ISE Cryptography – Lecture 04

Message Integrity

Announcements

- Midterm is taking place during class next week!
  - Be at your desk and ready to start at 09:00
  - Exam will be on Brightspace, but can't be taken remotely
- Questions will be a mix of MCQs and written responses
  - Based on the content of lectures, exercises and assigned reading
  - Covering Weeks 1 through 4. Weaker Qs from weekly quizzes may be repeated!
- What kind of questions? A mixed approach:
  - Remember, Understand, Apply, Analyze, Evaluate
  - I don't ask trick questions, but I do ask challenging questions
  - If the exam is really easy and everyone gets an A, it's not a useful assessment
  - If the exam is really difficult and everyone gets an F, it's also not useful
  - Expect it to be appropriately challenging!
- Learning off the slides is usually good for a C. You need understanding to get an A!

The Malleability Problem

Confidentiality is not enough!

Confidentiality vs Integrity

- We've built up strong defenses against eavesdropping adversaries
  - Passive adversaries
  - Can read transmitted messages, but can't alter them or send their own
- But we've neglected active adversaries so far
  - Malicious adversaries can modify messages in transit!
- If Bob receives a message from Alice...
  - ...can he convince himself that it wasn't modified in transit?
  - This is the question of ***message integrity***
  - Can any of the schemes we've looked at guarantee this?
- Integrity is orthogonal to message secrecy
  - You don't have to understand a message to alter it
- Many cases where integrity without secrecy is all that's required
  - Ethernet uses a CRC32, TCP uses a 16-bit checksum... any other examples?

CTR Mode is Malleable

- Consider a CTR-mode encrypted payment: Alice sends Bob a transfer
  - The plaintext has a structured format with an amount, sender and recipient
- An adversary intercepts the ciphertext $c = m \oplus F(k, \text{nonce})$
  - They don't know the key. They can't decrypt. But they don't need to!
- CTR mode uses XOR: flipping a bit in c flips the corresponding bit in the decrypted m
  - The adversary knows the structure of the message
  - They XOR the appropriate ciphertext bytes to change the amount
- Bob decrypts successfully and sees a different amount
  - The cipher did exactly what it was supposed to do
  - It delivered confidentiality. Nobody read the message in transit
- But the adversary changed the message without detection
  - CTR mode, CBC mode, and all CPA-secure ciphers that we've seen so far are vulnerable to this
  - We asked the wrong question: we need integrity, not just secrecy

Types of Integrity

- The adversary can intercept a message, alter it and forward it
  - Checksum-style ideas can help with this
  - But anyone can compute a valid checksum! We need to include a secret
- The adversary can also attempt a ***replay attack*** by resending old messages
  - Messages can also be reordered
  - Some messages may be dropped entirely
- Embedding some kind of sequence number in the message can help
  - ...but only if we ensure that the sequence number is valid!
- Keep this idea of tagging a message with associated data in mind for later
  - Message metadata should be tamper-proof, but not necessarily secret

Authenticity

- Integrity asks if the message Bob receives has been altered in transit
  - Some integrity checks guard against problems with the channel, e.g. Ethernet
  - We're interested in guarding against malicious third-party interference!
- Authenticity asks a related, but distinct, question
  - Is the message actually from the supposed sender?
  - Did it really come from Alice, or from an adversary?
  - Can we use a shared secret to guarantee that the message is authentic?
- We're going to tackle both these questions today!

Keyed and Keyless

- Keyless integrity checks are designed to detect transmission errors
  - Random errors due to interference in the transmission medium
  - Anyone can compute a valid CRC32 tag for an Ethernet message
  - No secret involved, analogous to encoding rather than encryption
- We'll discuss keyed integrity checks for our purposes
  - An adversary shouldn't be able to make valid tags for arbitrary messages
  - Only those who know the secret key should be able to do so!
  - More like encryption than encoding
- Tags are generally short, fixed-length bitstrings
  - Only a small transmission overhead

MAC Definition

Tagging messages with a shared secret.

Message Authentication Code (MAC)

- A ***Message Authentication Code (MAC)*** ensures that a message has not been tampered with during transmission
  - Uses a shared secret key between two parties to verify message integrity
  - Why? So the adversary can't produce valid MACs!
- A MAC system $\mathcal{I} = (S, V)$ consists of two efficient algorithms:
  - S is the ***signing algorithm*** used to generate tags
  - V is the ***verification algorithm*** used to validate tags
- MACs are defined over the key, message and tag spaces $(\mathcal{K}, \mathcal{M}, \mathcal{T})$
- S is a probabilistic algorithm invoked as $t \leftarrow S(k, m)$
  - t is the ***tag***, k is the key, and m is the message
  - S can be probabilistic, but doesn't have to be
- V is a deterministic algorithm invoked as $r \leftarrow V(k, m, t)$
  - The output r is either `accept` or `reject`
- MACs must satisfy their ***correctness property*** for all keys k and messages m
  - $\Pr[V(k, m, S(k, m)) = \texttt{accept}] = 1$

Randomised vs Deterministic

- Our definitions for computational ciphers also allowed for encryption to be probabilistic
  - And we did! CBC with a random IV is probabilistic encryption
  - With MACs, probabilistic versions are also common in practice
- ***Deterministic MAC system***
  - Same input (k, m) always produces the same tag t
  - Verification is simple: `accept` if $S(k, m) = t$
  - Guarantees unique tag per message
- ***Randomized MAC system***
  - Same (k, m) might produce different valid tags on different runs
  - May provide better trade-offs in terms of security and efficiency
- Both types can be secure!

