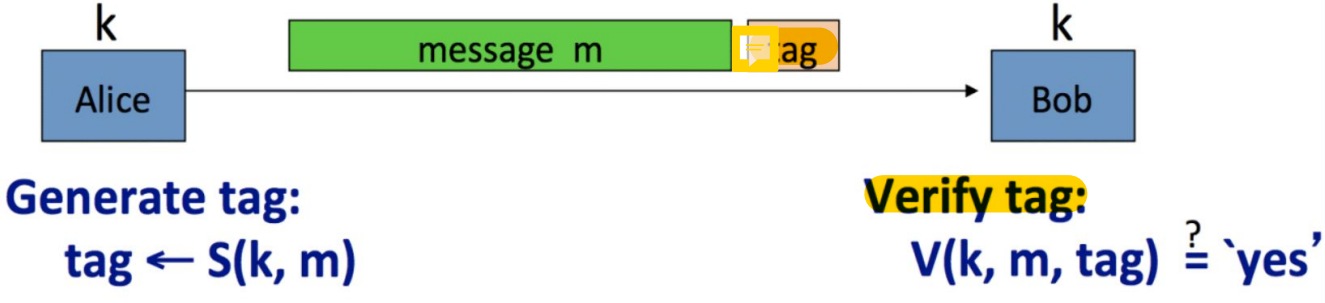
Tutorial 6: Message Integrity

CS3235 - Spring 2022

Message Integrity

Message Authentication Code (MAC)



Pair of Algorithms (S, V)

1. Signing S

   S: K x $M \rightarrow T$, secret key k and message m are inputs, tag t is output

2. Verification V

   V: K x M x $T \rightarrow \{0,1\}$, secret key k, message m and tag t are inputs

   $V(k,m,t) = 1$ iff $S(k,m) == t$

   $= 0$ otherwise

Message Authentication Code (MAC)

Ensures

- Authenticity:

  Message comes from the person with the shared key

- Integrity:

  No tampering attacks, attacker cannot modify the message

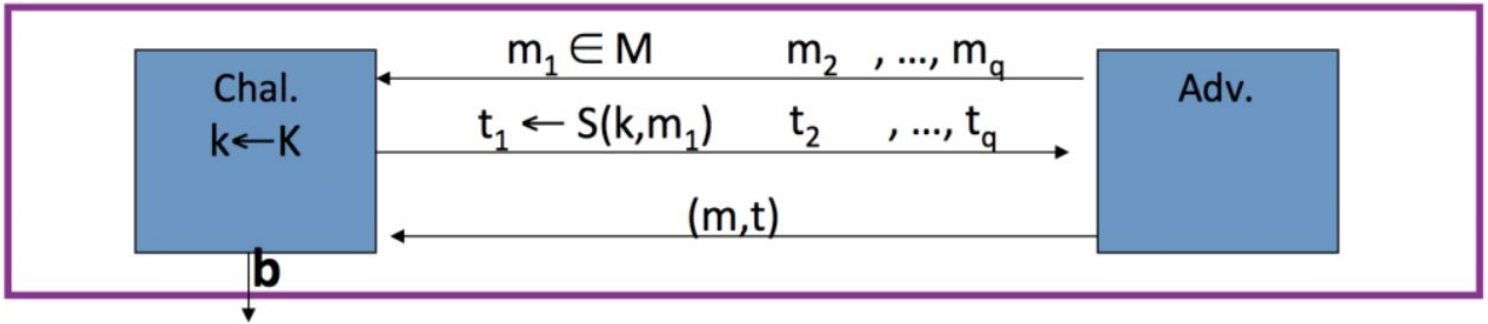
Existential Forgery

Given a set of messages m1, m2, …, mq, and tags t1, t2, …, tq

Can an attacker create a tag t’ for some message m’?

