Encryption Attack Details, Hash Functions

CMSC 23200/33250, Autumn 2018, Lecture 5

David Cash

University of Chicago

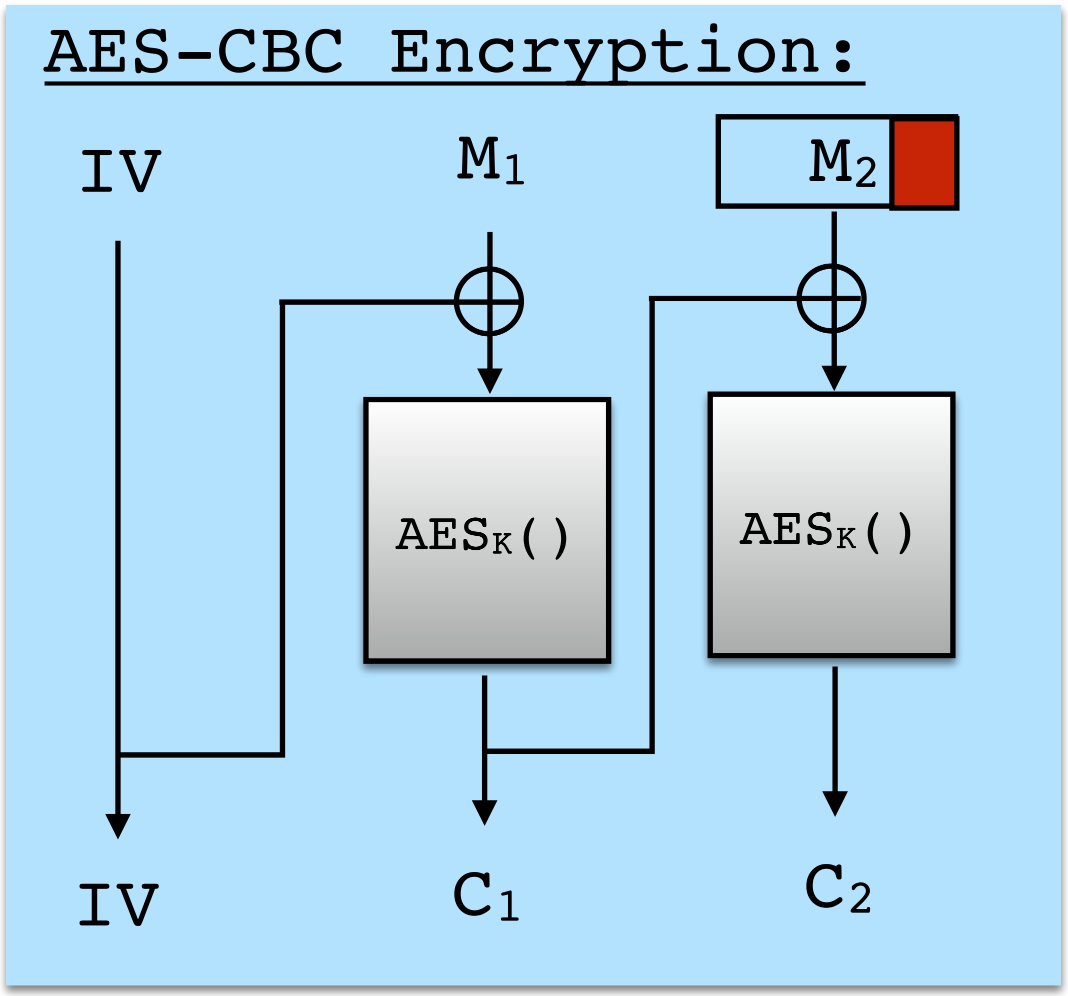
Plan

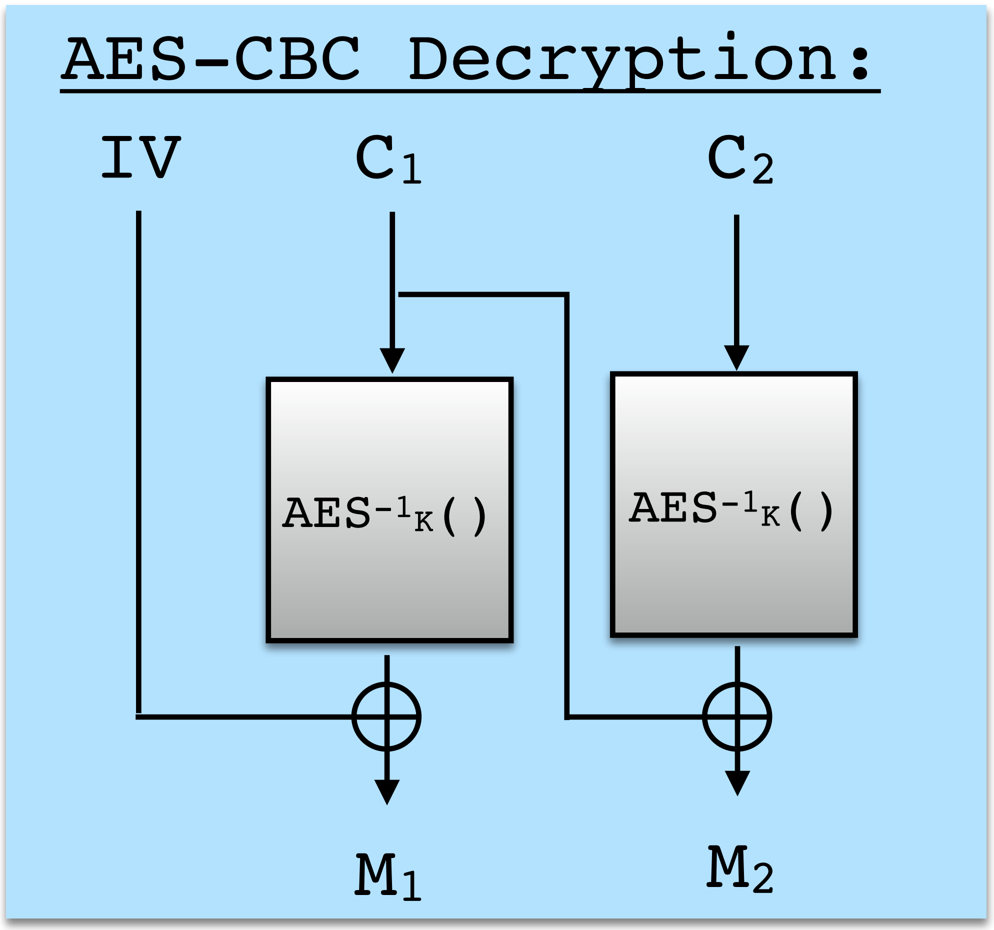
1. Padding Oracle Attack Details
2. Introduction to Cryptographic Hash Functions
3. Pitfalls with Hash Functions
4. Begin Public-Key Encryption