The UF-CMA Game

Existential unforgeability under chosen message attack.

Defining MAC Security

- Let's take a conservative approach to security. If a MAC can hold up in a hostile environment against powerful adversaries, then it's good for day-to-day use too!
- As usual, let's build an attack game to demonstrate!
- Let's hold the key k constant, like we did for cipher attack games
- The adversary can request tags for millions of arbitrary messages of their choice
  - They can request $t = S(k, m)$ for arbitrary m
  - The result is a signed pair (m, t)
  - This is called a ***chosen message attack***
- We challenge the adversary to forge a new valid message-tag pair (m, t)
  - New? Has to be different from all of the signed pairs generated so far
  - This is called an ***existential MAC forgery***

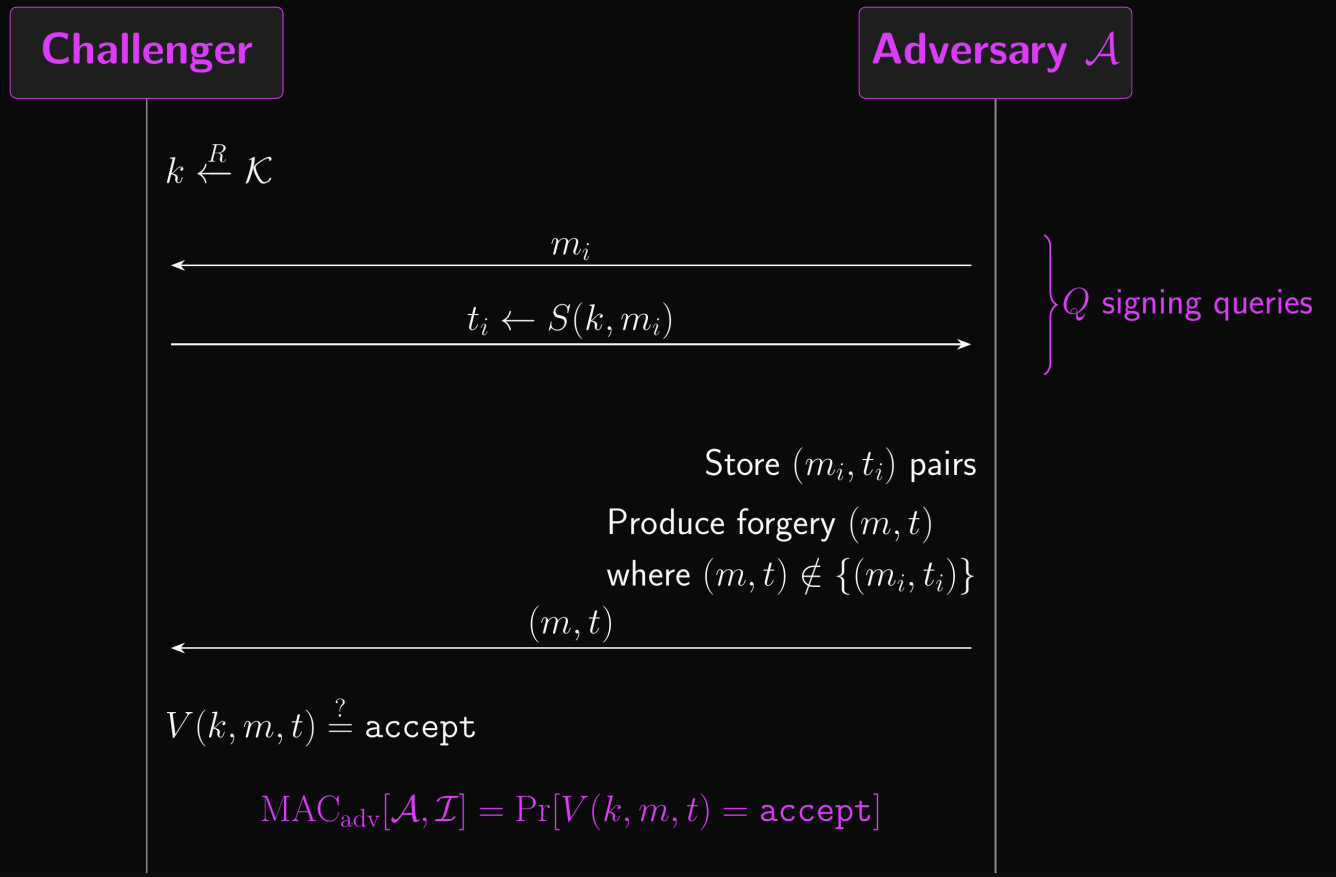
Why Chosen Messages? Why Existential Forgery?