(m’, t’) are not in the queried set

Chosen Message Attack



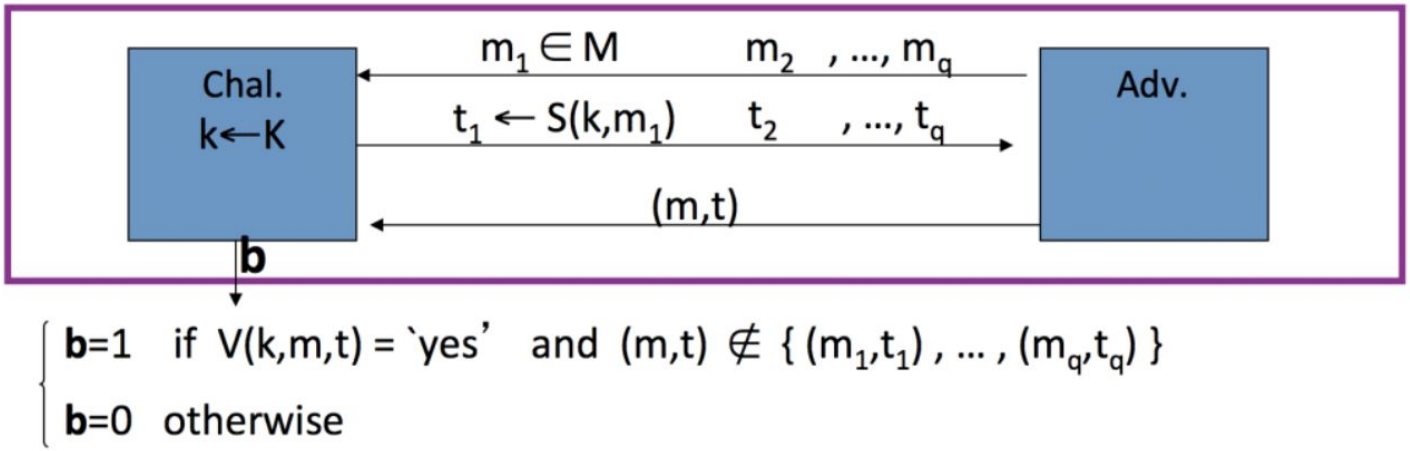
Attacker request tags for messages of his choice

$m_1, m_2, \ldots, m_q$ gets $t_1, t_2, \ldots, t_q$

Attacker sends a pair (m,t) where $m \notin m_1, m_2, \ldots, m_q$

Challenger verifies

Secure MAC



The MAC is secure if for all efficient adversaries, the probability that the challenger outputs 1 is negligible.

Pseudo Random Function (PRF)

- $F : K \times X \rightarrow Y$ PRF is not a 1-1 function

  F is a PRF defined over (K,X,Y) if there is an efficient algorithm to evaluate $F(k,x)$

- Secure PRF

  Let Funs$[X,Y]$ be the set of all functions from X to Y

  $S_F = \{F(k,x)$ such that $k \in K, x \in X\} \subseteq$ Funs$[X,Y]$

  is a secure PRF if it is indistinguishable from a random function

MAC from Secure PRF

$F : K \times X \rightarrow Y$ is a secure PRF

We define MAC as

- $S_F(k,m) = F(k,m)$
- $V_F(k,m,t) = 1$ if $t = F(k,m)$ else 0

Example MAC: $F : K \times X \rightarrow Y$ with $Y = \{0,1\}^{16}$

Is it a secure MAC?

MAC from Secure PRF

(Theorem) If $F : K \times X \rightarrow Y$ is a secure PRF and $|Y|$ is large, then (S_F, V_F) is a secure MAC.

