# WEEK 5

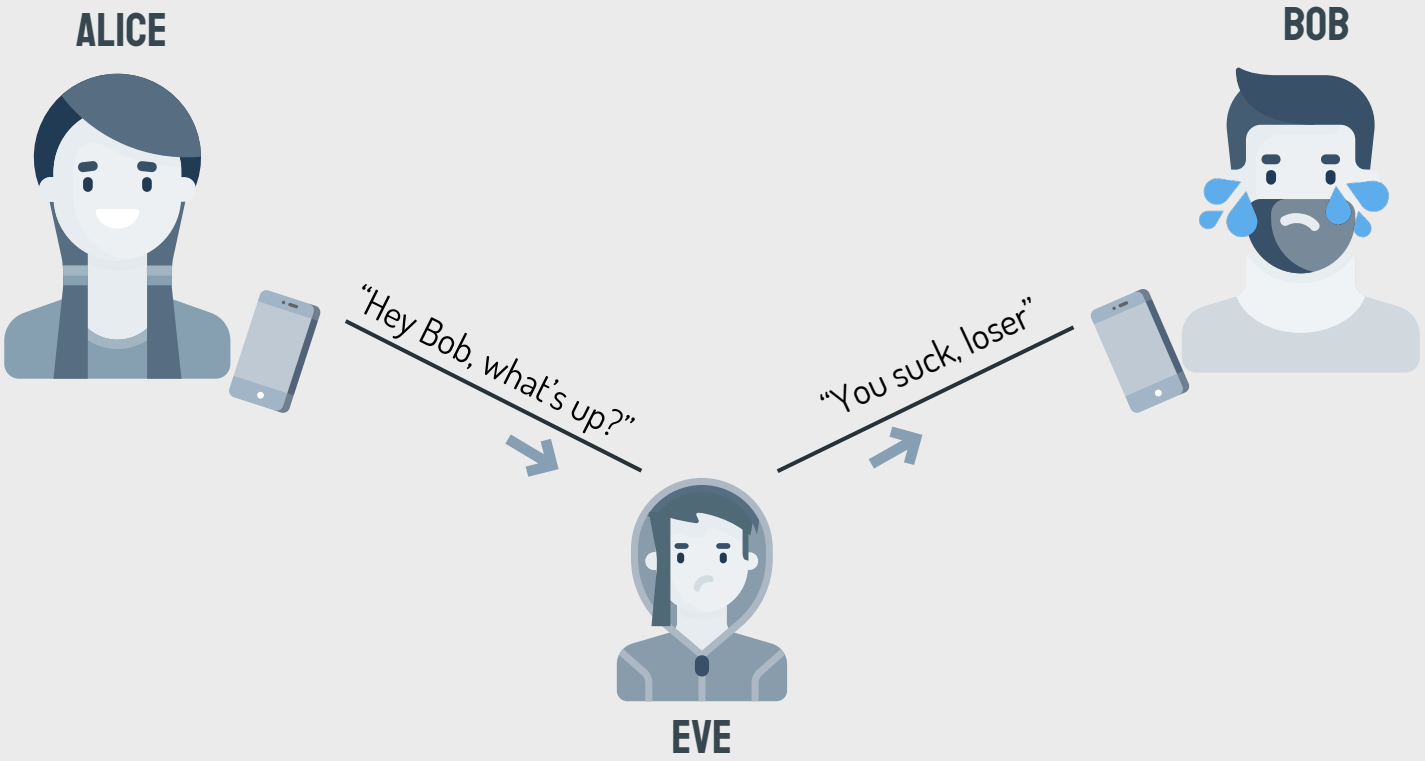
# AUTHENTICATING DATA

SE 4472 - Information Security

Western Engineering

WHISPER LAB

# HOW IMPORTANT IS MESSAGE INTEGRITY, REALLY?



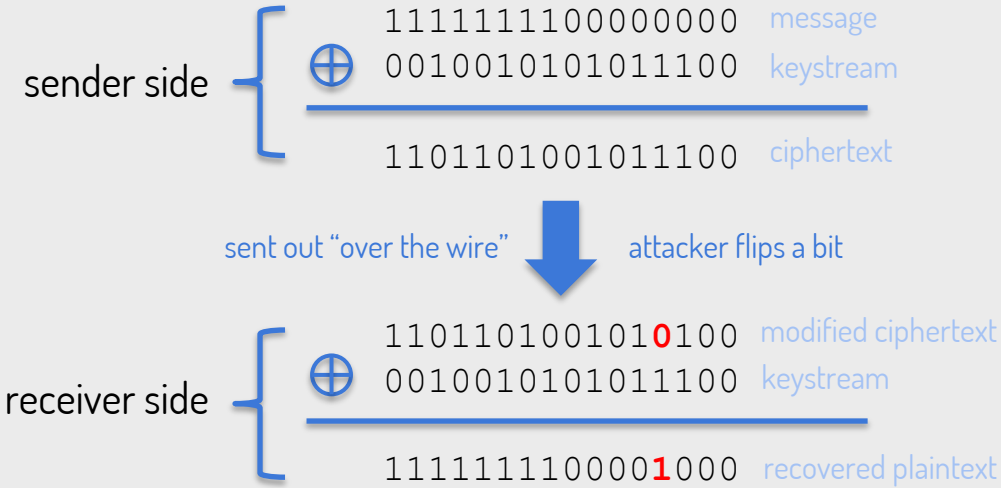
# HOW IMPORTANT IS MESSAGE INTEGRITY, REALLY?

## IT'S ACTUALLY VERY IMPORTANT

IT'S SO IMPORTANT THAT WITHOUT INTEGRITY, YOU CANNOT GUARANTEE CONFIDENTIALITY

# EFFECTS OF MESSAGE MODIFICATION

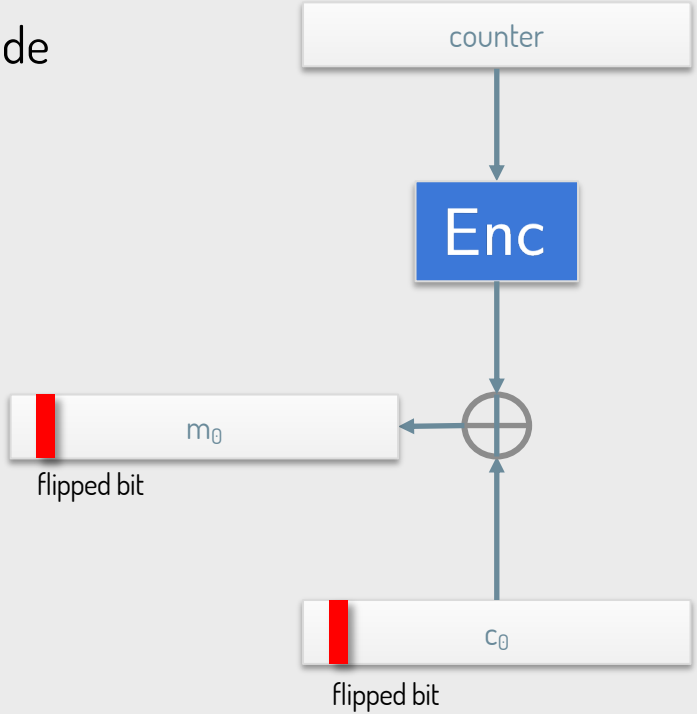
What happens if the attacker modifies the ciphertext in the one-time pad?