Assignment 1 is Coming Today

Updated due-date will be announced

AES-CBC with Padding





PKCS7 Padding

- Need to pad a byte string up to a multiple of 16 bytes
- First look at how many bytes are missing. Here, need 10 bytes
- Fill missing k bytes with value k (k = 10 = 0x0A in example)

`48 65 6C 6C 6F 20 43 53 31 37 37 20 73 74 75 64` `65 6E 74 73 21 00` `0A 0A 0A 0A 0A 0A 0A 0A 0A 0A`

- If data is already a multiple of 16 bytes long, add an entire block of 0x10 bytes

`48 65 6C 6C 6F 20 43 53 31 37 37 20 73 74 02 02` `10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10`

Can't leave data unchanged;
Bytes might be interpreted as padding.

- Un-padding is easy

Invalid Padding

Fact: Not every byte string is a “valid padding”. Some strings have to be handled as “malformed”

`48 65 6C 6C 6F 20 43 53 31 37 37 20 73 74 75 64`	`48 65 6C 6C 6F 20 43 53 31 37 37 20 73 74 01 02`

Invalid. Why?

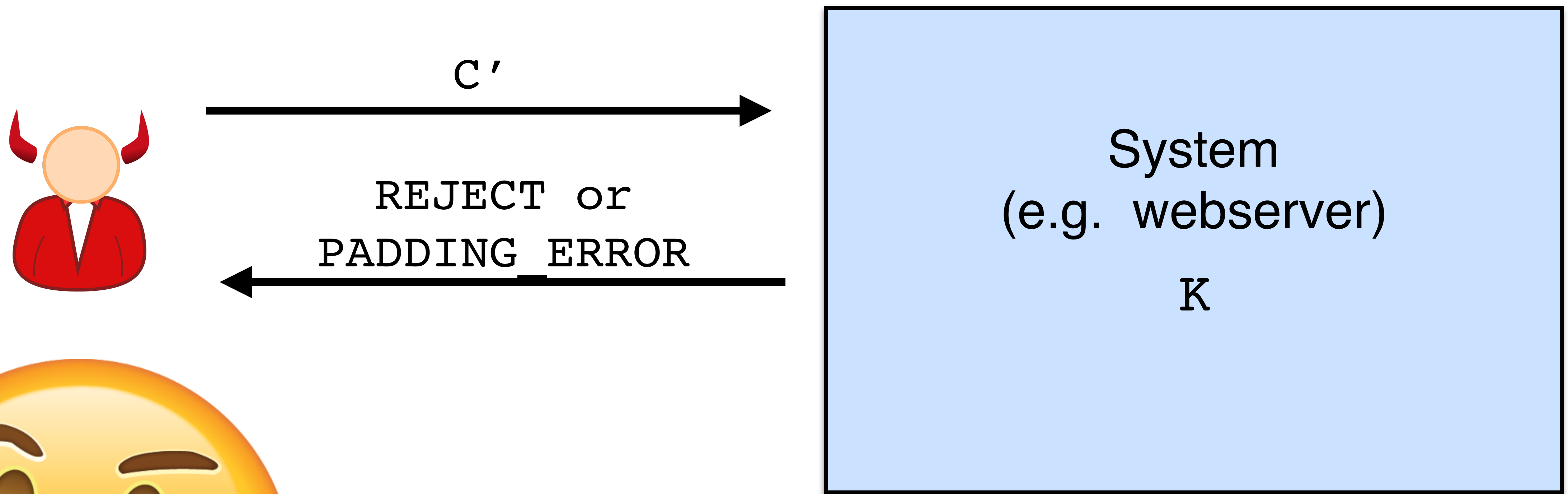
`48 65 6C 6C 6F 20 43 53 31 37 37 20 73 74 75 64`	`48 65 6C 6C 0A 0A 0A 0A 0A 0A 0A 0A 0A 0A 0A 0A`

Valid. Why?

`48 65 6C 6C 6F 20 43 53 31 37 37 20 73 74 75 64`	`48 65 6C 6C 0A 0A 0A 0A 0A 0A 0A 0A 0A 0A 0A 01`

Valid! All strings that end in 0x01 are valid.

