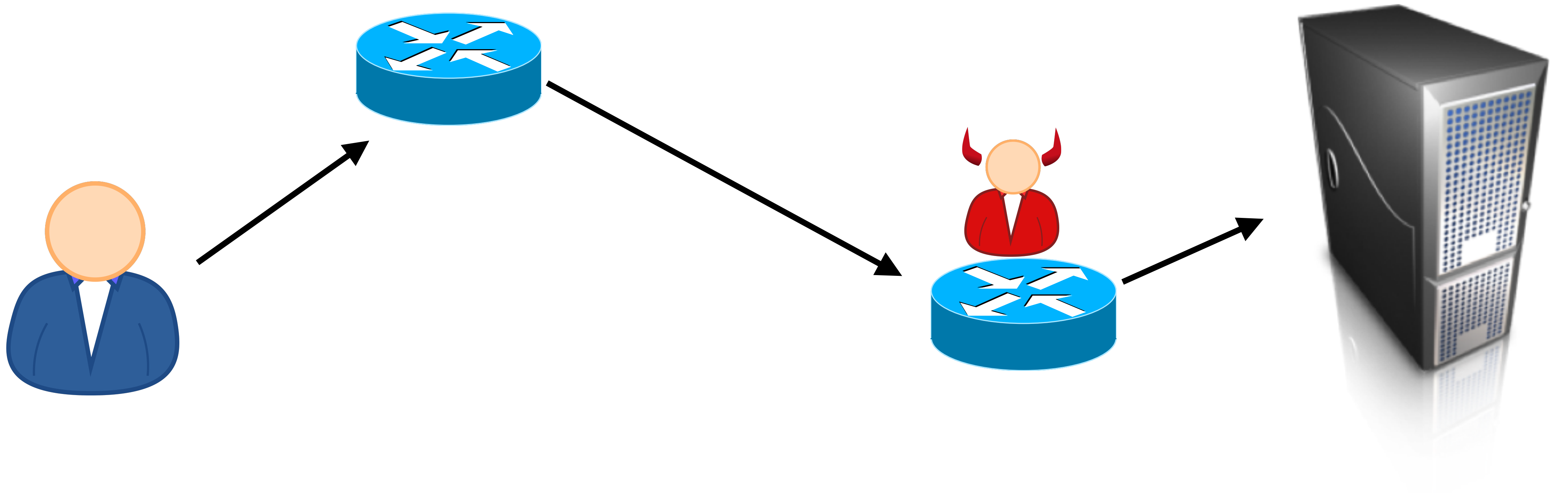
Blockcipher Summary

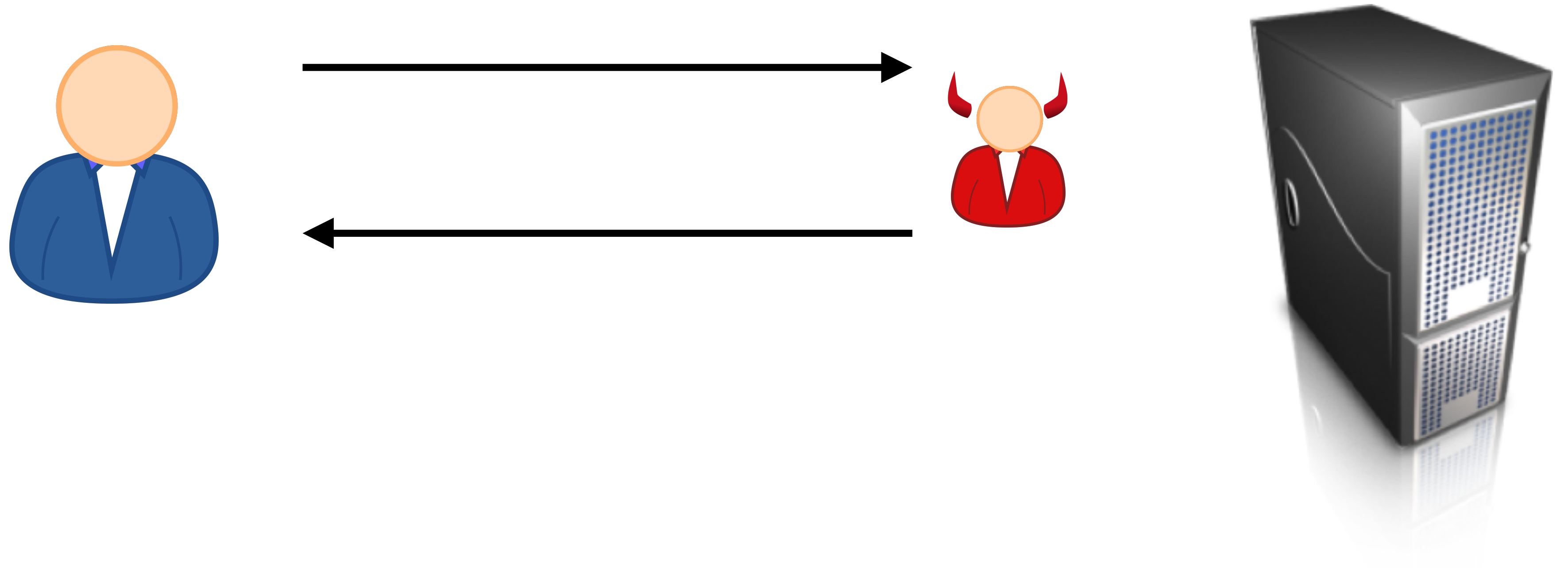
- AES is unbroken
- AES-CTR is most robust construction for confidentiality
- AES-CTR/AES-CBC do not provide authenticity/integrity and should almost never be used alone.