# EFFECTS OF MESSAGE MODIFICATION

Block cipher in CTR-mode

**Result**: Flipping bit of ciphertext flips bit of plaintext



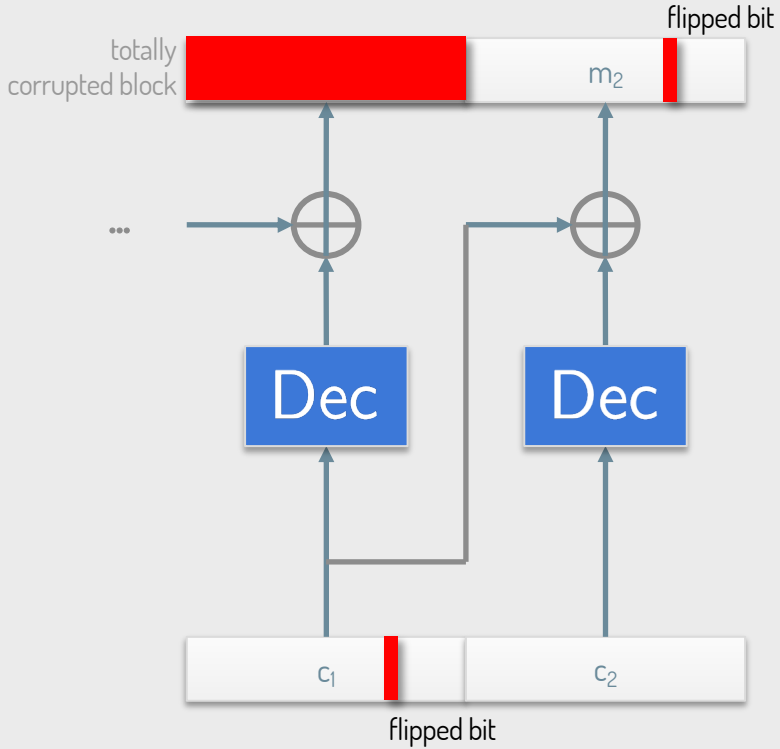
# EFFECTS OF MESSAGE MODIFICATION

Block cipher in CBC-mode

**Result:** Flipping bit of ciphertext totally corrupts current plaintext block (avalanche effect)

BUT

Flips bit in *next* block!



# HOW IMPORTANT IS MESSAGE INTEGRITY, REALLY?

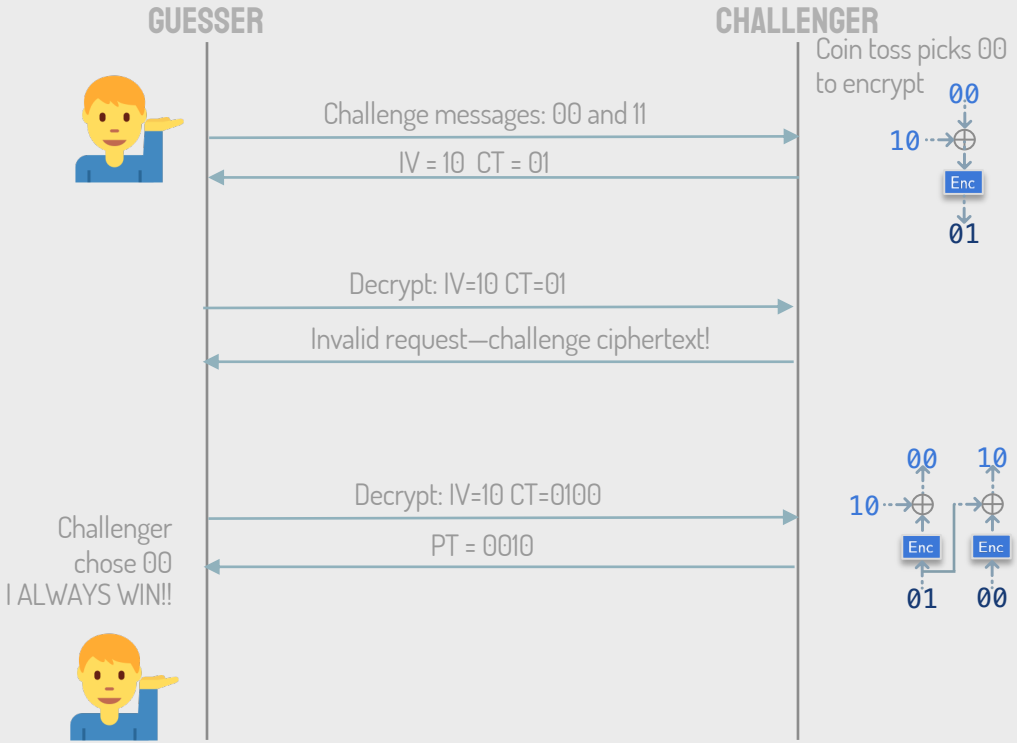
## FLIPPING BIT OF CIPHERTEXT FLIPS BIT OF PLAINTEXT

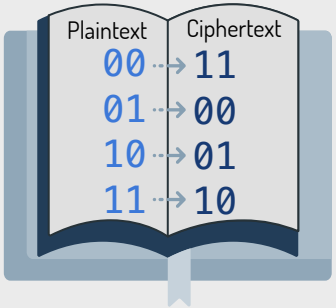
A MAN-IN-THE-MIDDLE CAN MODIFY PLAINTEXT IN A LINEAR WAY WITHOUT KNOWING THE KEY

# OK, SO WHAT?

THIS IS JUST AN ERROR-CORRECTION PROBLEM, RIGHT? WON'T BOB JUST NOTICE IF THE PLAINTEXT COMES OUT FUNNY-LOOKING?