Padding Oracle Attacks



Real-world attacks against:
- TLS (2003)
- IPSec (2007,2010)
- Ruby on Rails (2010)
- ASP.NET (2010)
- SecurID Auth Tokens (2012)
- Steam Client (2016)

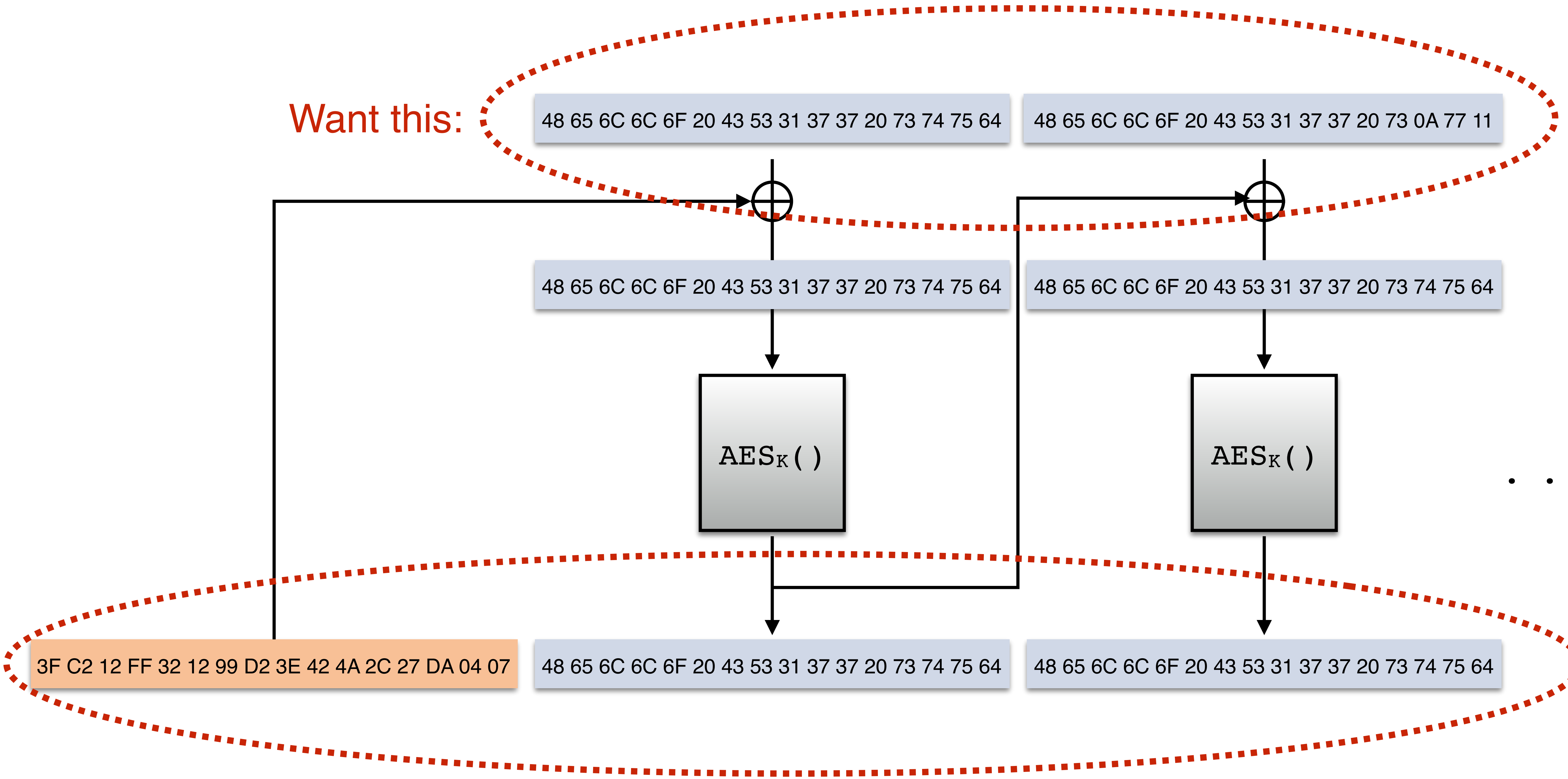
```
,K2(IV,C1,C2,T)
C-Decrypt_K1(IV,C1,C2)
dding format wrong:
put PADDING_ERROR
3. If tag T wrong:
     Output REJECT.
4. Output M'
```

Broken

Now: How to find <u>one byte</u> of plaintext.

Attack Setting

- First two blocks of long valid ciphertext are pictured.