- Chosen message attack seems too generous to the adversary. Why allow it?
  - In practice, adversaries often can trigger MACs on messages of their choice
  - A web server might MAC cookie values an attacker controls
  - A protocol might authenticate attacker-influenced metadata
  - If the MAC is secure even when the adversary picks the messages, it's secure everywhere
- Existential forgery seems too weak. Who cares about forging a random message?
  - If the adversary can't forge a tag for any message of their choosing...
  - ...they certainly can't forge one for a specific target message!
  - Conservative definitions make for stronger guarantees

MAC Security: Formal Definition

- The challenger picks a random $k \in \mathcal{K}$
- $\mathcal{A}$ issues Q signing queries to the challenger
  - The ith signing query is a message $m_i \in \mathcal{M}$
  - The challenger computes $t_i \leftarrow S(k, m_i)$ and returns t_i to $\mathcal{A}$
  - $\mathcal{A}$ stores the signed pair (m_i, t_i)
- $\mathcal{A}$ outputs a candidate forgery pair $(m, t) \in \mathcal{M} \times \mathcal{T}$
  - Where (m, t) is different from all of the signed pairs queried by $\mathcal{A}$
- $\mathcal{A}$ wins if (m, t) is a valid pair under k, i.e. if $V(k, m, t) =$ `accept`
  - $\mathrm{MAC}_{\mathrm{adv}}[\mathcal{A}, \mathcal{I}] = \Pr[V(k, m, t) = \texttt{accept}]$
- $\mathcal{A}$ is a Q-query MAC adversary
- A MAC system $\mathcal{I}$ is secure if for all efficient adversaries $\mathcal{A}$
  - $\mathrm{MAC}_{\mathrm{adv}}[\mathcal{A}, \mathcal{I}]$ is negligible
- Secure MAC systems are ***existentially unforgeable under chosen message attack***

MAC Security: Attack Game



Deterministic vs Randomised Security

- A MAC system can be deterministic or randomized
- For deterministic MAC systems, security implications are simpler
  - The adversary must produce a valid tag for a new message
  - Similar to block cipher unpredictability
- For randomized MAC systems, this is a stronger security property
  - May be many valid tags for a given message
  - A new, valid tag for an already-queried message is still an existential forgery!
- In general, having a valid tag for a message gives no useful information...
  - To generate a valid tag for a different message
  - To generate a new, valid tag for the same message
- We could also allow the adversary to perform verification queries in addition to signing queries. This seems like an even more powerful adversary...
  - But it turns out that the two attack games are equivalent (with an error term)
  - Intuition: the challenger can just reply `reject` to all verification queries. If that answer is ever wrong, the adversary already submitted a valid forgery!

Why Freshness? Replay is Trivial

- Note that the UF-CMA game requires the forgery (m, t) to be fresh
  - The adversary can't just replay a pair they already received
- This isn't because replay attacks don't matter. They absolutely do!
  - Replaying a valid "transfer funds" message is a real attack
- But replay protection is a protocol-level concern, not a MAC-level concern
  - Sequence numbers, timestamps, and nonces handle replay at the protocol layer
- The MAC's job is to prevent the adversary from creating new valid tags
  - If they can't forge, they can only replay (which the protocol blocks)

One-Time MACs

Information-theoretic security for a single message.

The Polynomial MAC

- Before building multi-message MACs from PRFs, let's see that perfect one-time security is achievable
  - No computational assumptions needed! Secure against unbounded adversaries
  - But only for a single message
- This mirrors the OTP/stream cipher pattern from Lecture 01
  - The OTP is perfectly secret for one message; PRG-based ciphers extend to many
  - The one-time MAC is perfectly unforgeable for one message; PRF-based MACs extend to many
  - The limitation motivates why we need computational assumptions for multi-message security
- Construction: work in $\mathbb{Z}_p$ for a large prime p
  - Key is a pair $(k_1, k_2) \xleftarrow{R} \mathbb{Z}_p^2$
  - For a message $m = (a_1, \ldots, a_v)$ with each $a_i \in \mathbb{Z}_p$:
  - $S((k_1, k_2), m) = a_1 k_1^v + a_2 k_1^{v-1} + \cdots + a_v k_1 + k_2 \mod p$
  - Evaluate the message as a polynomial at k_1, then mask with k_2
  - Example: if $m = (a_1, a_2, a_3)$, the tag is $a_1 k_1^3 + a_2 k_1^2 + a_3 k_1 + k_2$
  - The message defines the polynomial; the secret k_1 is where we evaluate it

One-Time MAC Security

- With a single signing query, the adversary learns one point on the polynomial
  - The polynomial has degree v, so one point reveals almost nothing about k_1
  - And k_2 acts as a one-time pad on the output
- For any message $m' \neq m$, the probability of guessing the correct tag is at most $(v+1)/p$
  - With $p \approx 2^{128}$, this is negligible
- But with two signing queries, the adversary learns two points
  - Two points on a degree-v polynomial with a linear mask: k_1 can be recovered
  - k_2 is then trivially computed
  - The MAC is completely broken after reuse!
- This is the fundamental limitation: one-time security requires fresh keys
  - For multi-message security, we need computational assumptions (PRFs)
  - But one-time MACs are not just a curiosity: Poly1305 (used in ChaCha20-Poly1305) is exactly this kind of polynomial MAC

Building MACs from PRFs

The PRF-to-MAC reduction.

When You Have a Hammer...

- Let's say we already have a secure PRF. Can we turn it into a MAC?
  - Remember what a PRF is from Lecture 03?
  - Block ciphers like AES (under certain conditions) are secure PRFs
  - No point inventing something brand-new, right?
- It's surprisingly easy! Here's how we can do it:
  - Define $S(k, m) = F(k, m)$, i.e. just apply the PRF to the message.
  - To verify, check if $F(k, m) = t$. If so, accept; if not, reject.
- This gives us a deterministic MAC
  - A single valid tag t exists for a given message m under a fixed key k
- The security comes from the PRF's unpredictability
  - If F is a secure PRF, attackers can't predict tags for new messages
- Well, that was suspiciously easy, wasn't it?
  - How big is the tag if we use AES? 128 bits, so forgery probability is $1/2^{128}$. That's fine!
  - How big is the message if we use AES? Also 128 bits. That's a problem!