# ADAPTIVE CHOSEN CIPHERTEXT ATTACK



# HOW IMPORTANT IS MESSAGE INTEGRITY, REALLY?

## OK, SO YOU CAN WIN THE CCA2 GAME IF THERE'S NO INTEGRITY

BUT HOW LIKELY IS THIS TO HAPPEN IN PRACTICE? IT'S NOT LIKE BOB IS GOING TO JUST START DECRYPTING THINGS FOR STRANGERS, RIGHT?

## RIGHT??

# BLOCK CIPHER PADDING

**Question**: You want to encrypt a message that is not an even multiple of the block length. What do you do?

**Example**: you have a 11-byte message and you want to encrypt with AES, which has a 16-byte block.

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Plaintext: | H | E | L | L | O | | W | O | R | L | D | | | | |
| UTF-8 ASCII bytes: | 68 | 65 | 6c | 6c | 6f | 20 | 77 | 6f | 72 | 6c | 64 | | | | |
| 16-byte AES plaintext block: | 68 | 65 | 6c | 6c | 6f | 20 | 77 | 6f | 72 | 6c | 64 | ?? | ?? | ?? | ?? | ?? |

what do we put here??

# BLOCK CIPHER PADDING

- Use padding: a bunch of extra bytes to fill up the block
- PKCS #7 is one way to do it: Pad with N bytes of 0xN
- Always pad, therefore unambiguous: every plaintext gets padded, even if plaintext is a multiple of the block length (add an entire block of padding!)

Need 5 bytes of padding?

11-byte plaintext block: `68 65 6c 6c 6f 20 77 6f 72 6c 64 ?? ?? ?? ?? ??`

Padded plaintext block: `68 65 6c 6c 6f 20 77 6f 72 6c 64 05 05 05 05 05`

Put 5 bytes of 0x05

# BLOCK CIPHER PADDING

WHAT HAPPENS IF THE PADDING IS WRONG? WHAT SHOULD YOU DO?

# A PADDING ORACLE

- Alice and Bob use PKCS7 padding. Suppose Eve injects a random ciphertext. Bob decrypts and gets:

```
5D 37 64 73 43 6A 85 31 40 BC 37 27 88 37 74 4E
```

- Uh oh! The padding's wrong. What does Bob do? Return an error?

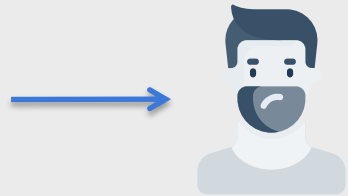
- Bob is going to have to somehow behave differently when the padding is incorrect. May take different amounts of time to handle error vs. non error conditions

- Eve can sit back and observe if Bob changes his response based on her modification and use this information to her advantage!

# A PADDING ORACLE

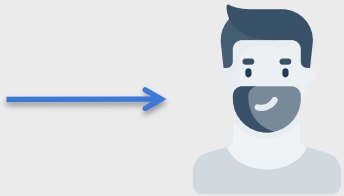
## WHETHER HE MEANS TO OR NOT, BOB IS A PADDING ORACLE

```
5D 37 64 73
43 6A 85 31
40 BC 37 27
88 37 74 4E
```



Plaintext has invalid padding

```
F8 20 35 A4
03 CA 55 41
49 19 97 07
33 47 8D 50
```



Plaintext has valid padding

# A DECRYPTION ORACLE

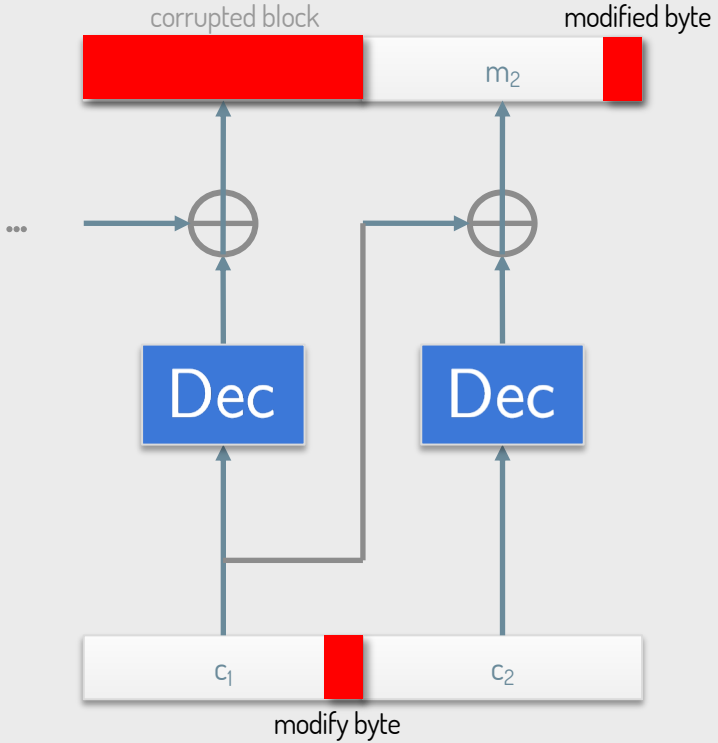
## PROBLEM:

PADDING ORACLES CAN BE TURNED INTO DECRYPTION ORACLES

# A DECRYPTION ORACLE

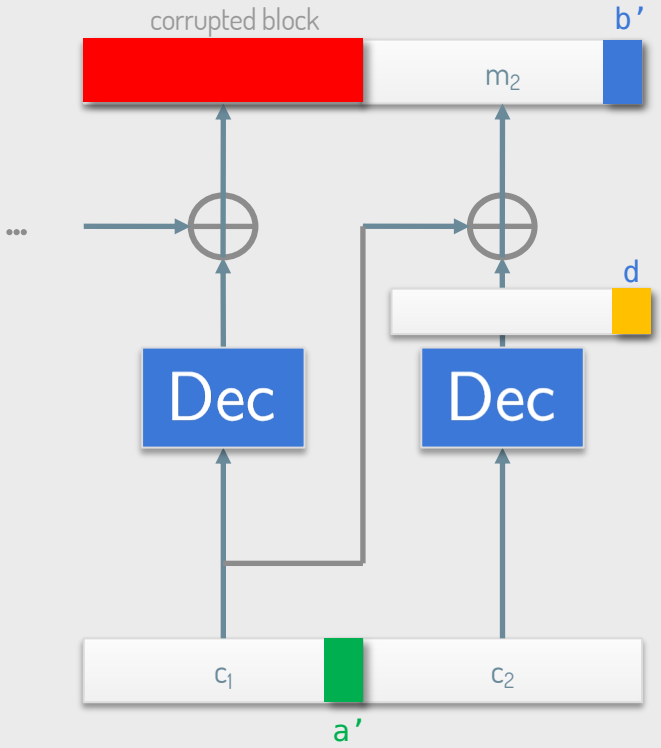
**Suppose:** Eve modifies the last byte of the first block of a ciphertext