Next Up: Integrity and Authentication

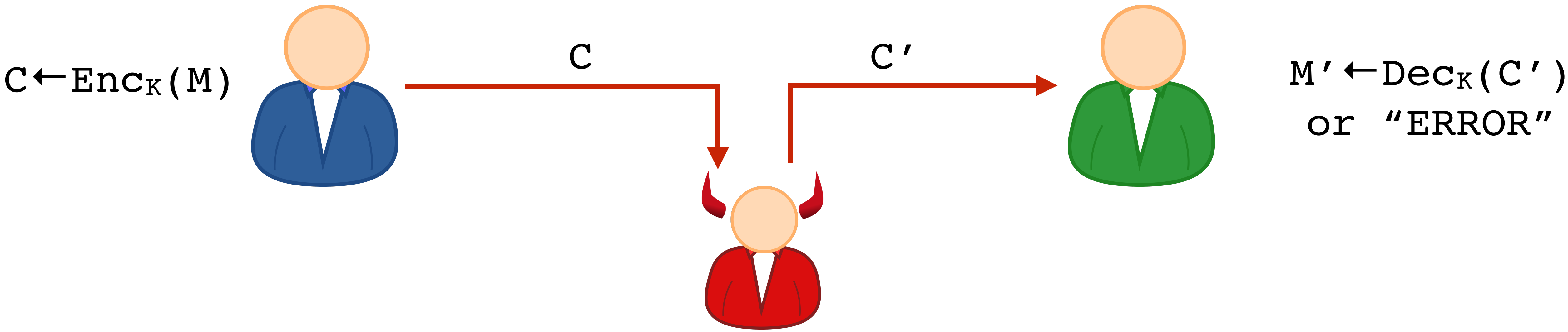
- Authenticity: Guarantee that adversary cannot change or insert ciphertexts
- Achieved with MAC = “Message Authentication Code”

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`

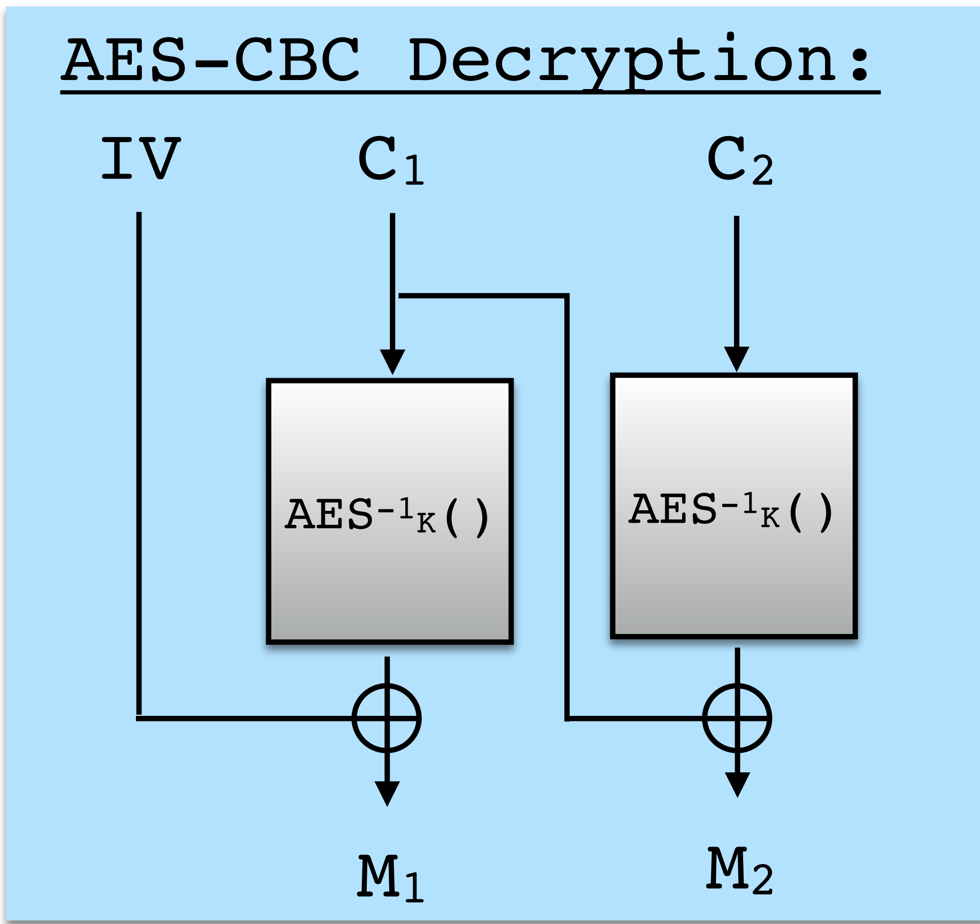
`C = b0595fafd05df4a7d8a04ced2d1ec800d2daed851ff509b3e446a782871c2d`