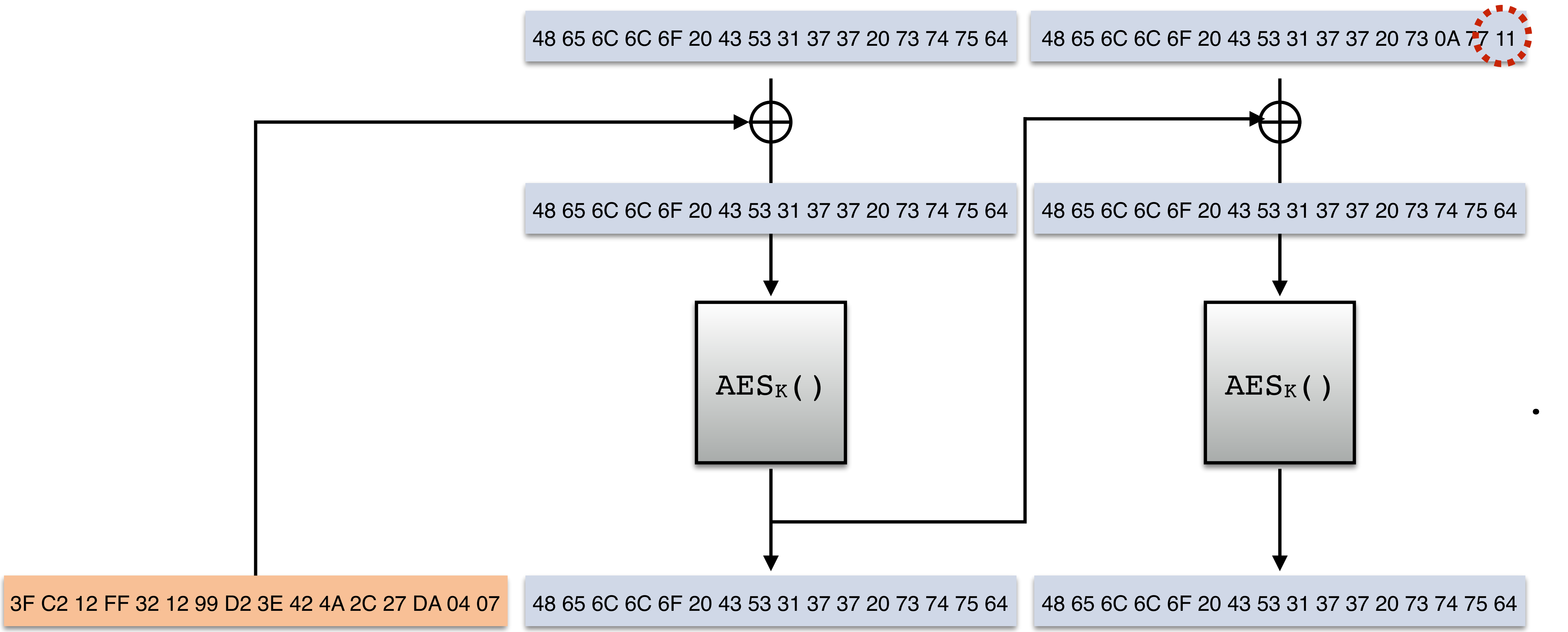


Have this, plus the padding oracle.

Initial goal: Learn last byte of a block

- First observation: If we truncate the long ciphertext down to these blocks, what will padding oracle say?
  - `Padding_Error`!

Let's learn this:

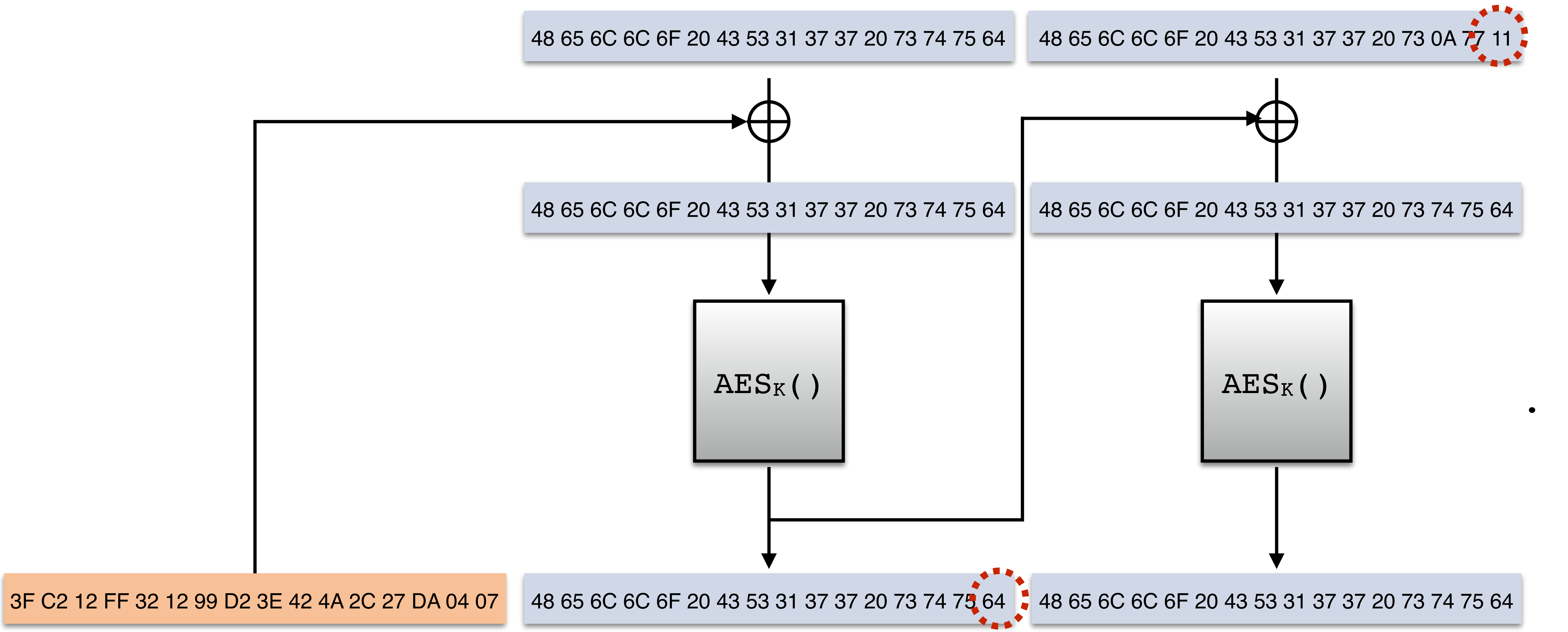


- What can we infer from the padding error response?
  - Last plaintext byte of that block could not have been 0x01.