If Bob acts like a padding oracle, Eve can exploit his reactions to recover the plaintext



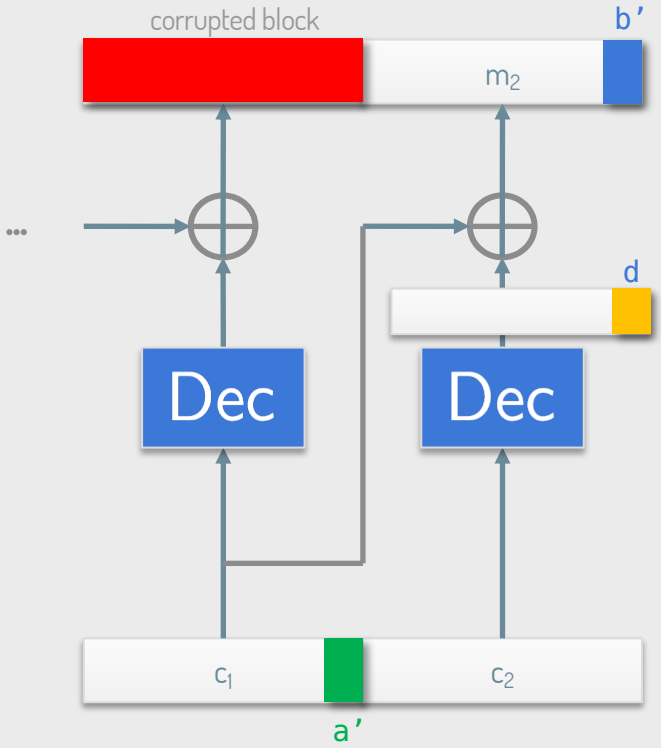
# A DECRYPTION ORACLE

1. Let the last byte of ciphertext block $c_1$ block be called `a`
2. Let the last byte of the plaintext block $m_2$ be called `b = a XOR d`
3. Eve makes a guess `g`. Eve replaces `a` with:
   `a' = g XOR 0x01`
4. Bob now decrypts and gets:
   `b' = d XOR a'`
   `= d XOR g XOR 0x01`
5. There are two outcomes:
   - Guess `g` was correct, i.e., `g=d`. Then `g,d` cancel out, and we have `b'= 0x01`. This is a valid padding no matter what the rest of $m_2$ is. We know `g=d` and therefore `b=a XOR g`
   - Guess g was incorrect. Then `b' != 0x01`, which is invalid padding (most of the time, depending on rest of plaintext)



# A DECRYPTION ORACLE

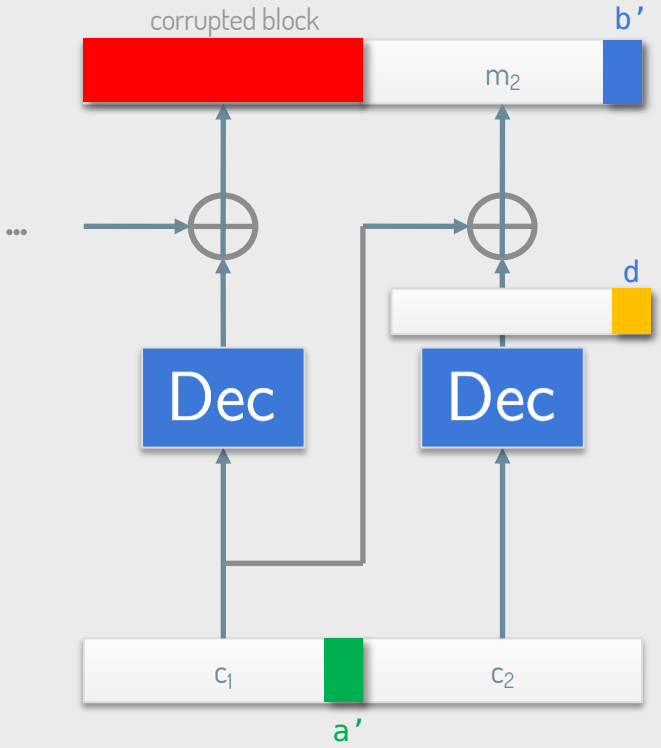
- Eve replays the ciphertext for each possible guess, i.e., submits a ' for all g = 0...255
- If only one guess succeeded, she has successfully decrypted the last byte
- What if more than one guess succeeds, e.g., suppose:
  b = ... 03 03 03
  Then guesses for the last byte:
  g = 03 and g = 01 would both produce valid pads
- Eve can try to "decrypt" the second last byte to figure out which case it was




**Exercise:**
How could Eve extend this attack to decrypt all the bytes of $m_2$?

# PADDING ORACLE ATTACKS

- For each byte of ciphertext, Eve can recover the associated plaintext typically in 255 queries
- AES: 16 byte block times 255 queries = ~4000 queries per block.
- Real PO implementations can decrypt plaintext in a few seconds
- Attack can be used, e.g., to recover your authentication tokens/cookies to hijack your session