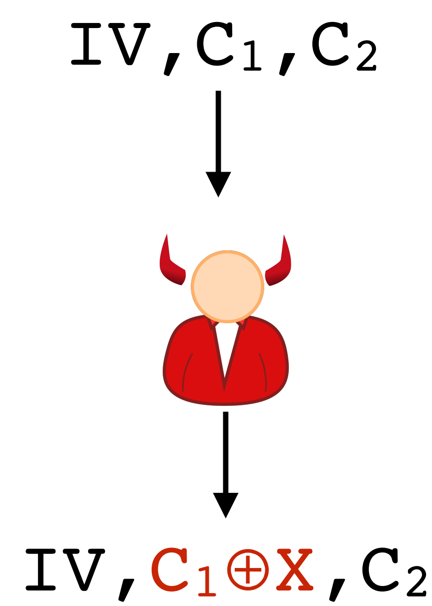
`C’= b0595fafd05df4a7d8a04ced2d1ec800d2daed851ff509b3e546a782871c2d`

`M’ = please pay ben 21 bucks`

Inherent to stream-cipher approach to encryption.

AES-CBC does not satisfy integrity





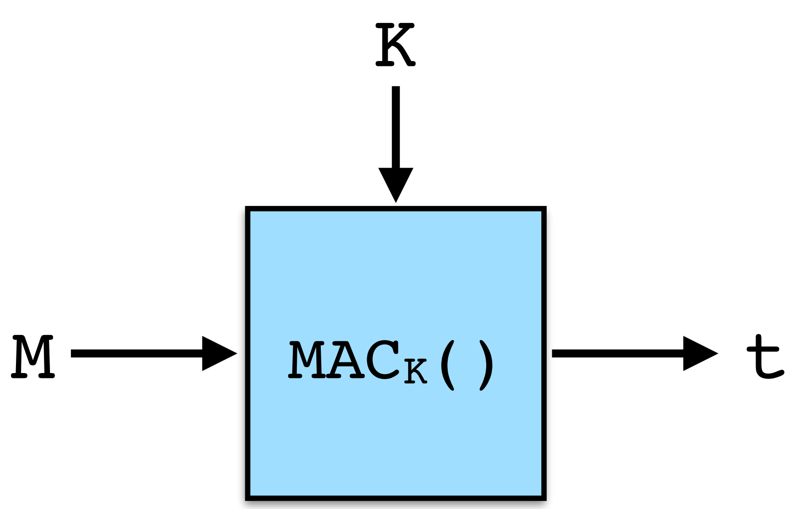
Decrypts to:

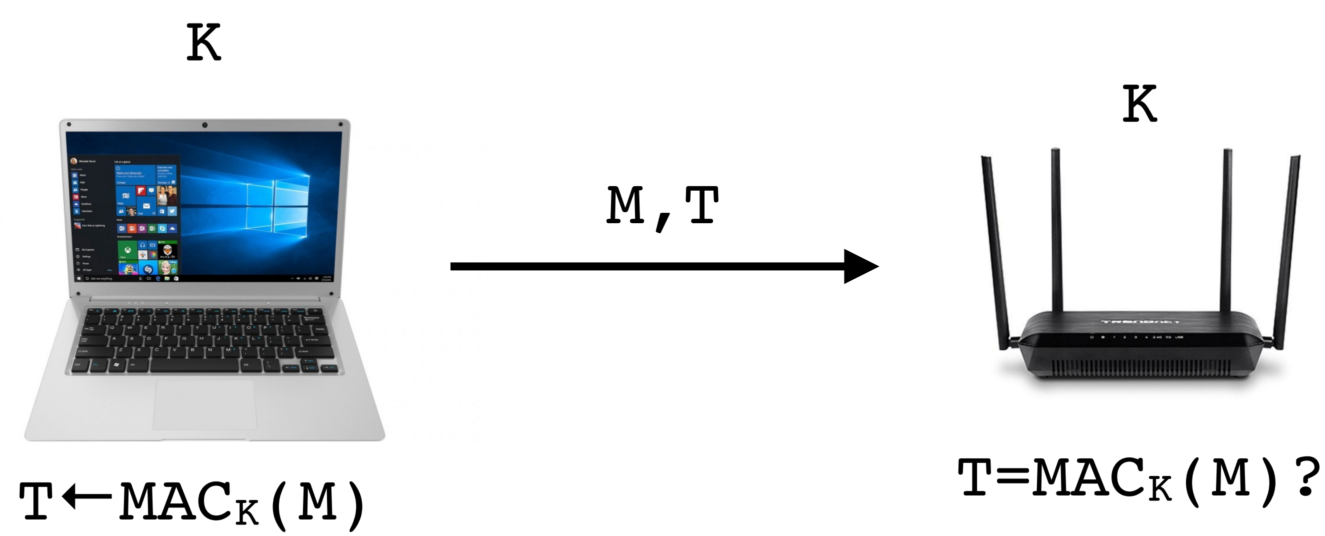
$R, M_2 \oplus X$

Where R is some unpredictable block.