Initial goal: Learn last byte of a block.

- What will padding oracle say?
  - (still) `Padding_Error`!

After decryption, this byte becomes 0x10.

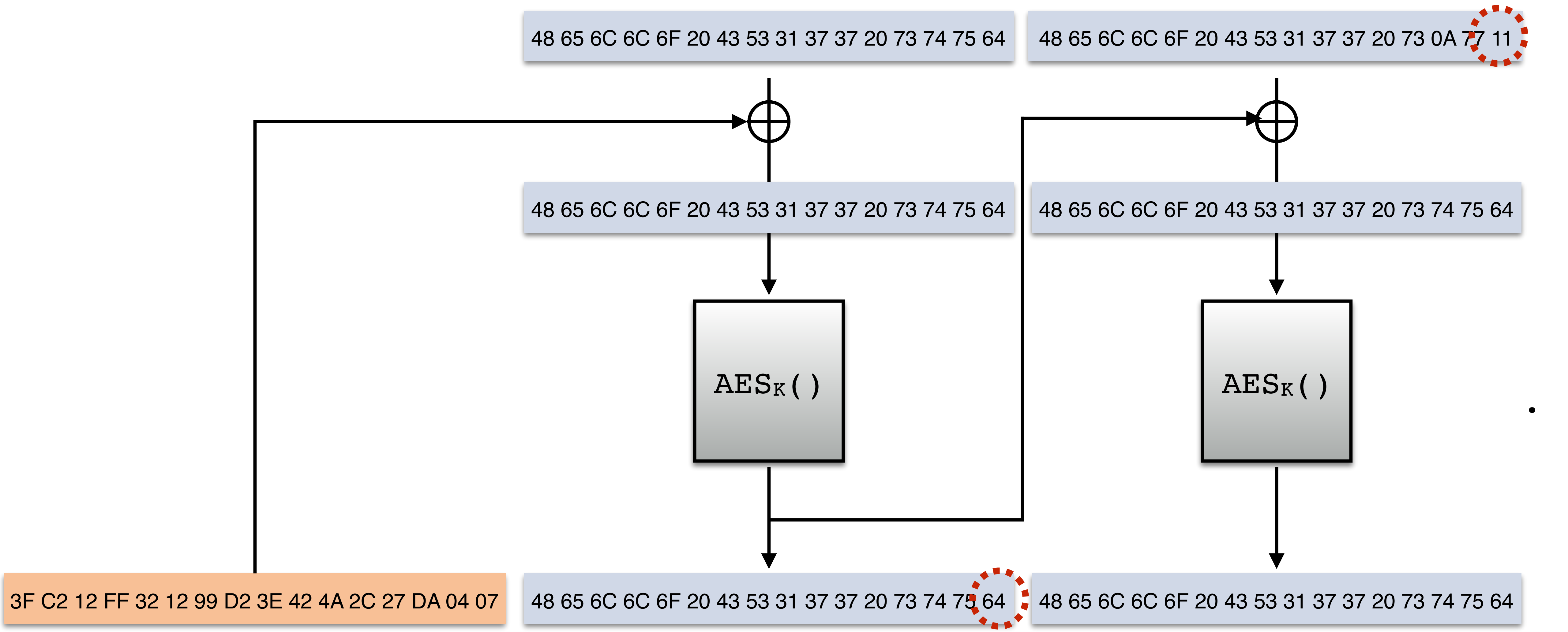


Consider changing this byte to 0x65, then submitting ciphertext to oracle.

Initial goal: Learn last byte of a block.