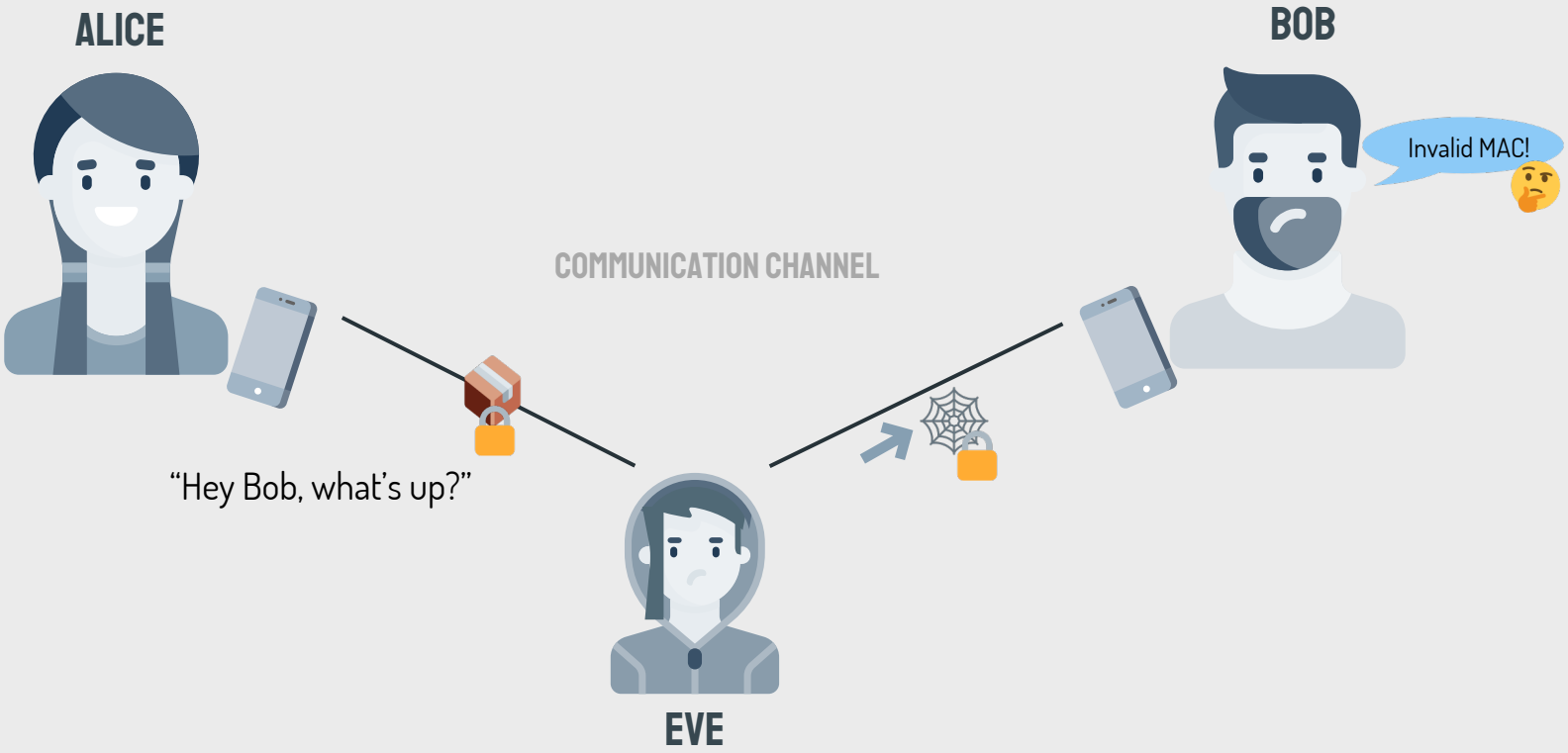
# HOW TO STOP PADDING ORACLES

WHAT IF DECRYPTION ONLY WORKED IF THE CIPHERTEXT WAS VALID?

AND WHAT IF IT WAS HARD TO DECIDE IF A CIPHERTEXT WAS VALID UNLESS YOU HAD A KEY?

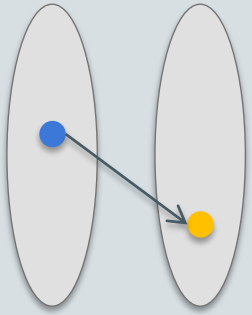
# MESSAGE AUTHENTICATION CODES

# MESSAGE AUTHENTICATION

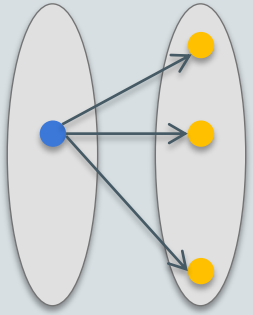


# MESSAGE AUTHENTICATION

**IND-EAV**: Every plaintext maps to a unique ciphertext

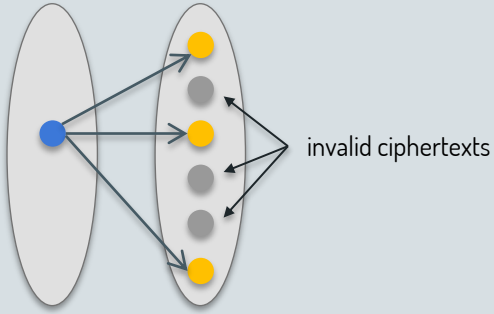


**IND-CPA**: Every plaintext maps to one of many possible plaintexts



- **Plaintexts**
- **Ciphertexts**

**IND-CCA**: Same as IND-CPA except most ciphertexts are invalid, and only a key holder can decide



# MESSAGE AUTHENTICATION CODES (MAC)

## KEYGEN

Accepts a security parameter k, outputs a random k-bit key

Gen : $k \rightarrow \{0,1\}^k$

## SIGN

Accepts an arbitrary length message and k-bit key and outputs an l-bit MAC tag

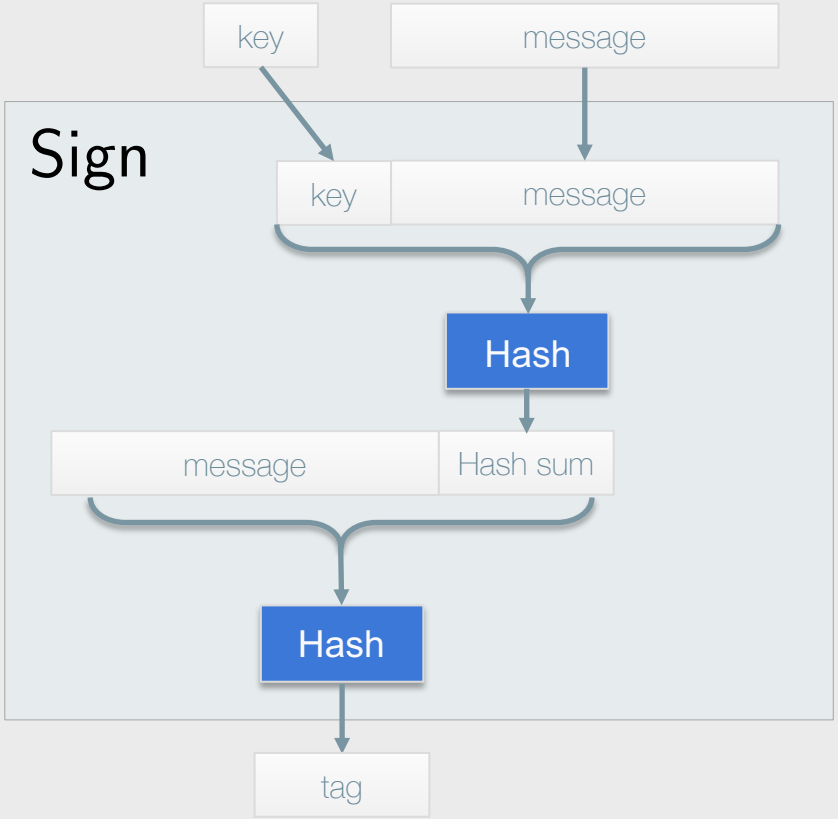
Sign : $\{0,1\}^* \times \{0,1\}^k \rightarrow \{0,1\}^\ell$