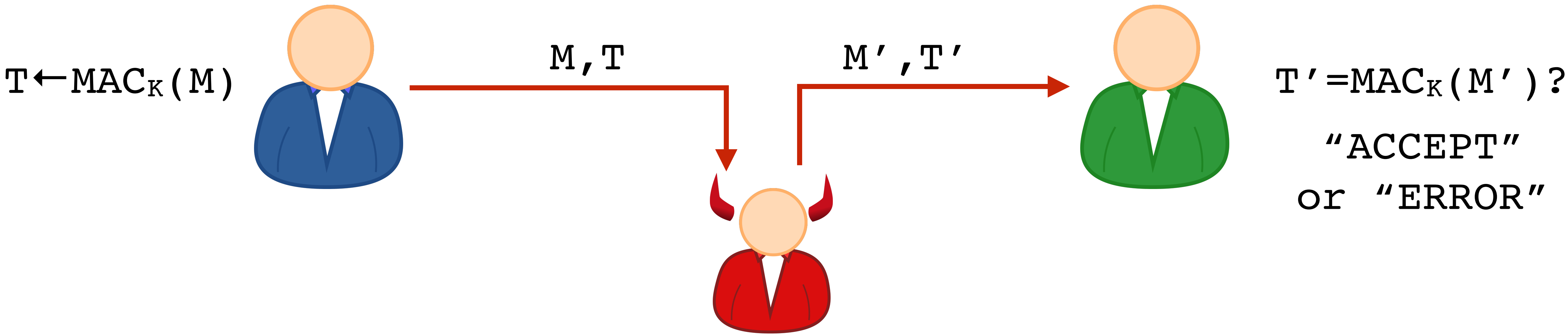
Message Authentication Code

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



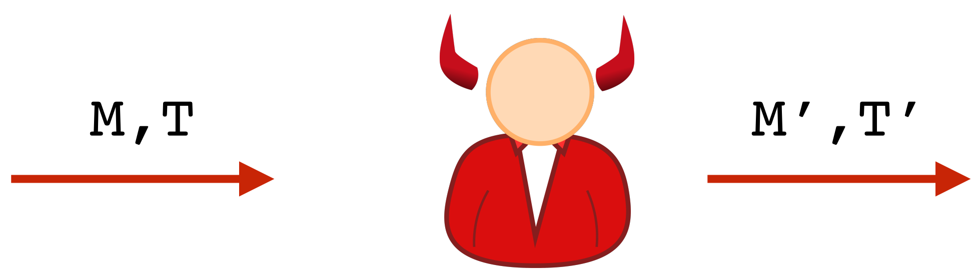
MAC satisfies **unforgeability** if it is unfeasible for Adversary to fool Bob into accepting `M′` not previously sent by Alice.

MAC Security Goal: Unforgeability

Note: No encryption on this slide.

`M = please pay ben 20 bucks`

`T = 827851dc9cf0f92ddcdc552572ffd8bc`



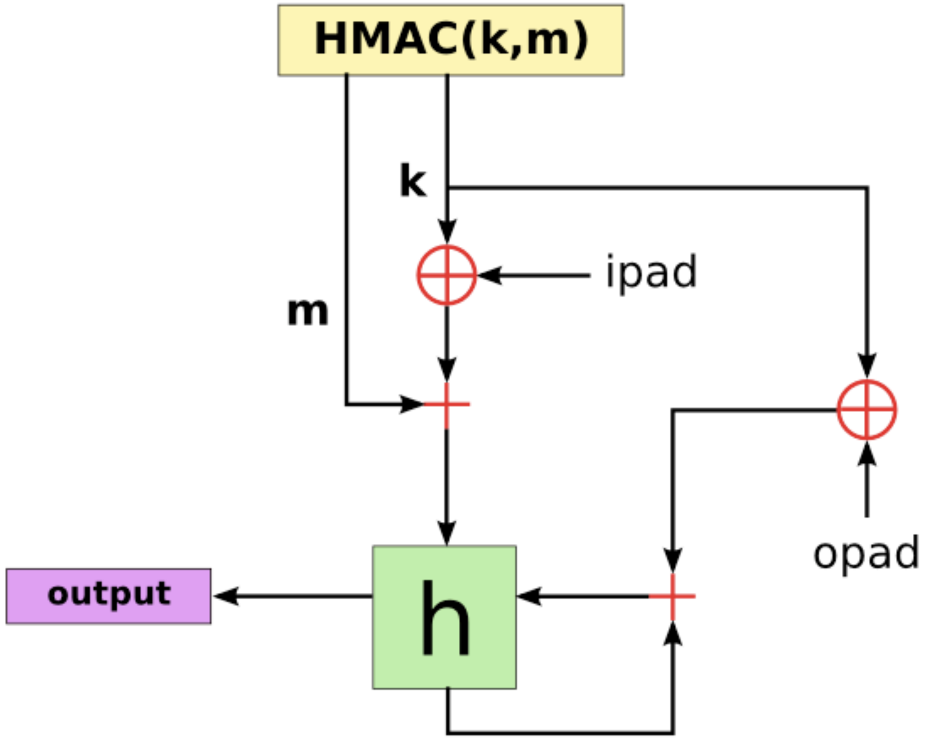
`M'= please pay ben 21 bucks`

`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.