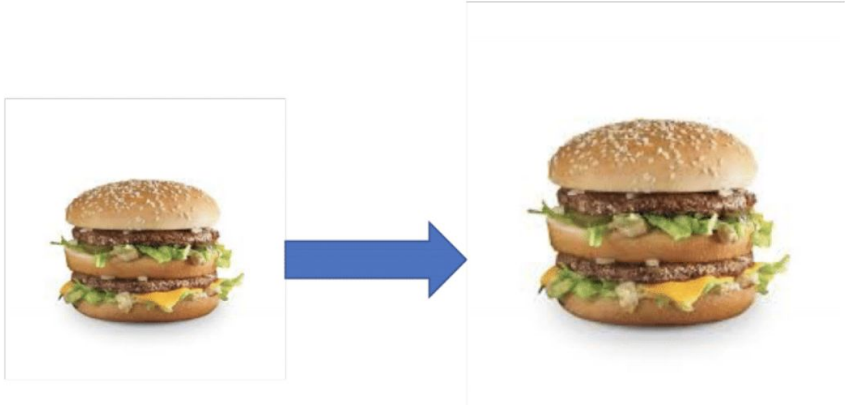
Eg: $|Y|$ is large, 2^{80}

Then can use AES-128.

Big MAC from Small MAC

- AES-128 takes input as 16 byte messages
- We want to compute tags for large messages eg: files

How to go from small PRF to large PRF?



Naive Construction

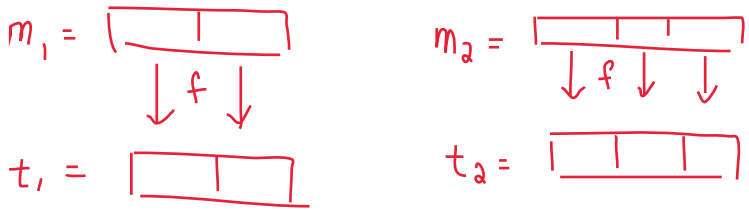
- Break the message into blocks of 16 bytes
- $S(k,m) = F(k,m_1) \,||\, F(k,m_2) \,||\, F(k,m_3) \,||\, \ldots. \,||\, F(k,m_q)$
- $V(k,m,t) = V(k,m_1,t_1) \wedge V(k,m_2,t_2) \wedge V(k,m_3,t_3) \wedge \ldots.. \wedge V(k,m_q,t_q)$

How can adversary misuse?

Length Extension Attack

Given (m_1, t_1), (m_2, t_2),

Can construct a valid tag $S(k, m_1 \,||\, m_2)$ from t_1 and t_2



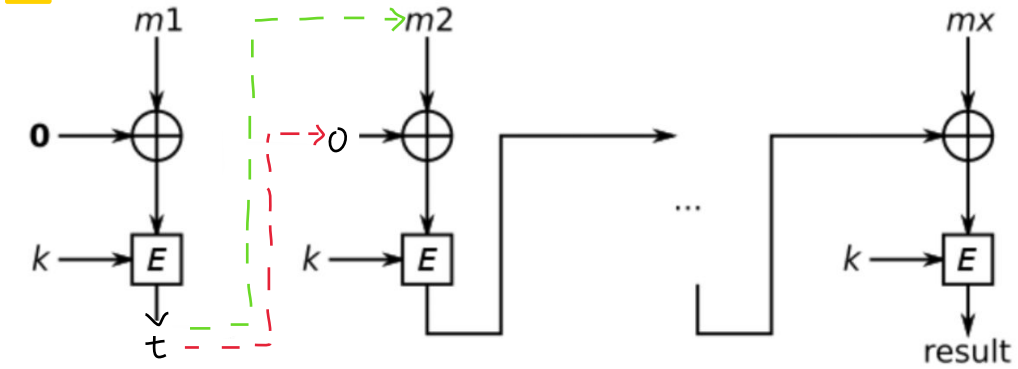
$\therefore \; m' = m_1 \,||\, m_2$

↳ corresponding tag

$= t' = t_1 \,||\, t_2$

Raw CBC-MAC

assuming m_1 is the original message, we want to append our own message m_2 to it



Divide message m into blocks $<m_1, m_2, \ldots, m_x>$

$c_i = E(k, m_i, c_{i-1})$

Is RAW CBC-MAC secure from length extension attack?