## VERIFY

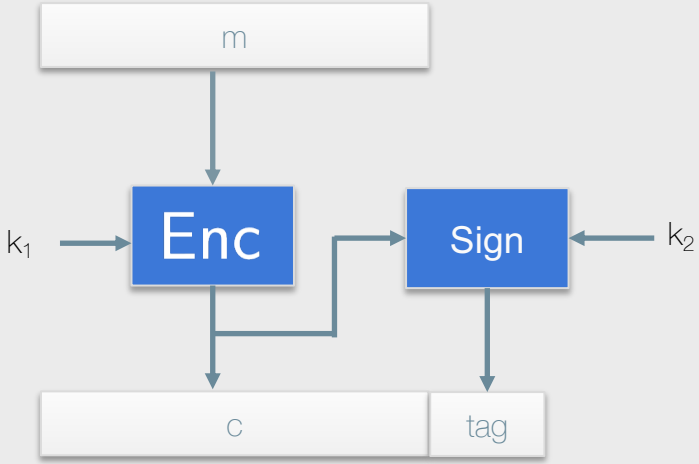
Accepts an arbitrary length message, a k-bit key, an l-bit MAC tag, and outputs a single (Boolean) bit

Verify : $\{0,1\}^* \times \{0,1\}^k \times \{0,1\}^\ell \rightarrow \{0,1\}$

# HASH-BASED MESSAGE AUTHENTICATION CODES (HMAC)



# MESSAGE AUTHENTICATION CODES (MAC)

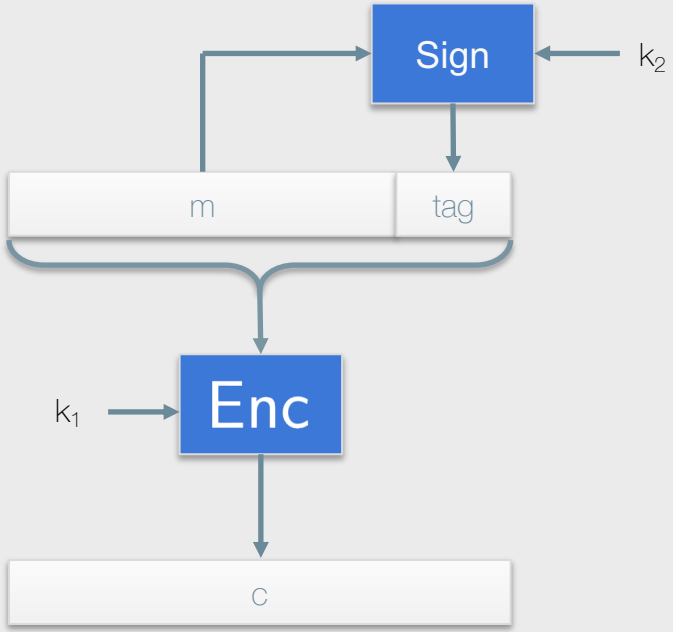



**Observe**: Encryption and MAC signing keys are different

The must be independently generated and kept separate, otherwise attacks exist

## ENCRYPT-THEN-MAC CONFIGURATION

# MESSAGE AUTHENTICATION CODES (MAC)

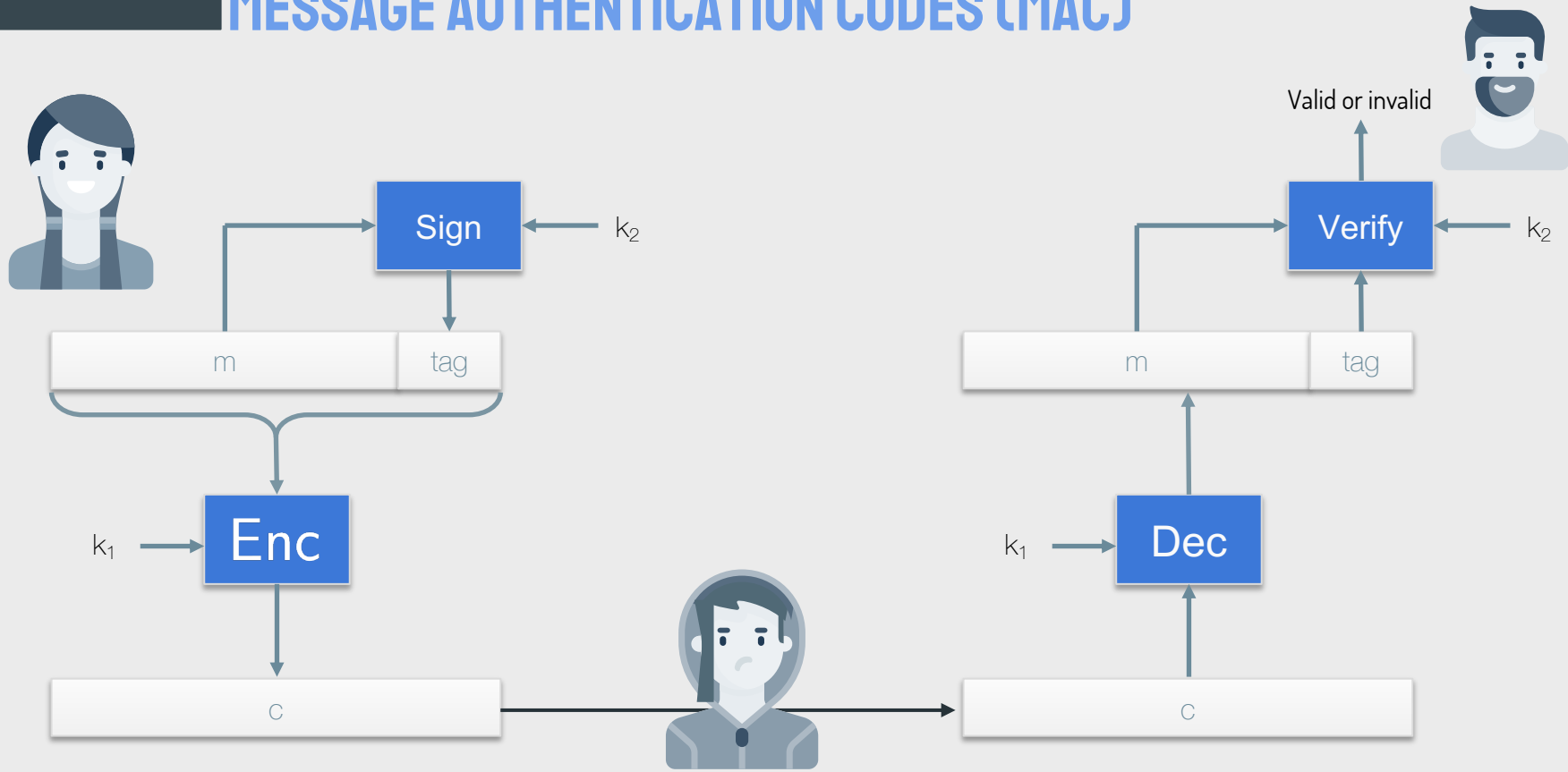



**Observe**: Encryption and MAC signing keys are different

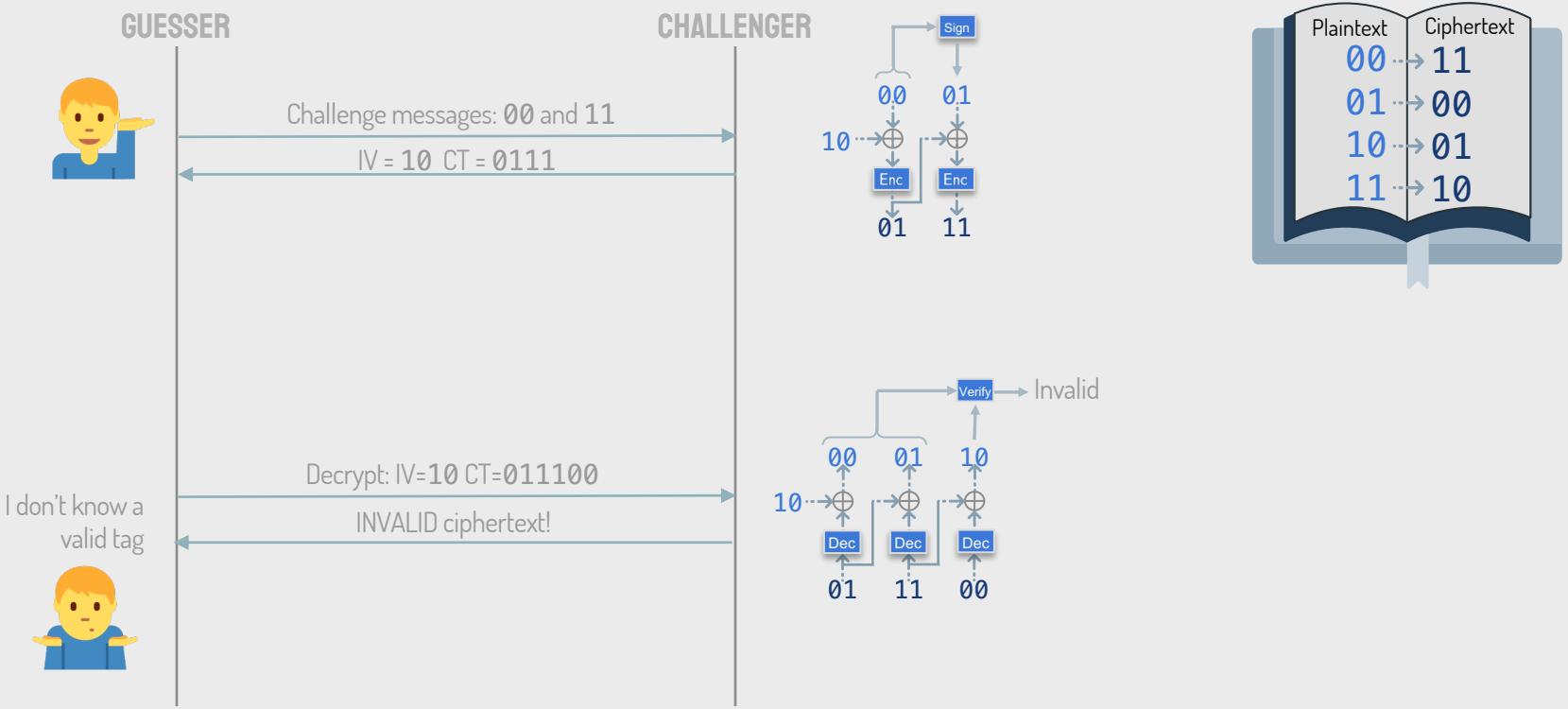
The must be independently generated and kept separate, otherwise attacks exist

## MAC-THEN-ENCRYPT CONFIGURATION

# MESSAGE AUTHENTICATION CODES (MAC)



# ADAPTIVE CHOSEN CIPHERTEXT ATTACK



# HOW MACS DEFEAT PADDING ORACLES

- Bob accepts or rejects a message based on its MAC, not on the plaintext.
- Bob does not even look at the plaintext unless MAC is valid
- MUCH harder for Eve to produce valid MAC (e.g., $1/2^{128}$) than it is to produce valid padding byte (i.e., 1/256) in the PO attack

# AUTHENTICATED ENCRYPTION

# AUTHENTICATED ENCRYPTION

Keygen (security parameter):

returns encryption key $k_e$ and MAC key $k_m$

Encrypt (plaintext, $k_e$, $k_m$):

returns ciphertext c, MAC tag t, and IV

Decrypt(c, t, IV, $k_e$, $k_m$):

returns plaintext if t is a valid tag for ciphertext c,
otherwise error if t is invalid

# AUTHENTICATED ENCRYPTION

Protects developers by combining a cipher, cipher mode of operation, MAC, MAC configuration, and exception handling behind all behind one API