The Reduction: Intuition

- Let's backtrack to our definition of MAC security. Does a secure PRF pass the test?
- To make this easier, let's pretend that our PRF F is a truly random function
- The adversary can make as many queries as they want to, but it won't help
- When they want to find $F(m)$ for a new m they haven't queried yet...
  - They're completely in the dark!
  - $F(m)$ is random and independent of previous signed pairs
- Their best strategy is to guess randomly, so the output space must be big enough
  - Chance of guessing correctly is $1/|\mathcal{Y}|$
  - $|\mathcal{Y}|$ should be super-poly to make $1/|\mathcal{Y}|$ negligible
- All very nice, but how does this help with a secure PRF?
  - Secure PRFs are indistinguishable from truly random functions...
  - And guessing or brute-forcing random outputs isn't feasible...
- The same reduction pattern from Lecture 02 applies
  - We'll build an adversary $\mathcal{B}$ that turns any MAC forger into a PRF distinguisher

The Reduction: Formal Argument

- Suppose adversary $\mathcal{A}$ can forge a MAC with non-negligible advantage
  - Build a PRF distinguisher $\mathcal{B}$ that uses $\mathcal{A}$ as a subroutine
  - $\mathcal{B}$ receives oracle access to either $F(k, \cdot)$ or a random function f
  - $\mathcal{B}$ forwards $\mathcal{A}$'s signing queries to its own oracle
  - When $\mathcal{A}$ outputs a forgery (m, t), $\mathcal{B}$ checks if t equals the oracle's output on m
- If the oracle is random, $\mathcal{A}$ succeeds with probability at most $1/|\mathcal{Y}|$
  - If the oracle is $F(k, \cdot)$ and $\mathcal{A}$ forges, $\mathcal{B}$ distinguishes with the same advantage
- So $\mathrm{MAC}_{\mathrm{adv}}[\mathcal{A}, \mathcal{I}] \leq \mathrm{PRF}_{\mathrm{adv}}[\mathcal{B}, F] + 1/|\mathcal{Y}|$
  - Secure PRF implies secure MAC
  - This is the course's first full ***proof by reduction***!
- We proved MAC security without knowing anything about how F works internally
  - Only that F is a secure PRF
  - The same reduction template recurs for digital signatures, CCA security, and key exchange

Long Messages

- As we've pointed out, there's a minor problem with building MACs directly from secure PRFs like AES
  - Messages tend to be a lot longer than the block size
  - A MAC that can only verify integrity for 128-bit strings is pretty useless
- But wait! We've solved this problem for block ciphers already, haven't we?
- Given a secure PRF on short inputs, let's try to build a secure PRF on long inputs
  - This is an analogous idea to block cipher modes of operation
  - In fact, some modes of operation come with built-in authentication and integrity checks!

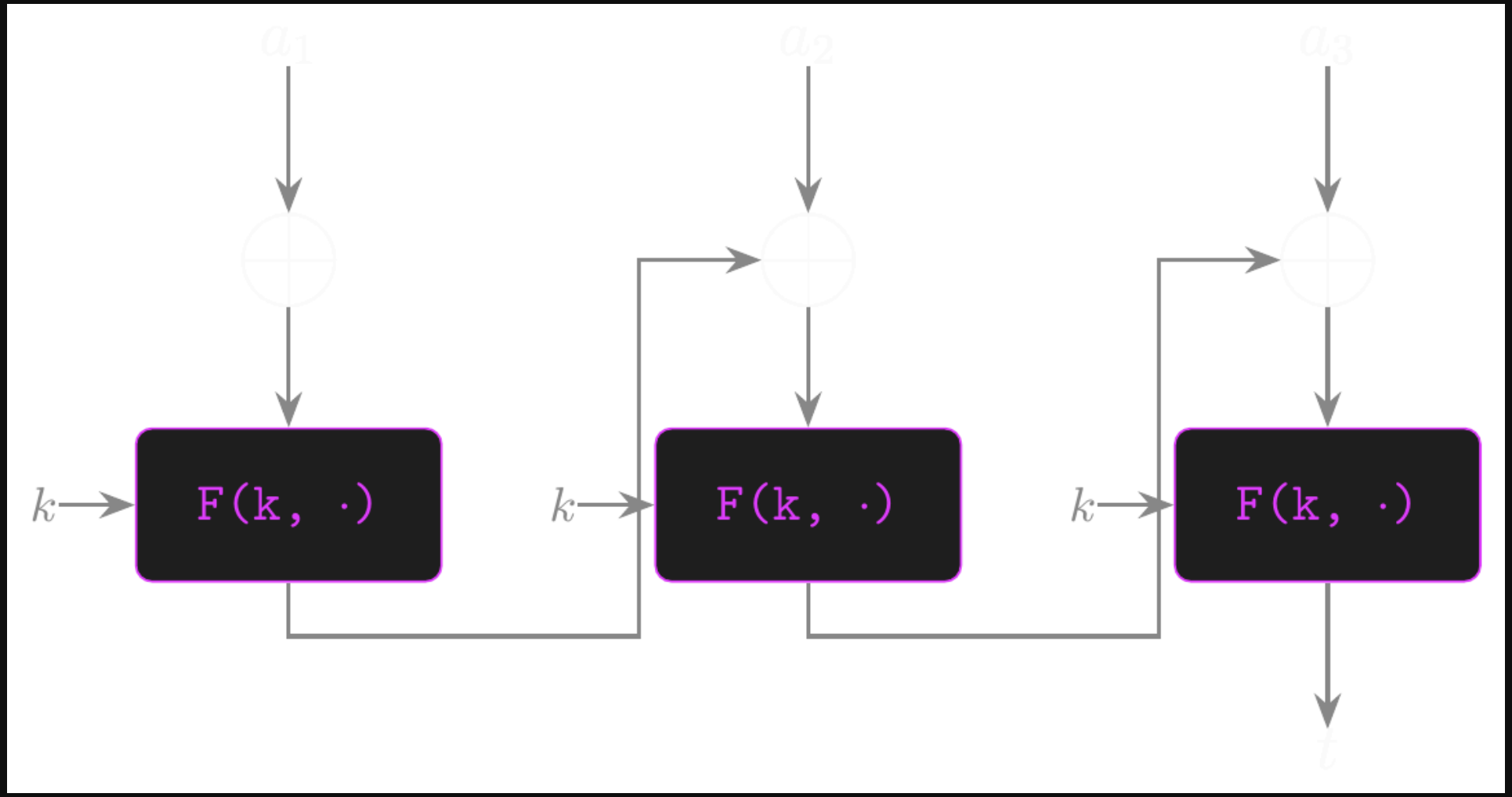
CBC-MAC and the Length Extension Problem