Raw CBC-MAC

Length Extension Attack

- Adversary gets pairs (m,t) , (m’ , t’)
- Let m’’ = m || m’
- Let m_1’ = first block of m’
- Replace first block of m’ with $m_1' \oplus t$

$m'' = m_1 || m_2 || \ldots m_q || m_1' \oplus t || m_2' || \ldots m_r'$

$m'' = m || m_1' \oplus t || m_2' || \ldots m_r'$

Raw CBC-MAC

$E(k,m'') = E(k, m \,||\, m_1' \oplus t \,||\, m_2' || \ldots m_r')$

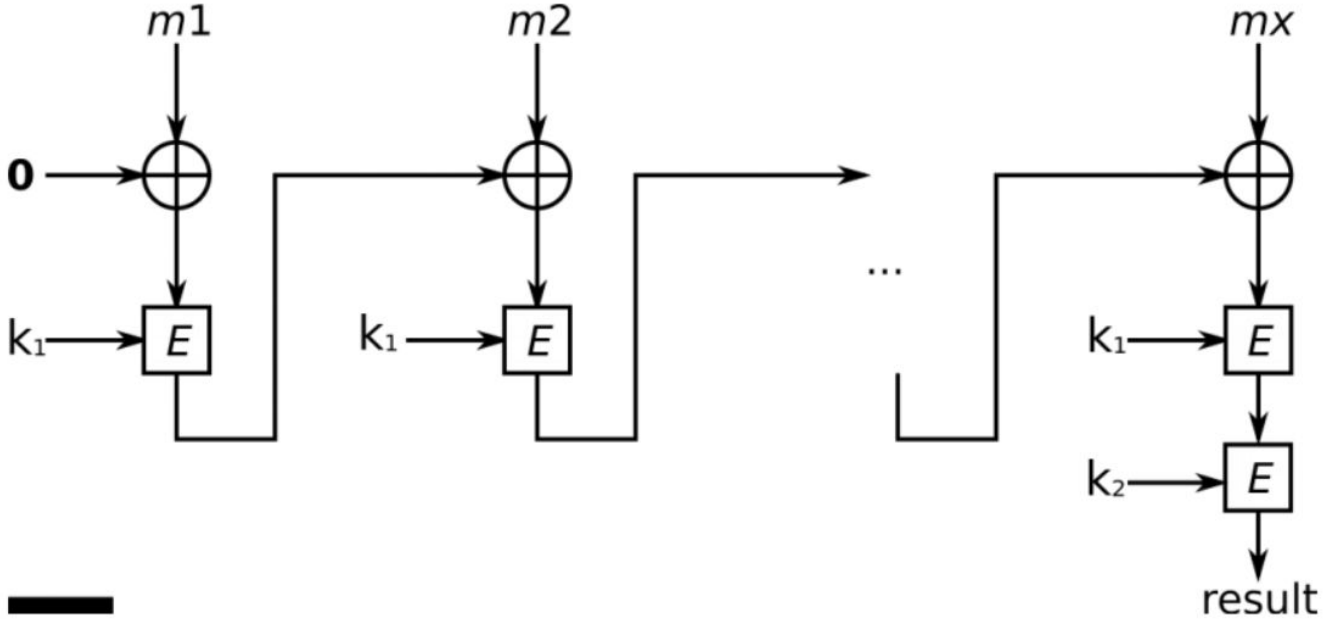
$= E(k, (t \oplus m_1' \oplus t \,||\, m_2' || \ldots m_r'))$

$= E(k, m_1' \,||\, m_2' || \ldots m_r')$

$= E(k, m')$

$= t'$

ECBC-MAC



$t = m_1 \oplus IV \qquad t' = m_1' \oplus IV'$

$\therefore IV' = m_1 \oplus m_1' \oplus IV$

Why is IV set to 0?