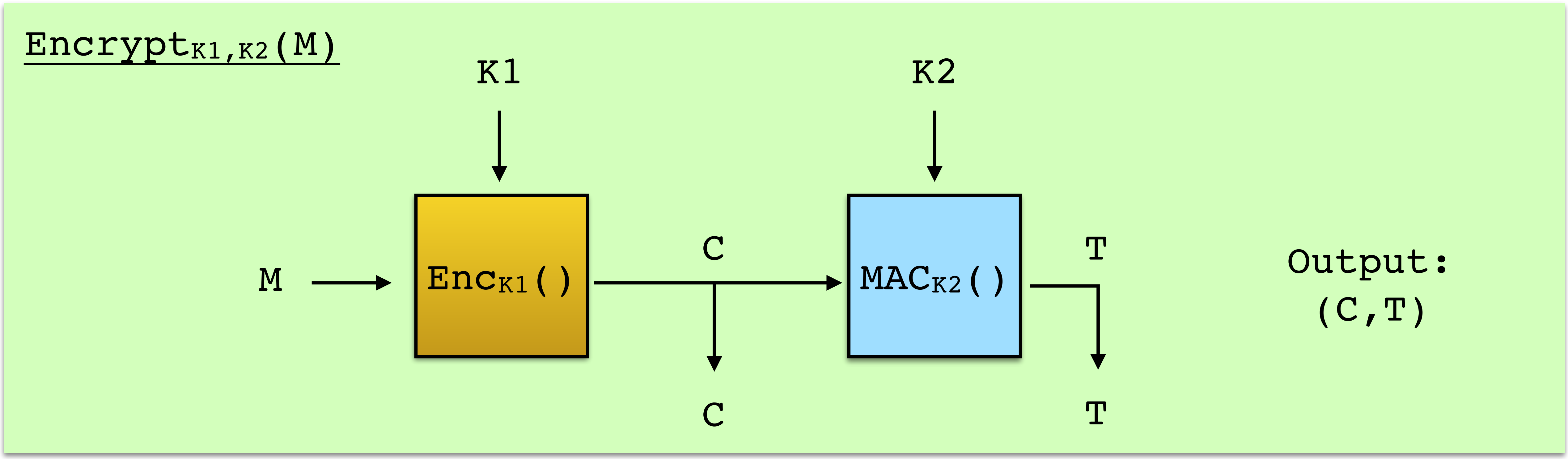
- Other options: Poly1305-AES or CBC-MAC (the latter is tricky)