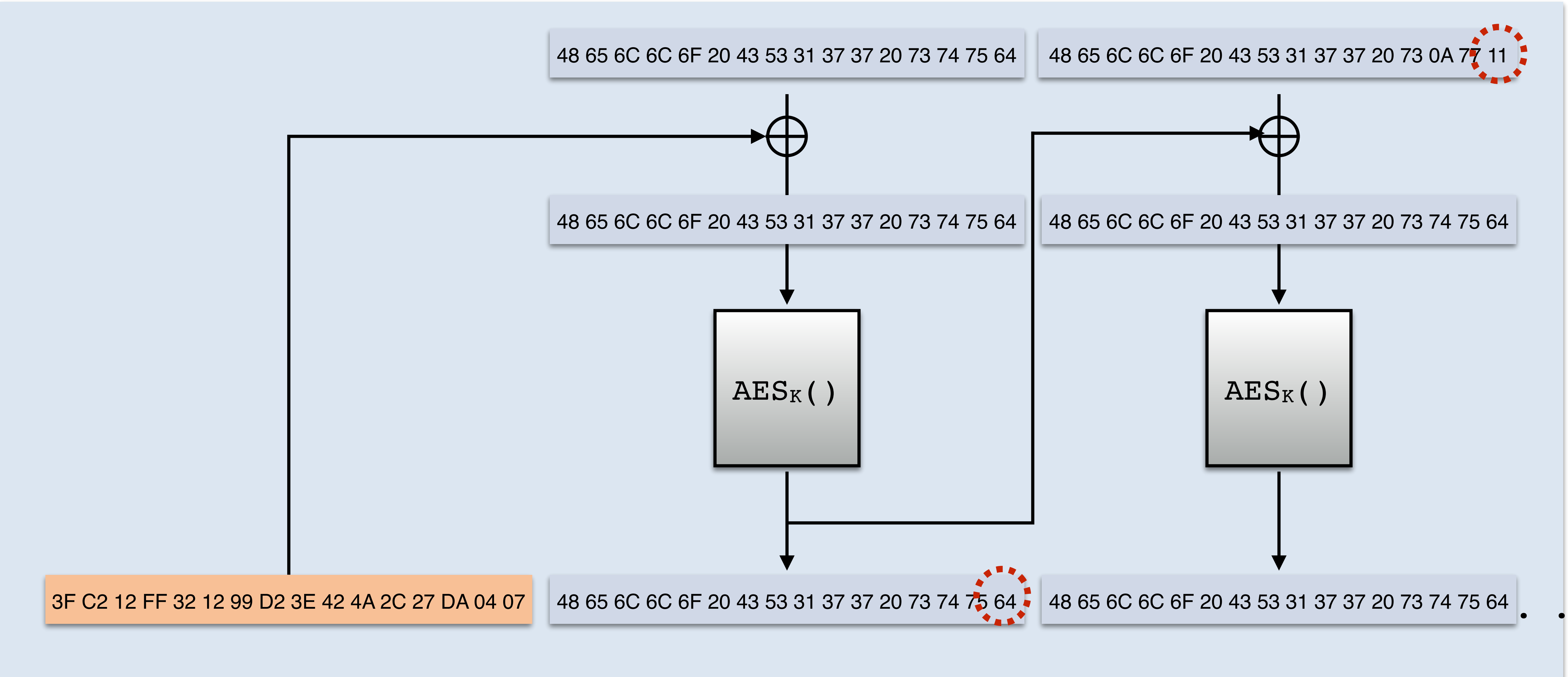
- What will padding oracle say?
  - Padding is valid!
  - Infer value of last byte: 0x10$\oplus$0x01 = 0x11

After decryption, this byte becomes 0x01.



Consider instead changing this byte to 0x10, then submitting ciphertext to oracle.

Initial goal: Learn last byte of a block.