- If IV is randomly chosen as say IV_1 and message $m = m_1 \,||\, m_2 \,||\, m_3 \ldots ||\, m_q$
- Attacker gets tag *t* for the message m from the challenger
- First block of ECBC-MAC gives $E_k(IV_1, m_1)$
- Attacker forms a message $m' = m_1' \,||\, m_2 \,||\, m_3 \ldots ||\, m_q$

  Such that $E_k(IV_1, m_1) = E_k(IV_1', m_1')$

  Attacker construct the valid tag *t* for message m' and thus a valid pair (m',*t*)

Digital Signatures

Digital Signature Definition

Alice and Bob do not share a key but each have a pair of (K_{priv}, K_{pub}) keys

- Signing

  $S(K_{priv}, m) = signature$

- Verification

  $V(K_{pub}, m, signature)$ = 1 if signature valid

  = 0 otherwise

Digital Signature - Textbook RSA

Recall RSA encryption scheme has $K_{pub} = (e,N)$, $K_{priv} = (d,N)$

- Signature

  $S(d, N, m) = c = m^d \bmod N$

- Verification

  $V(e, N, c, m) = 1$ if $c^e \bmod N == m \bmod N$
  $= 0$ otherwise

Is textbook RSA secure against existential forgery?

Digital Signature - Improved RSA

Recall RSA encryption scheme has $K_{pub} = (e,N)$, $K_{priv} = (d,N)$

- Signature

  $S(d, N, m, h) = c = h(m)^d \bmod N$

- Verification

  $V(e, N, c, m, h) = 1$ if $c^e \bmod N == h(m) \bmod N$

  $= 0$ otherwise

Authenticated Encryption

Authenticated Encryption Definition

Definition: An **Authenticated Encryption** system (E,D) is a cipher where:

- $E: K \times M \times N \rightarrow C$ (N optional nonce)
- $D: K \times C \times N \rightarrow M \cup \{\perp\}, \perp \notin M$ (bottom is indication that ciphertext rejected)

Security: System must provide:

- Semantic security under a CPA attack, and
- Ciphertext integrity: attacker cannot create new ciphertexts that decrypt correctly

Combining Encryption and Integrity

SSH (Encrypt-and-MAC):

1. Encrypt message, E(ke, m)
2. Append tag = S(ki, m), i.e. compute tag on plaintext
3. Output E(ke, m) || S(ki, m)

SSL (MAC-then-Encrypt):

1. Add tag = S(ki, m) to end of message
2. Encrypt whole thing, E(ke, m || tag)

IPsec (Encrypt-then-MAC):

1. Encrypt message, E(ke, m)
2. Append tag = S(ki, c), i.e. compute tag on ciphertext
3. Output E(ke, m) || S(ki, c)

Encrypt-AND-MAC

NEVER DO THIS!

MAC can leak info about PT - it was never designed for secrecy

Example of why you shouldn’t make your own crypto!

https://tonyarcieri.com/all-the-crypto-code-youve-ever-written-is-probably-broken

MAC-then-Encrypt

Usually secure, but…

- Have to decrypt first to see if message is authentic
- Leads to padding oracle attacks, BEAST, others

https://codeinsecurity.wordpress.com/2013/04/05/quick-crypto-lesson-why-mac-then-encrypt-is-bad/

Doom Principle

“If you have to perform *any* cryptographic operation before verifying the MAC on a message you’ve received, it will *somehow* inevitably lead to doom.”

