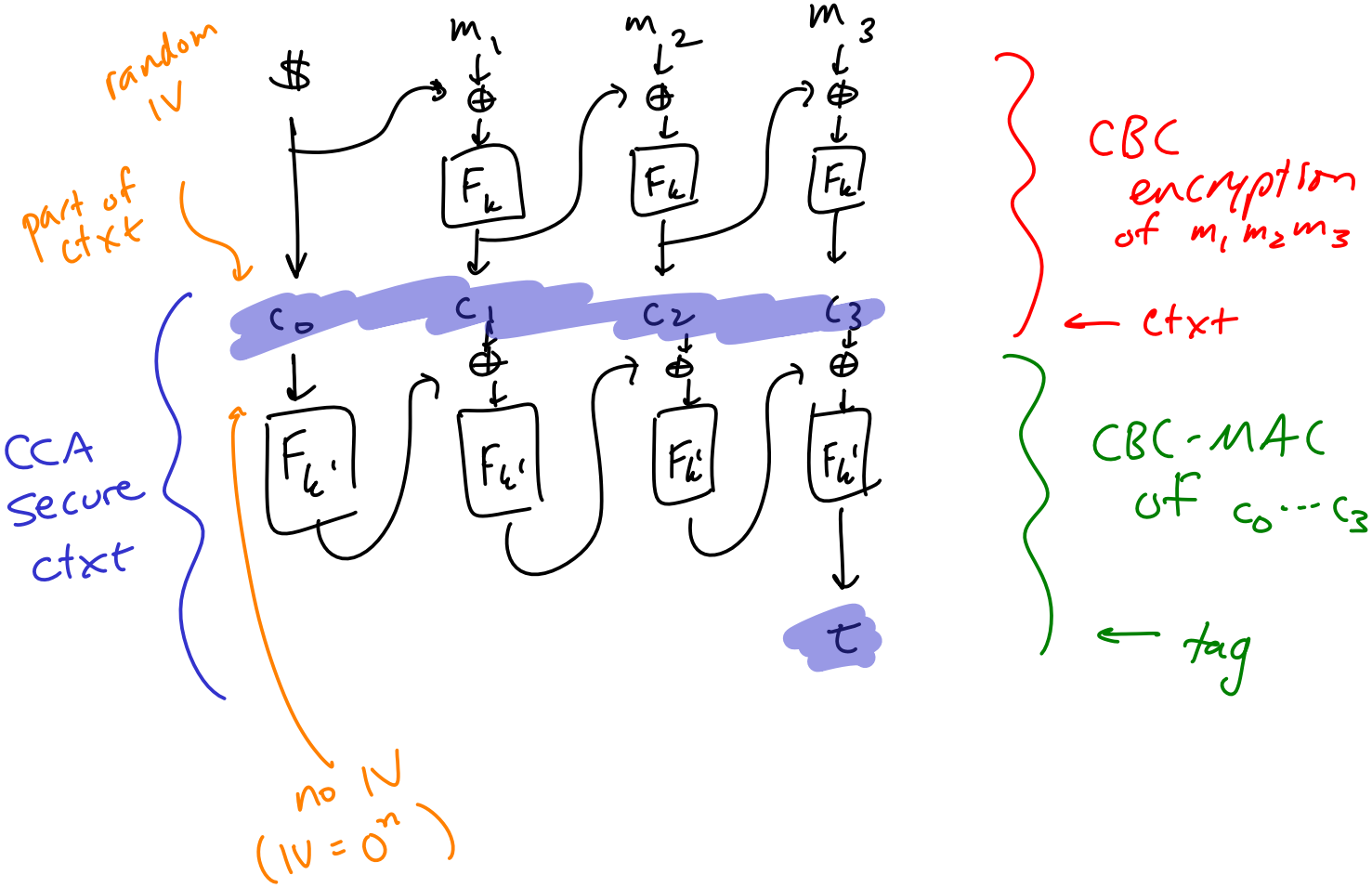
MACs

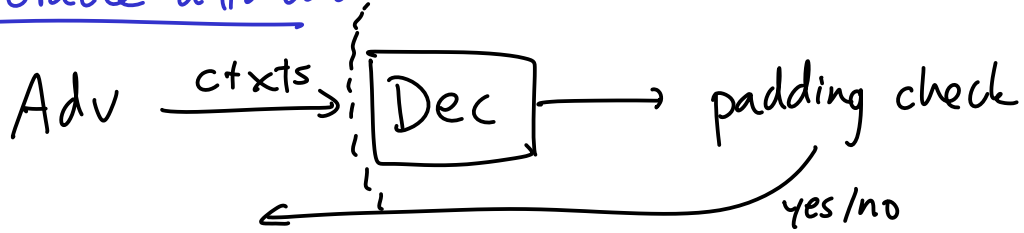
Exam Friday, review Wednesday, proj guidelines soon