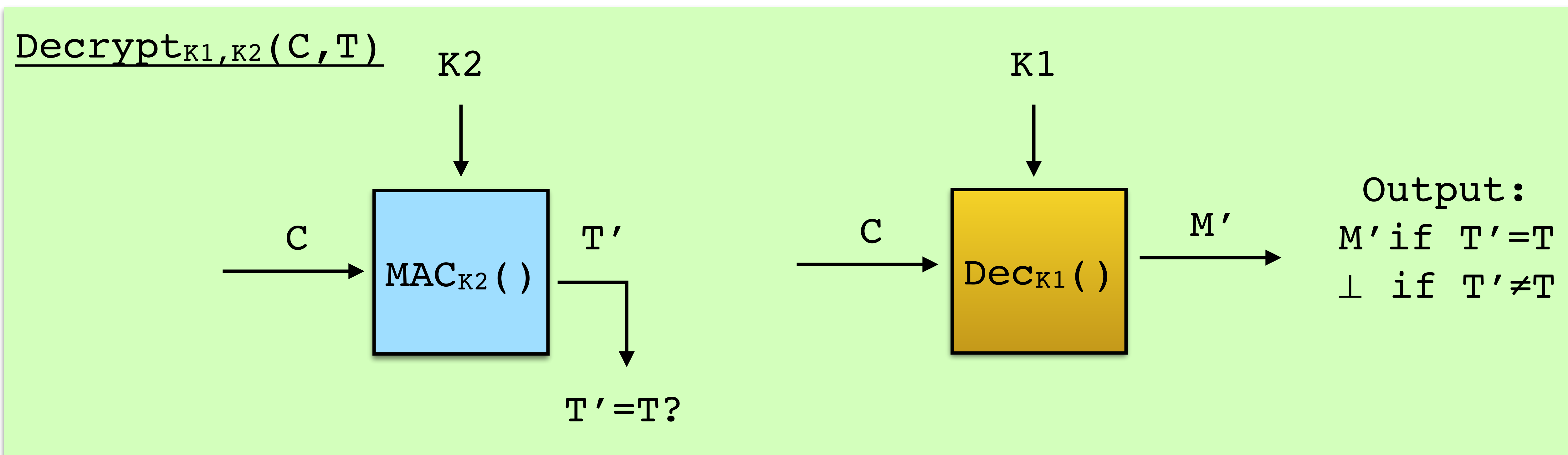
Authenticated Encryption

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

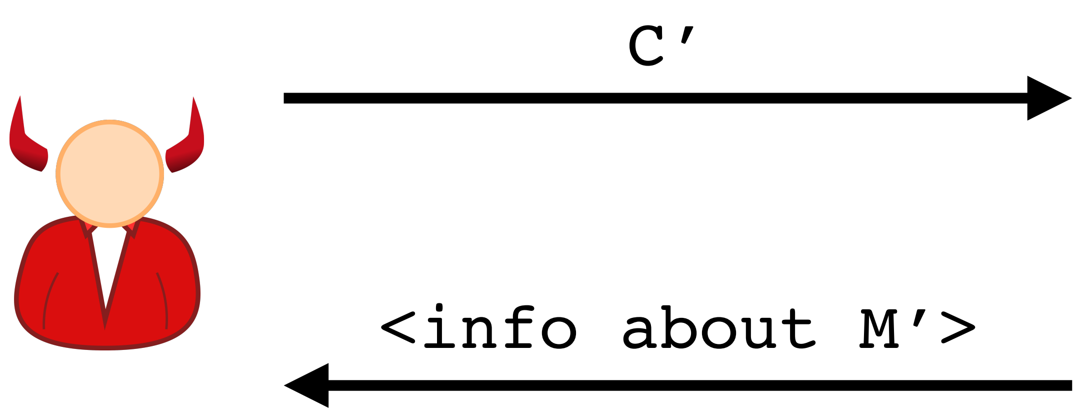


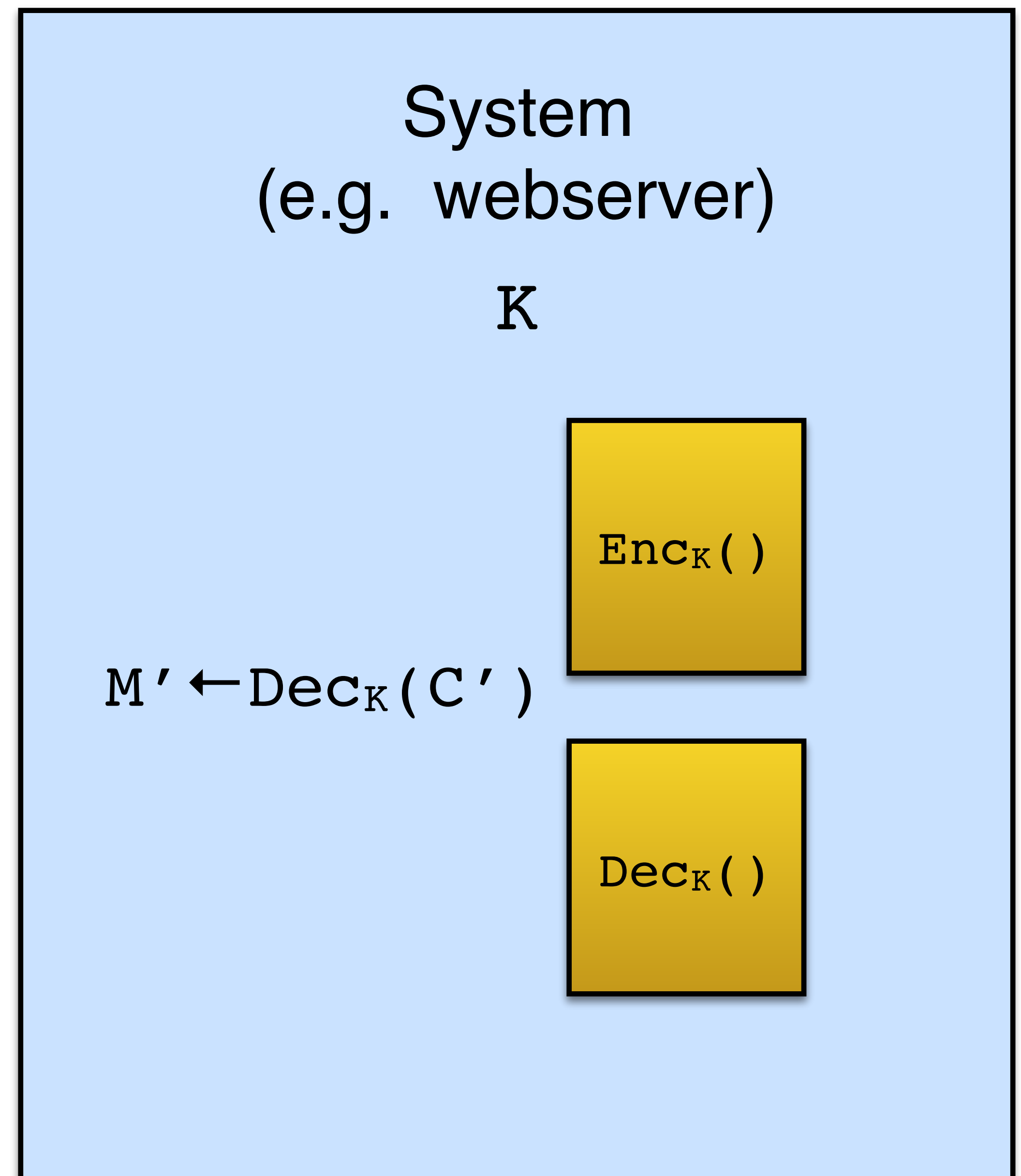


- Summary: MAC the ciphertext, not the message

Chosen-Ciphertext Attacks (CCA) against Encryption

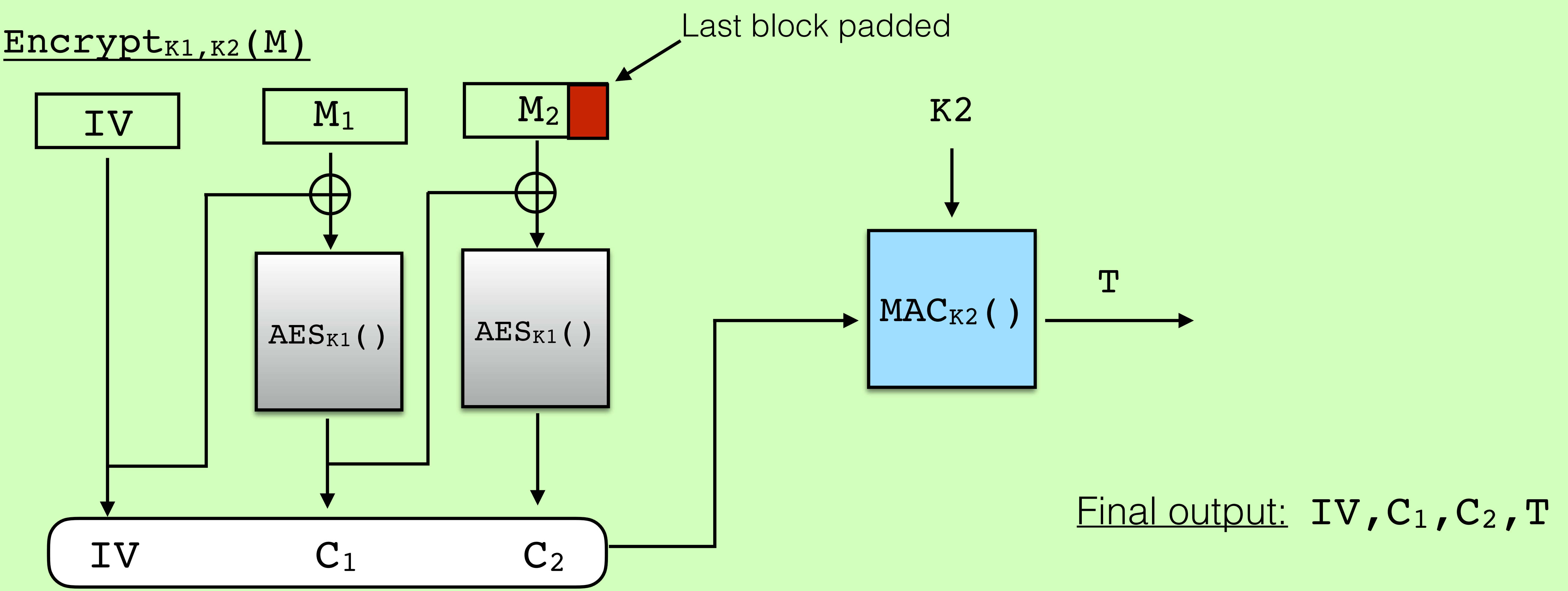
- Integrity + Confidentiality = security against CCAs





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

CBC-Based Auth. Enc. Error: Padding and MACs