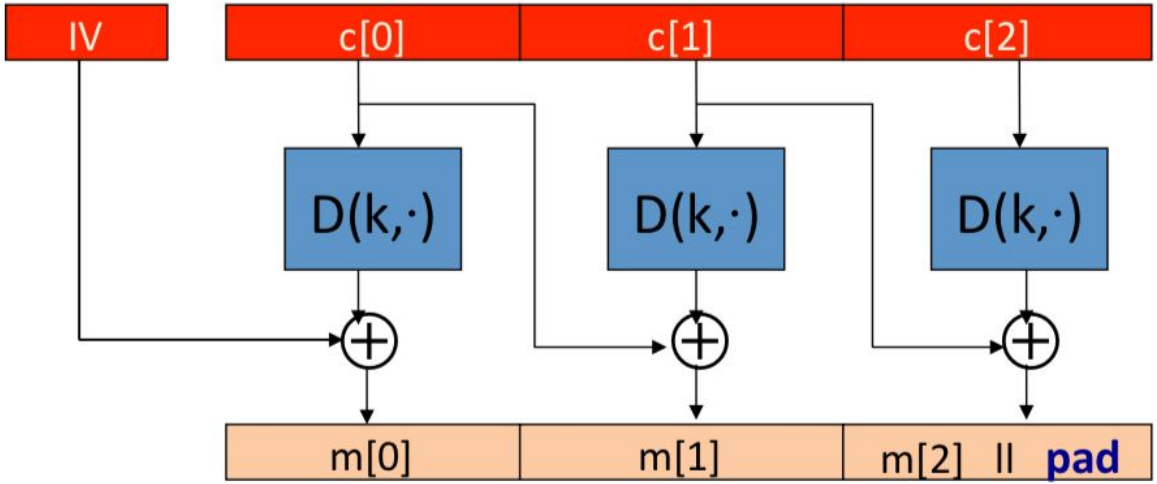
- Moxie Marlinspike,

Founder of Whisper Systems, Signal

Padding Oracle

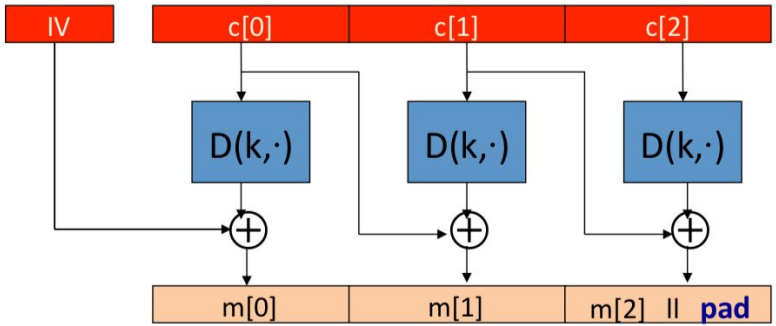
If attacker gets back any different information - including timing! - on a padding error vs any other error, leads to this attack!

Imagine attacker has 3 blocks of CT, and wants to get m[1]



Padding Oracle

1. Let g be a guess for the last byte of m[1]
2. Attacker XOR's into c[0] g and 0x01, $c'[0] = c[0] \oplus g \oplus 0x01$
3. Attacker submits only c'[0] and c[1] to server, i.e. modified first block, original second block, and nothing after that
4. When submitted to server, decryption results in $m'[1] = m[1] \oplus g \oplus 0x01$
5. If pad is invalid, then repeat for a different guess g
6. If the guess is correct, then this produces a valid pad with just 0x01 in the last byte, get invalid MAC message
7. Increase pad to (0x02 0x02) and repeat with a guess for second last byte
8. Do this over all bytes in m[1]



Encrypt-then-MAC

- Ciphertext protected by MAC
- Any tampering gets message immediately rejected
- MAC is applied after encryption, cannot leak any info

When applied correctly yields Authenticated Encryption, but must be implemented correctly!

Direct Construction: Offset Codebook (OCB)