When prefix-freedom matters.

CBC-MAC

- Let's start by setting some conditions for the secure PRF we'll use as a building block
- F is a secure PRF that maps n-bit inputs to n-bit outputs
  - F is defined over $(\mathcal{K}, \mathcal{X}, \mathcal{X})$ where $\mathcal{X} = \{0,1\}^n$
- Our new PRF F_{CBC} maps messages in $\mathcal{X}^{\leq \ell}$ to tags in $\mathcal{X}$
  - ℓ is poly-bounded
- ***CBC-MAC*** is based on CBC mode: chain message blocks through a PRF to get one final output to use as the tag. Changing any message block changes the tag!
  - Split the message into blocks as usual, so $m = (a_1, a_2, \ldots, a_v)$
  - CBC-MAC uses a fixed, all-zero IV, so start with $t = 0^n$
  - For each block, let $t \leftarrow F(k, a_i \oplus t)$, and the final value of t is the tag!
- Just like CBC encryption, but...
  - The IV is fixed, not unpredictable
  - No intermediate outputs, just the final tag
- If adversaries are restricted to ***prefix-free*** queries, CBC-MAC behaves like a random function

CBC Construction: $F_{\mathrm{CBC}}(k, m)$

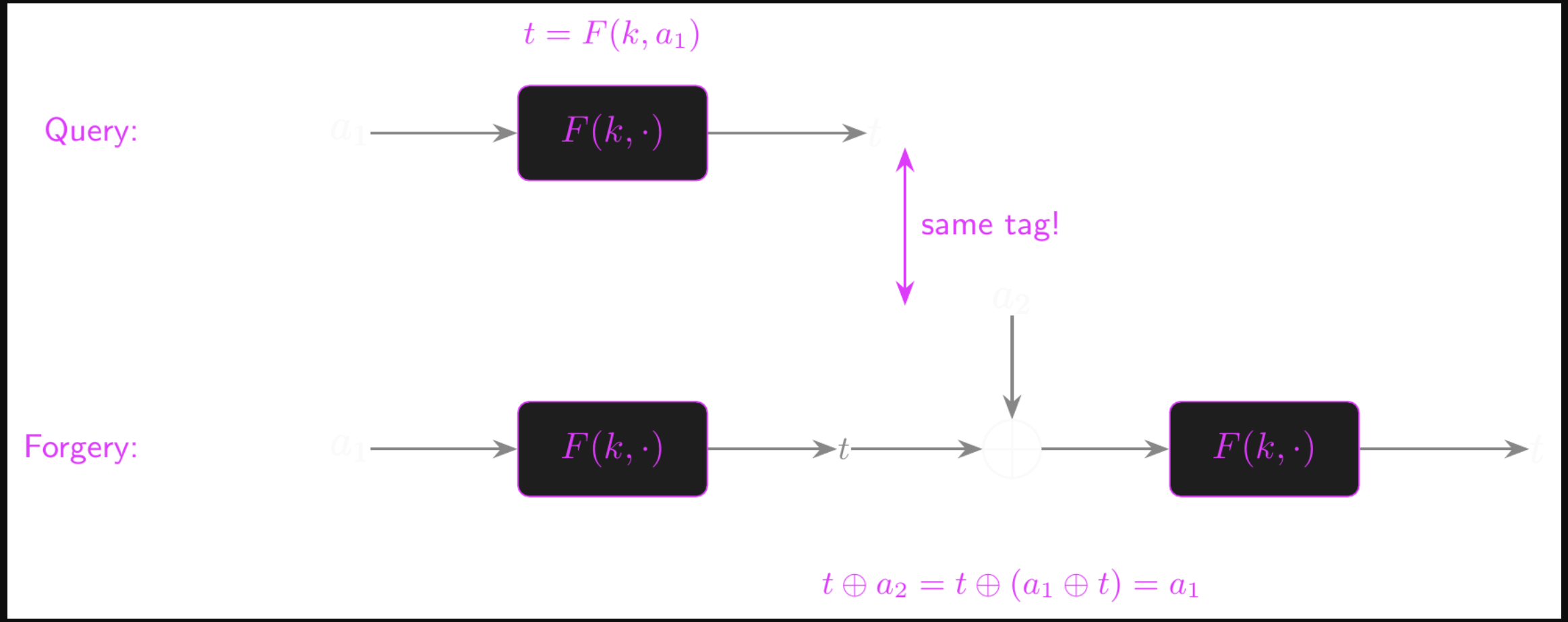


XORing the all-zero IV is a no-op, so it's excluded from the diagram.