Decrypt$_{K1,K2}$(IV, C_1, C_2, T)

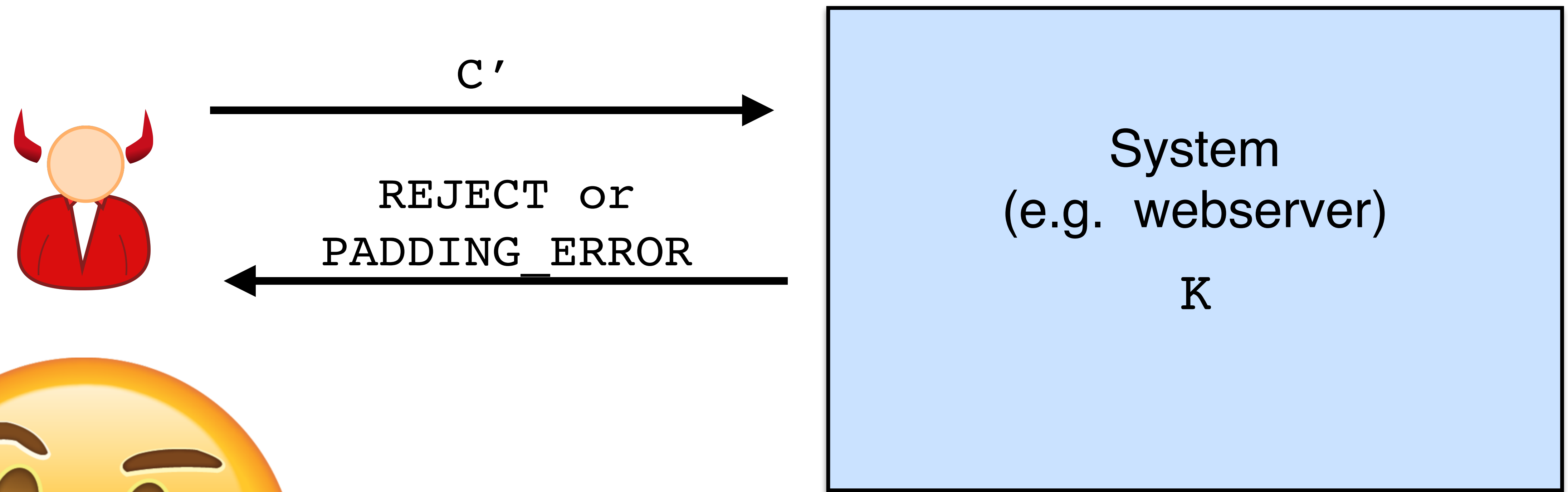
1. If tag T wrong:
   Output REJECT
2. M′ ← CBC-Decrypt$_{K1}$(IV, C_1, C_2)
3. If padding format wrong:
   Output PADDING_ERROR
4. Output M′

Decrypt$_{K1,K2}$(IV, C_1, C_2, T)

1. M′ ← CBC-Decrypt$_{K1}$(IV, C_1, C_2)
2. If padding format wrong:
   Output PADDING_ERROR
3. If tag T wrong:
   Output REJECT.
4. Output M′

Broken

Padding Oracle Attacks



Allows decryption of arbitrary ciphertexts by adversary!
… also by you, in Assignment 1.