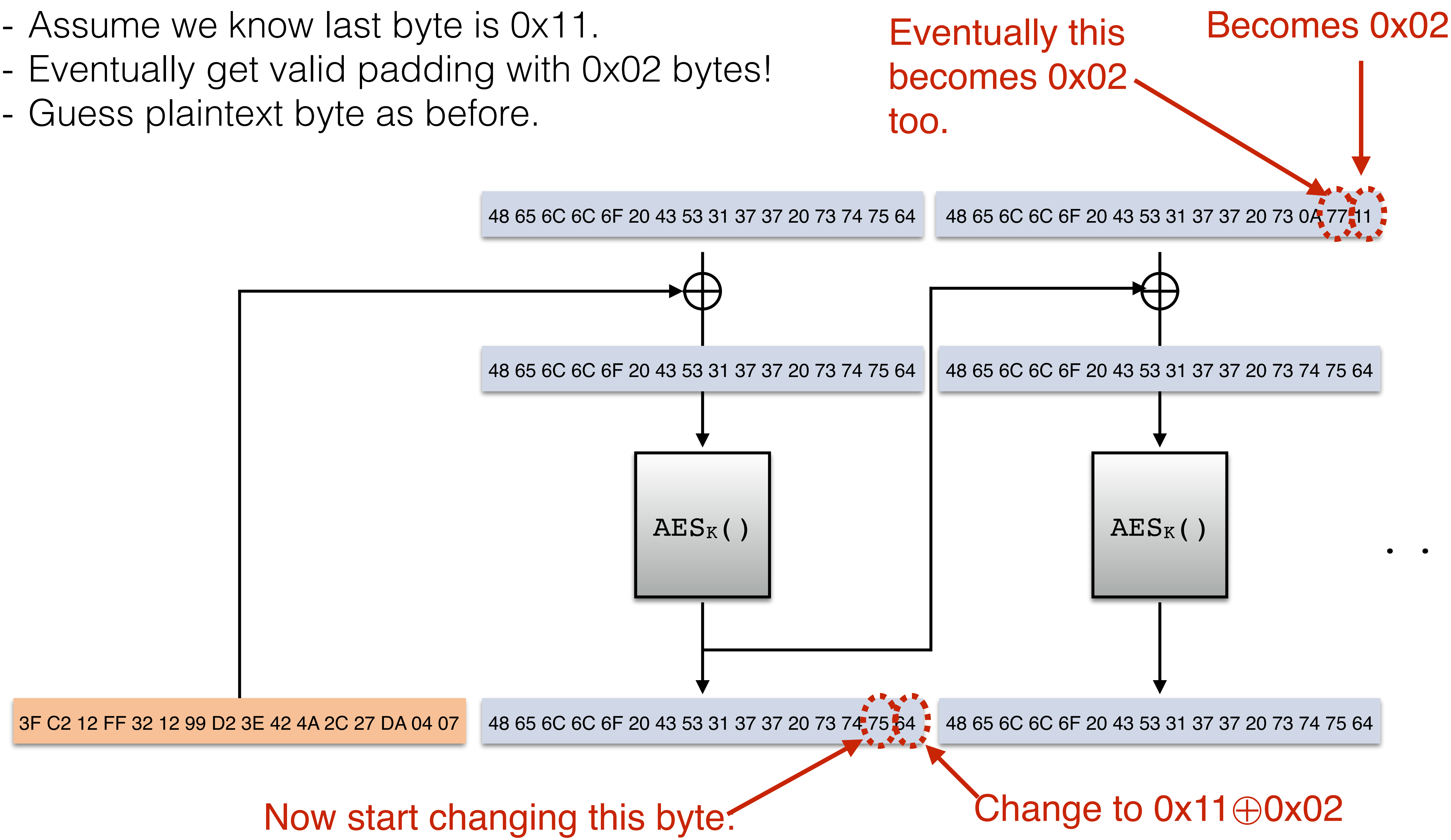


Fact: If last byte of plaintext is `X`, and we change last byte of prior ciphertext block to `X`$\oplus$`0x01`, then padding will be valid. Changing that byte to anything else will usually result in invalid padding.

Q: How do we know what to set that byte to? **A:** Just try all 256 until one works.

Recovering more plaintext bytes

- Assume we know last byte is 0x11.
- Eventually get valid padding with 0x02 bytes!
- Guess plaintext byte as before.



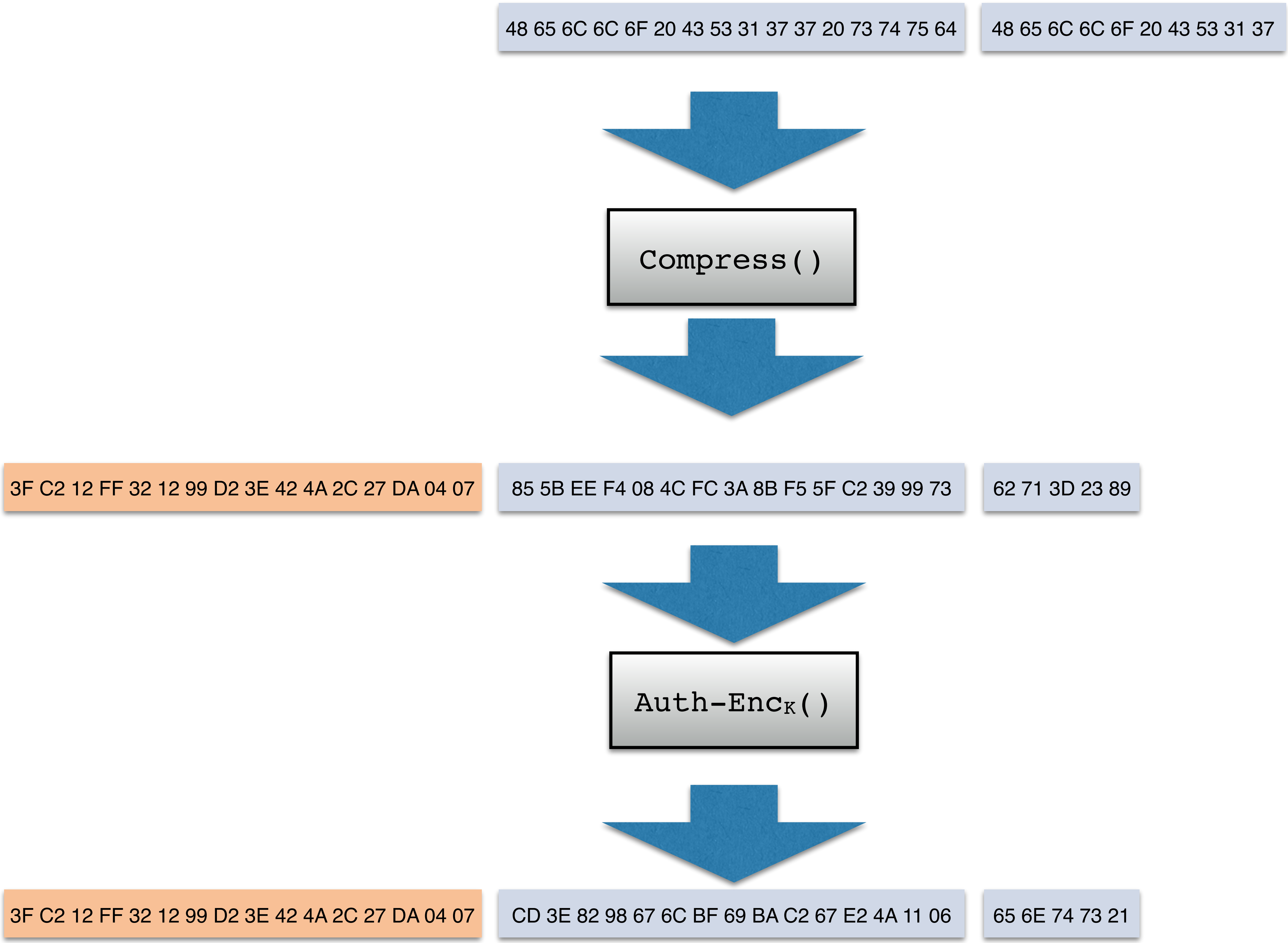
Assignment 1: Full plaintext recovery

- Extend idea using cases with larger padding to recover entire block
- Nothing special about the second block; We can take any block and apply attack.
- Some possibility for false positives (be careful).