Encrypt-then-MAC

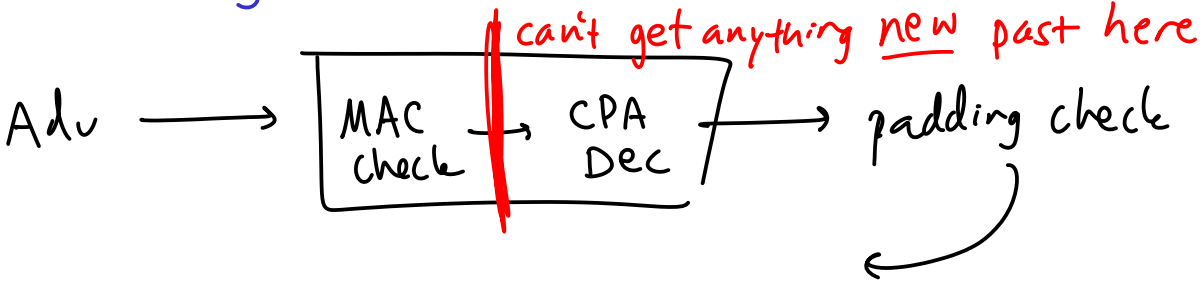
↖ CBC mode ↖ CBC-MAC ??



Padding oracle attacks



when using Enc-then-MAC scheme:



Enc-then-MAC proof: (prove CCA-secure)

IDEA: Adv can ask for things to be decrypted

Claim: Adv doesn't get any info from DEC subroutine

→ if Adv asks for DEC(something generated by library)

library will return error (part of CCA definition)

→ if Adv asks for DEC(something new)

library returns error unless Adv happened to find a forgery of the MAC

So if DEC subroutine is useless, things just collapse to CPA security

Big MAC Attack:

```
k ← MAC.KeyGen

GETMAC(m):
    ret MAC(k,m)

VER(m,t):
    ret MAC(k,m) ?= t
```

$\overset{?}{\approx}$

```
k ← MAC.KeyGen
T = ∅

GETMAC(m):
    t = MAC(k,m)
    add (m,t) to T
    return t

VER(m,t):
    ret (m,t) ∈? T
```

BADMAC: F = PRP

Idea: use block cipher somehow even when m is long

$$MAC(k, m_1 m_2 \cdots m_n) = F(k, m_1 \oplus m_2 \oplus \cdots \oplus m_n)$$

Attack: (Idea)

1. Ask for MAC's of some msgs,
2. use the results to determine MAC of some other msg.
3. call VER on new MAC
   - ↳ in left library, VER says yes since it is a valid MAC
   - ↳ in right library, VER says no since it's a "new" thing ($\notin T$)

Obs: permuting the order of $m_1 \cdots m_n$ blocks doesn't change the MAC

Attack (more formal):

$t = \text{GETMAC}(0^\lambda 1^\lambda)$

return $\text{VER}(1^\lambda 0^\lambda, t)$

$0^\lambda 1^\lambda$ and $1^\lambda 0^\lambda$ have same MAC

Another:

$$MAC(k, m_1 \cdots m_n) = F(k, 1 \| m_1) \oplus F(k, 2 \| m_2) \oplus \cdots \oplus F(k, n \| m_n)$$

Idea: Ask for 3 MACs

$m_1, m_2 \longrightarrow F(k, 1\|m_1) \oplus F(k, 2\|m_2)$

$m_1', m_2 \longrightarrow F(k, 1\|m_1') \oplus F(k, 2\|m_2)$

$m_1, m_2' \longrightarrow F(k, 1\|m_1) \oplus F(k, 2\|m_2')$

XOR all 3 MACs = valid MAC of m_1', m_2':

$m_1', m_2' \longrightarrow F(k, 1\|m_1') \oplus F(k, 2\|m_2')$

can compute this MAC w/o asking library

$\Rightarrow$ forgery!!