<u>$\text{Decrypt}_{K1,K2}(IV, C_1, C_2, T)$</u>

1. $M' \leftarrow \text{CBC-Decrypt}_{K1}(IV, C_1, C_2)$
2. If padding format wrong:
   Output PADDING_ERROR
3. If tag T wrong:
   Output REJECT.
4. Output M'

Broken

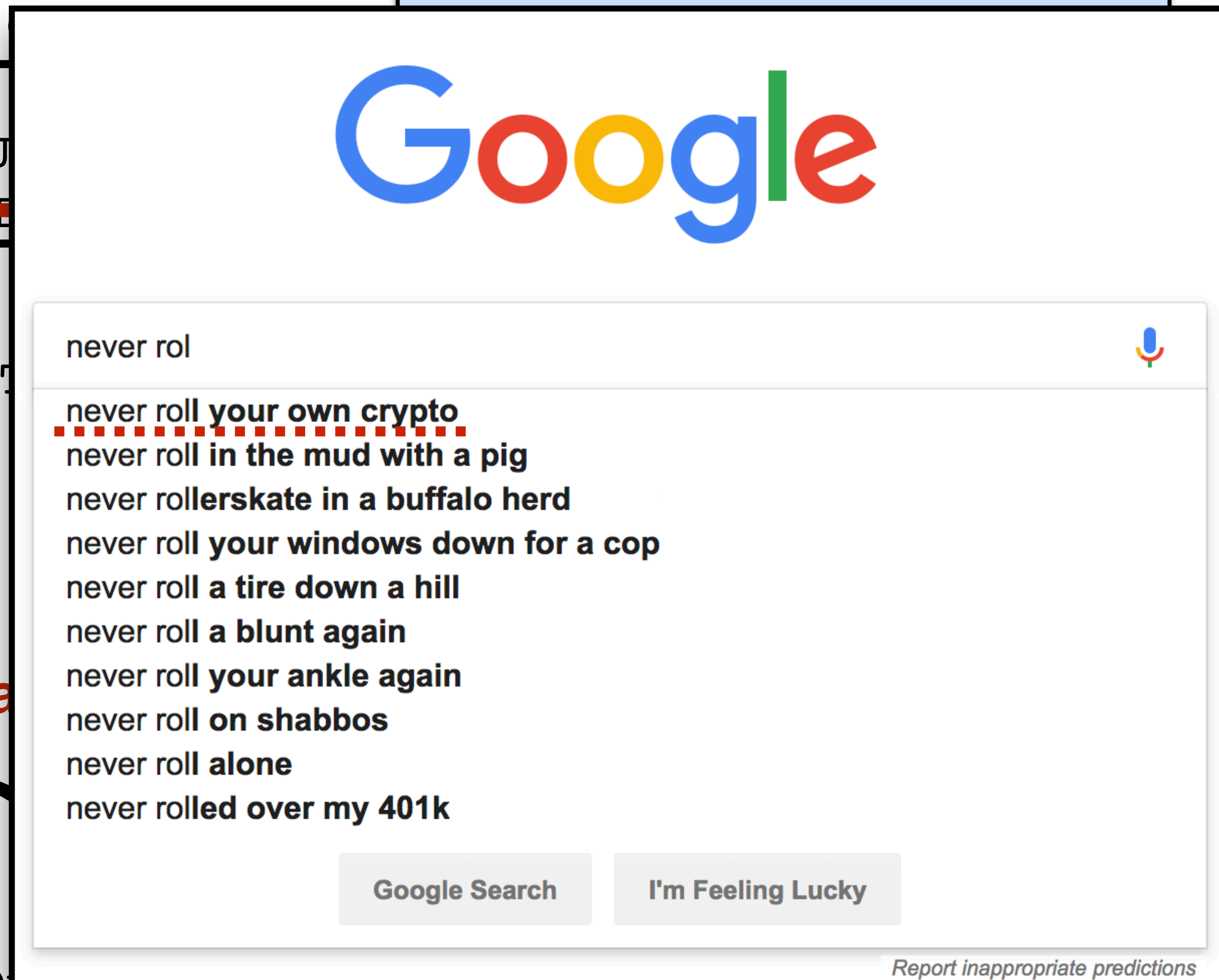
Padding Oracle Attacks: It gets worse

REJ

PADDI

REJECT

REJECT



Output REJECT lines will ta
different times to reach:

Attack still possible.

Output REJECT.

```
3. If tag T wrong:
     Output REJECT.
4. Output M'
```

Broken

<u>Solutions:</u>

1. Constant-time code (extremely difficult).
2. Use un-padded encryption like CTR.

The End