The Length Extension Attack on CBC-MAC

- CBC-MAC on its own is not a secure MAC for variable-length messages!
  - Pick an arbitrary $a_1 \in \mathcal{X}$
  - Request the tag t for the message $m = (a_1)$
  - Let $a_2 = a_1 \oplus t$
  - Return t as a forgery for the message (a_1, a_2)
- How does this work? Let's see...
  - We defined $a_2 = a_1 \oplus t$ where $t = F(k, a_1)$, so $F(k, a_1) \oplus a_2 = t \oplus (a_1 \oplus t) = a_1$
  - $F_{\mathrm{CBC}}(k, (a_1, a_2)) = F(k, F(k, a_1) \oplus a_2)$
  - $= F(k, a_1)$
  - $= t$
- $((a_1, a_2), t)$ is an existential forgery for CBC-MAC, so F_{CBC} must be insecure!
  - The root cause: the ***extension property*** of CBC-MAC lets an adversary continue the chain
  - CBC-MAC is only secure when all messages have the same fixed length

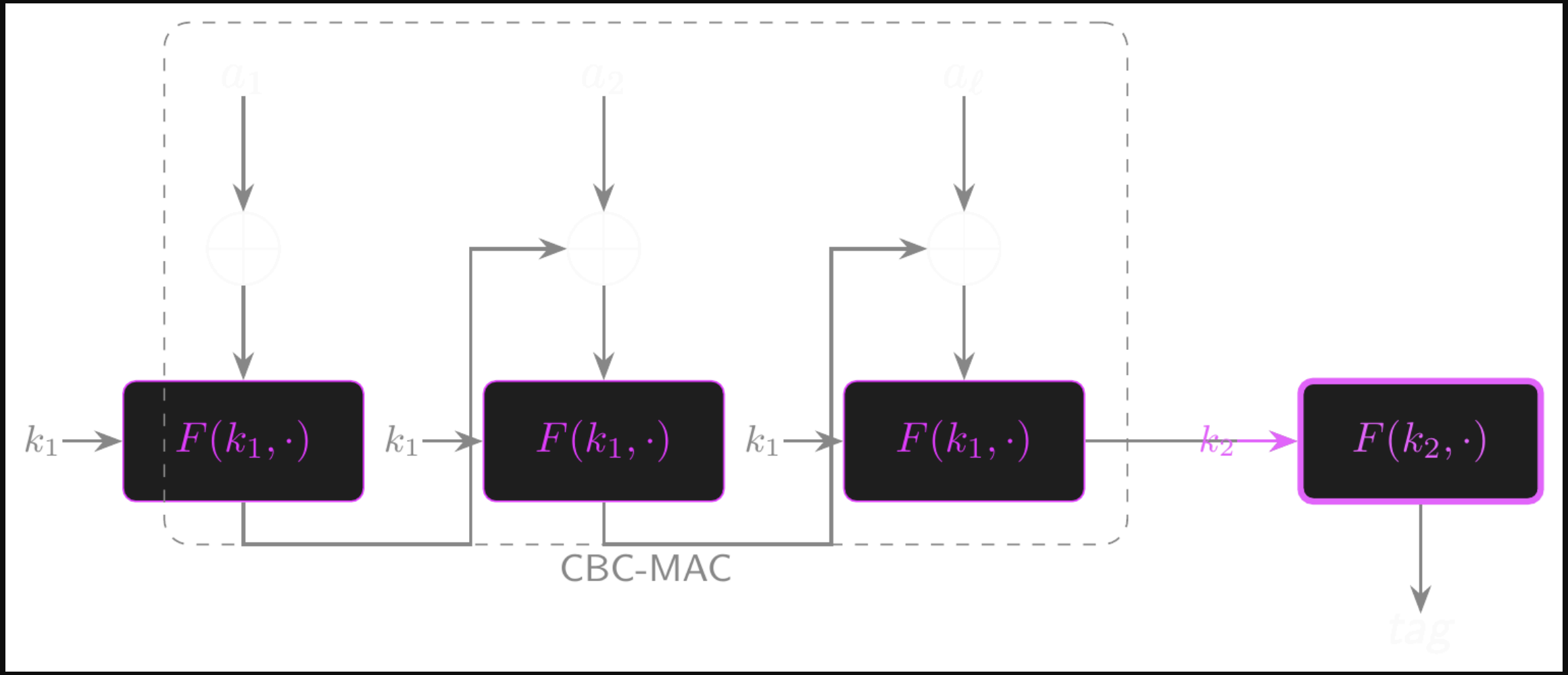
Extension Attack on CBC-MAC



Fixing CBC-MAC: Three Approaches

- Given a prefix-free secure PRF like CBC-MAC, how do we make it fully secure?
- Approach 1: ***Encrypted CBC-MAC (ECBC)***
  - Encrypt the final output with a second key: $\mathrm{ECBC}((k_1, k_2), m) = F(k_2, F_{\mathrm{CBC}}(k_1, m))$
  - Hides the internal state; the extension property is broken
  - Two independent keys are essential!
- Approach 2: Prepend message length
  - $pf(m) = (\langle v \rangle, a_1, \ldots, a_v)$ where $\langle v \rangle$ is a fixed-width encoding of the block count v
  - No message can be a prefix of another if lengths differ
  - Drawback: message length must be known before processing starts (no streaming)
- Approach 3: ***CMAC***
  - XOR the last block with a derived subkey before the final PRF evaluation
  - Different subkeys for full and partial last blocks (avoids padding collisions)
  - Standardized by NIST in 2005; used in AES-CMAC
  - Streaming, efficient, no extra PRF invocations, no dummy blocks

ECBC Construction: $\mathrm{ECBC}((k_1, k_2), m)$



HMAC

Failure cascade and construction.

Building MACs from Hash Functions

- CBC-MAC and CMAC are built from block ciphers (PRFs)
  - What about building MACs from cryptographic hash functions instead?
- Cryptographic hash functions like SHA-256 are keyless: $H : \mathcal{M} \to \mathcal{T}$
  - No secret key involved, so $H(m)$ alone is not a MAC
  - Anyone could compute $H(m)$!
- We need to incorporate a secret key somehow
  - The obvious approaches all fail (in instructive ways!)
  - Let's walk through the failure cascade

Attempt 1: $H(k||m)$

- Prepend the key to the message and hash
- This fails due to the length extension property of Merkle-Damgard hashes (a chained compression-function design used by SHA-256, which we'll formalise shortly)
  - SHA-256 processes input in blocks through a compression function
  - The final internal state is the hash output
  - Given $H(k||m)$, an attacker can continue hashing from that state
  - They can compute $H(k||m||\text{padding}||m')$ without knowing k!
- This is exactly the cascade extension attack from CBC-MAC, in a different setting
  - The hash function's internal chaining is exploitable

Attempt 2: $H(m||k)$

- Append the key to the message instead
- This blocks the length extension attack: the key comes last, so an adversary can't extend
- But it's vulnerable to collision attacks on H
  - If the adversary finds $m_0 \neq m_1$ with $H(m_0) = H(m_1)$ (two distinct inputs with the same hash, called a ***collision***)...
  - ...then $H(m_0||k) = H(m_1||k)$ for any key k
  - The adversary can forge without knowing the key!
- Collision resistance of the hash function directly limits MAC security
  - For broken hashes (MD5, SHA-1), this is a real problem