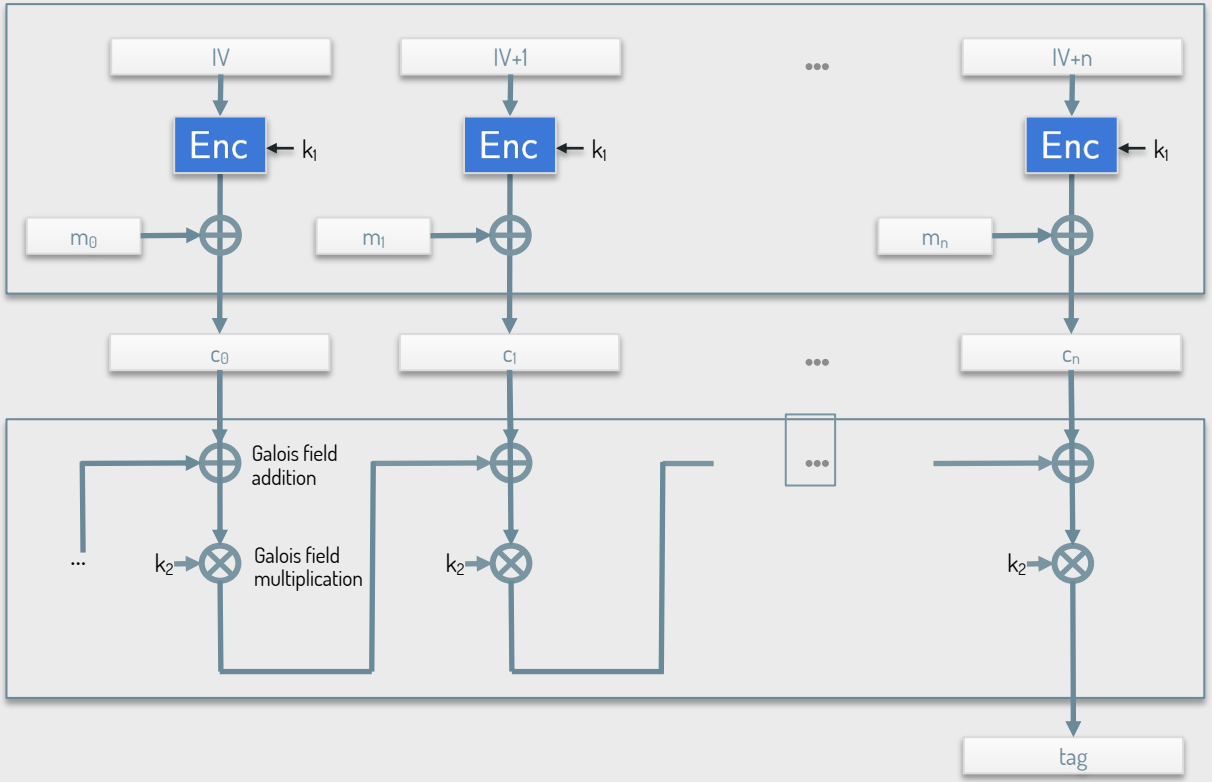
Example AES-GCM:

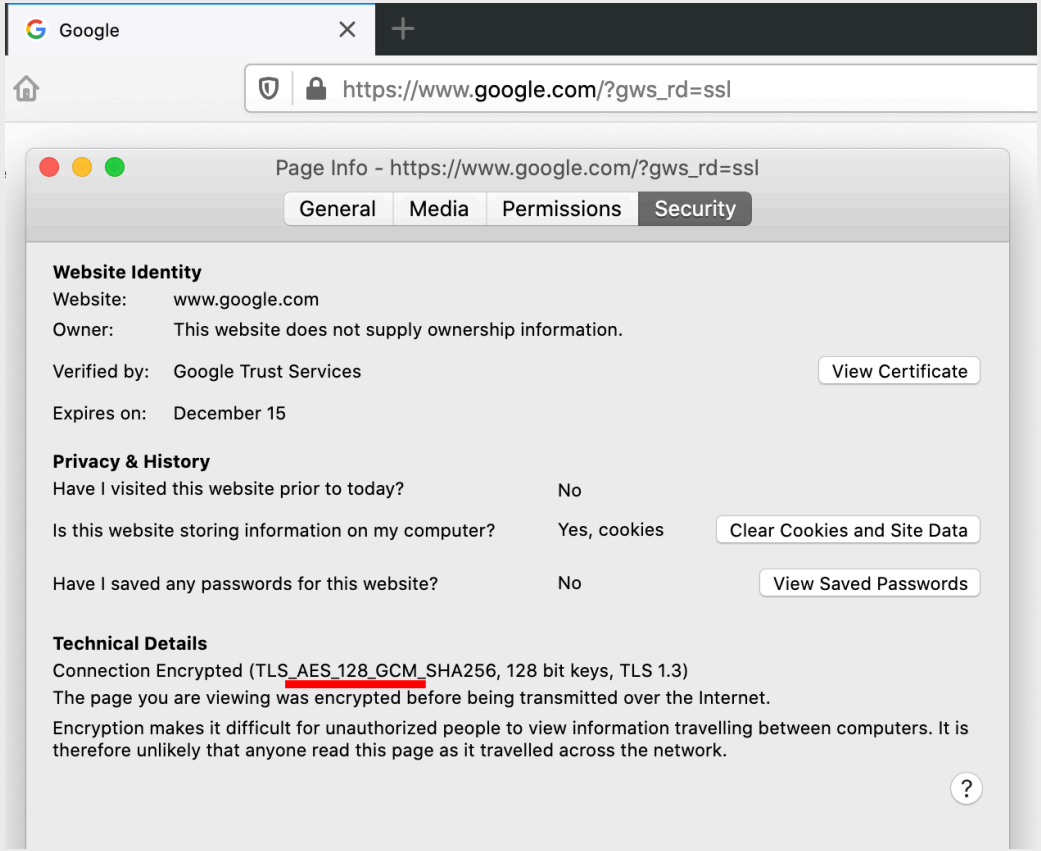
- AES block cipher
- CTR mode
- G-HASH MAC
- Encrypt-then-MAC configuration

# AES GALOIS COUNTER MODE (AES-GCM)

Encryption (AES in CTR mode)

MAC (Galois hash, Encrypt-then-MAC




Google
https://www.google.com/?gws_rd=ssl
Page Info - https://www.google.com/?gws_rd=ssl
General
Media
Permissions
Security
Website Identity
Website: www.google.com
Owner: This website does not supply ownership information.
Verified by: Google Trust Services
View Certificate
Expires on: December 15
Privacy & History
Have I visited this website prior to today? No
Is this website storing information on my computer? Yes, cookies
Clear Cookies and Site Data
Have I saved any passwords for this website? No
View Saved Passwords
Technical Details
Connection Encrypted (TLS_AES_128_GCM_SHA256, 128 bit keys, TLS 1.3)
The page you are viewing was encrypted before being transmitted over the Internet.
Encryption makes it difficult for unauthorized people to view information travelling between computers. It is therefore unlikely that anyone read this page as it travelled across the network.
?