Combining encryption and MAC means doing things twice

Leaves room for errors in application of two schemes

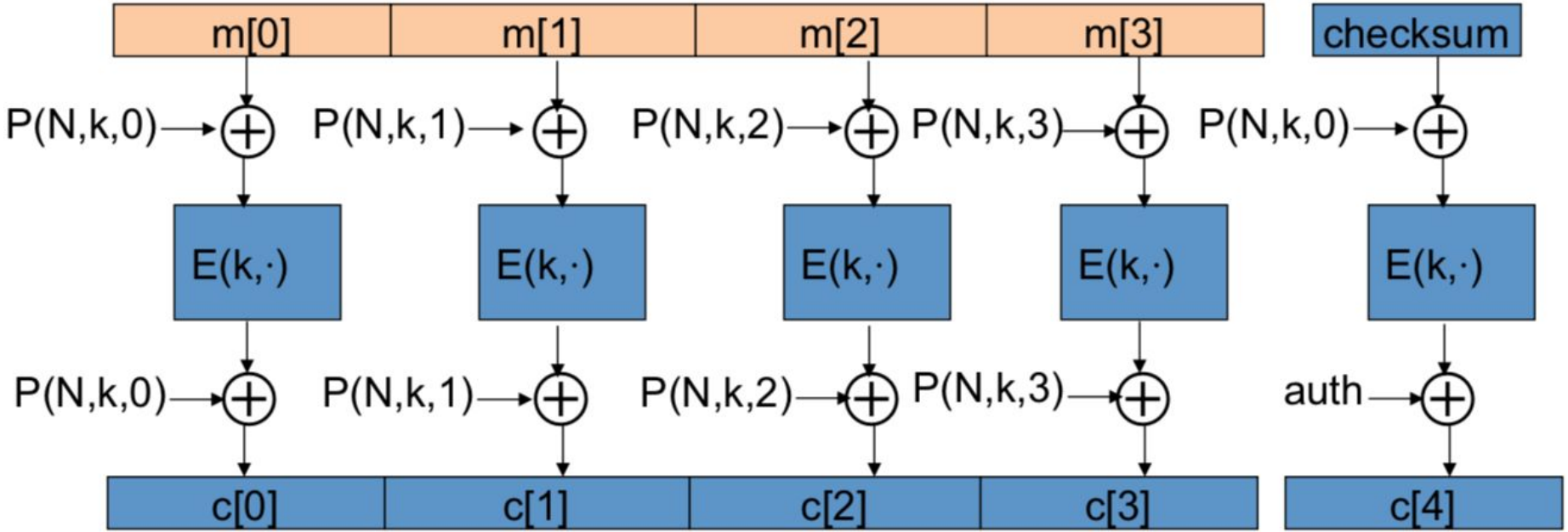
Can we do things just once?

Can we construct Authenticated Encryption directly from a PRP?

Can we compute PRP only once per block?

Answer is yes: Offset Codebook Mode (OCB) Construction

Direct Construction: Offset Codebook (OCB)



Direct Construction: Offset Codebook (OCB)

Advantages:

- Combines Encrypt and MAC into one step
- Execute PRP once per block
- Entirely parallelizable

So why isn’t everyone using it? Patents.

Week 5 Homework Tips

Goals

- Get you to do a *real* attack (or as real as we can get)
- Means dealing with actual systems and limitations, and lots of setup
- **Do not be alarmed by the amount of setup. The problems are actually quite easy once you have things set up correctly.**
- Also means that it’s largely pass/fail. It’ll either work or it won’t.
- We’ve tried to ensure that you can test your own answers, so please do so!

Format

- **Quiz score will be capped at 6**
- If you do the hashing problem and one other, eligible for full marks
- **Better to focus on one problem and get full marks then try for both and get neither of them correct!**

Tips

- If you find yourself writing *a lot* of code, stop. You're doing something wrong. All solutions should be < 100 LOC (especially in Python!)
- Pay attention to what is being provided and asked for. Don't make these common mistakes:
  - Mixing up % vs probability out of 1
  - Input / output formats (hex vs decimal vs ascii)
  - Rounding up vs down (e.g. min guesses for probability P, round up, max users round down)

Certificate Chain

- Don’t be fooled by the amount of setup. This is actually the easier question.
- Answers will be verified in Python. You can use anything you want for your solution, but check with Python!
- Make sure you solve for the right site! Your site should be AXXXXXXXX.sg
- Grading will be done after the quiz closes to run through our script, so don’t be surprised when you don’t get a grade immediately from Luminus
- For that reason, check your answers!

Padding Oracle

- Very interesting problem - fun to get the message at the end
- Much *much* faster to test if you implement your own oracle as a function
- Make sure to include all punctuation (if present) in the message!
- Do remove the pad and submit the actual ASCII message string