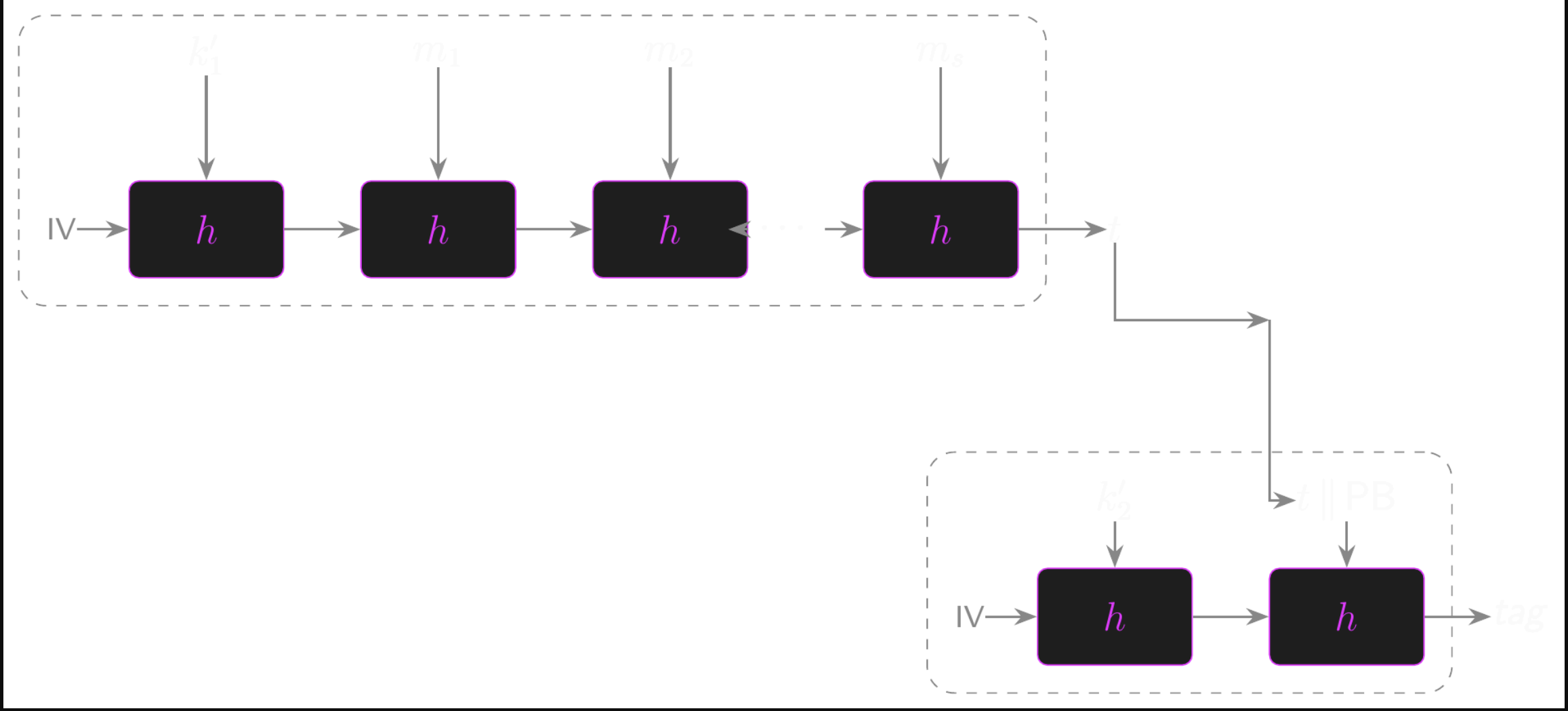
Attempt 3: $H(k||m||k)$

- Sandwich the message between two copies of the key
- This blocks both extension attacks and simple collision attacks
- But the security analysis is fragile and depends on hash internals
  - No clean reduction to standard hash function properties
  - Difficult to prove secure in any standard model

HMAC: The Solution

- ***HMAC*** uses a nested construction with two derived keys
- Let B be the block size of the hash function in bytes
  - If key k is longer than B bytes, hash it: $k' \leftarrow H(k)$
  - Otherwise, pad k with zero bytes until it's B bytes long
- Derive two keys:
  - Let `ipad` = the byte `0x36` repeated B times, `opad` = the byte `0x5C` repeated B times
  - $k'_1 \leftarrow k' \oplus \text{ipad}$, $k'_2 \leftarrow k' \oplus \text{opad}$
- $\text{HMAC}(k, m) = H(k'_2 \| H(k'_1 \| m))$
  - Inner hash: $H(k'_1 \| m)$ compresses the keyed message to a fixed-size digest
  - Outer hash: $H(k'_2 \| \text{inner})$ applies a second keyed hash to the result
- Why does this work?
  - Length extension on the inner hash gives $H(k'_1 \| m \| \text{ext})$, but that value gets hashed again with a different key in the outer hash
  - Collision attacks on the inner hash don't help because the outer hash re-keys

HMAC Structure



HMAC in Practice

- HMAC is provably secure when the compression function of H is a PRF
- The chosen hash is appended to the name of the MAC
  - E.g. HMAC-MD5, HMAC-SHA1 and HMAC-SHA256/384 are all found in TLS 1.2
  - Dropped from TLS 1.3 in favour of AEAD
- HMAC's security depends on that of the hash function selected
  - Don't throw a generic hash function into HMAC and expect good results!
- HMAC works with almost any strong cryptographic hash function
  - And even some broken ones!
  - HMAC-MD5 has no practical attacks (but shouldn't be used)
- Tends to be much faster than CBC-MAC and other alternatives

Hash Function Properties

Collision resistance, preimage resistance, and the Random Oracle Model.

Cryptographic Hash Functions

- What makes a hash function a ***cryptographic hash function (CHF)***?
  - SHA-256 is an example of a CHF
  - MD5 is an example of an extremely broken CHF