Lessons

- Always use Authenticated Encryption, with properly implemented integrity checks.
- If a library provides Authenticated Encryption, always try to use it.
  - AES-GCM is a good choice
- If not, then use AES-CTR + a good MAC like HMAC.
  - It’s easy and compact
  - Components are very common
  - Don’t implement AES-GCM, especially if you don’t know what a Galois Field is. (But don’t even if you do)

Another avenue for attacks: Compress-then-Encrypt



Compression Attack Setting: Browser Data

token=uchic4gos3cr34;userdata=…

Secret info

Username, etc
(Adversary controlled)

Real-world attacks against:
- TLS (2012)
- HTTPS (2013)

Compress()

3F C2 12 FF 32 12 99 D2 3E 42 4A 2C 27 DA 04 07 | 85 5B EE F4 08 4C FC 3A 8B F5 5F C2 39 99 73 | 62 71 3D 23 89

Adversary can see ciphertext length.

More overlap with secret means better compression, i.e shorter ciphertext.

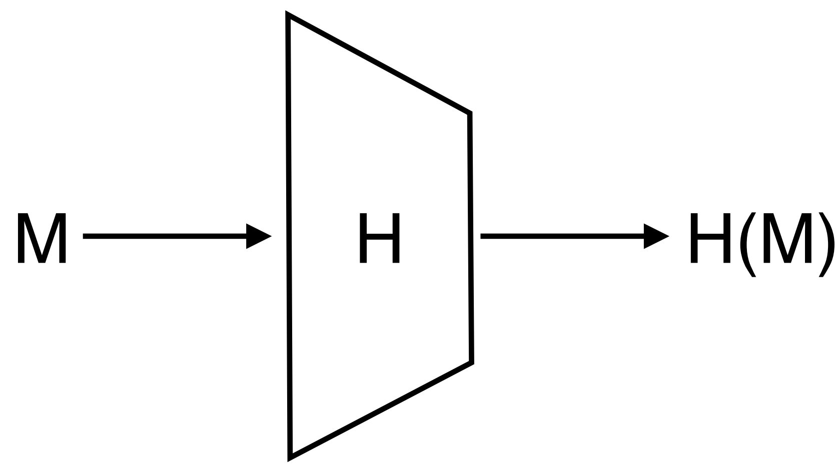
Auth-Enc$_K$()

Assignment 1: Seeing several ciphertexts enables full plaintext recovery.

3F C2 12 FF 32 12 99 D2 3E 42 4A 2C 27 DA 04 07 | CD 3E 82 98 67 6C BF 69 BA C2 67 E2 4A 11 06 | 65 6E 74 73 21

Next Up: Hash Functions and MACs

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



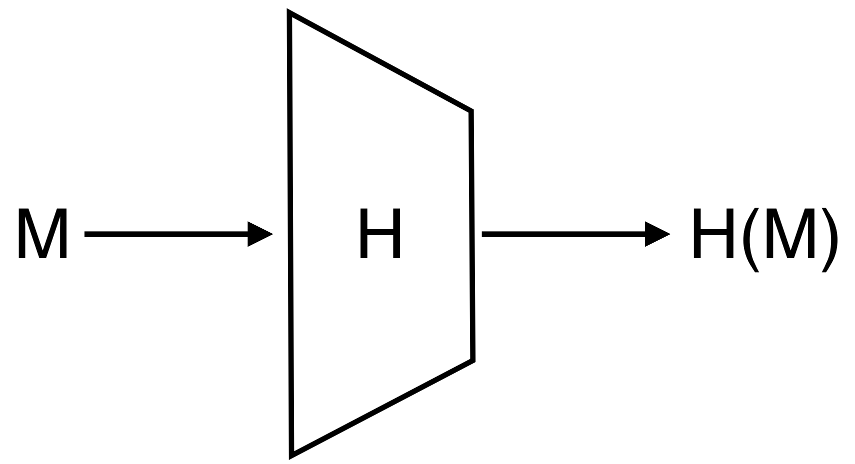
	<u>Output length</u>
MD5:	m = 128 bits
SHA-1:	m = 160 bits
SHA-256:	m = 256 bits
SHA-512:	m = 512 bits
SHA-3:	m >= 224 bits

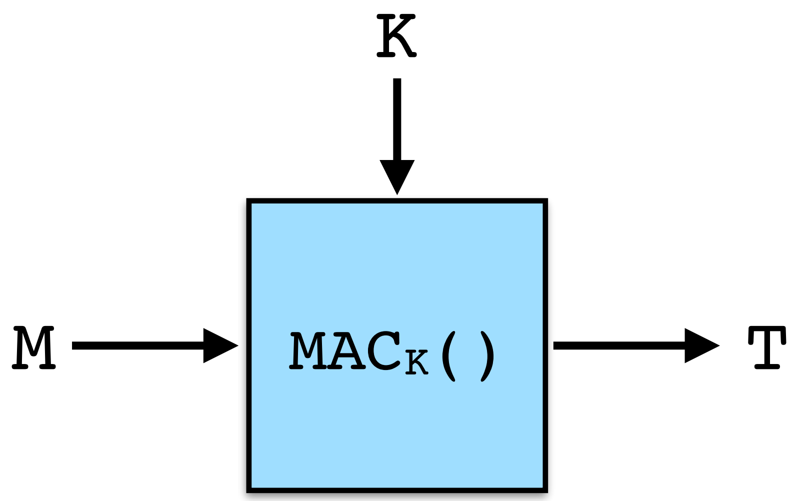
Some security goals:

- 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!

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)

MACs from Hash Functions

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

Common construction: MAC(K, M) = H(K || M)

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

Upshot: Use HMAC and avoid various issues.

Later: Hash functions and certificates

The End