- ***Collision resistance***: it should be hard to find any two distinct messages $m_0 \neq m_1$ such that $H(m_0) = H(m_1)$
  - Can't find two messages that produce the same digest!
  - The birthday attack finds collisions in about $2^{n/2}$ evaluations for an n-bit hash
  - This is why SHA-256 (256-bit output) provides only 128-bit collision resistance
- ***Preimage resistance***: given a digest $t \in \mathcal{T}$, it should be hard to find any $m \in \mathcal{M}$ such that $H(m) = t$
  - The hash function should be irreversible
- ***Second preimage resistance***: given some m_0, it should be hard to find $m_1 \neq m_0$ such that $H(m_0) = H(m_1)$
  - Can't change the message without changing the hash
  - ***Avalanche effect***: even a small change to the input should drastically change the output

These Properties are Not Equivalent

- For compressing hash functions, the implications form a chain:
  - Collision resistance $\Rightarrow$ 2nd-preimage resistance $\Rightarrow$ preimage resistance
  - Finding m_1 given m_0 with $H(m_0) = H(m_1)$ is already a collision (CR $\Rightarrow$ SPR)
  - Inverting a hash gives a second preimage for most inputs (SPR $\Rightarrow$ OW)
- The converses do NOT hold:
  - Preimage resistance does NOT imply collision resistance
  - A hash can be hard to invert but easy to find collisions for
- Precision matters when evaluating hash function security
  - MD5 and SHA-1 have broken collision resistance
  - Their preimage resistance is still considered intact (no practical attacks)
  - But you still shouldn't use them for new systems!
- SHA-256 and SHA-3 have no known breaks for any of these properties
  - SHA-256 is Merkle-Damgard based (vulnerable to length extension, but not broken)
  - SHA-3 uses the sponge construction (no length extension vulnerability)

The Random Oracle Model

- Sometimes we need a stronger assumption than collision resistance
- The ***Random Oracle Model (ROM)*** treats the hash function as a truly random function
  - For every input, the output is uniformly random and independent
  - The only way to learn $H(m)$ is to query the oracle
- This is a heuristic, not a theorem: real hash functions are deterministic algorithms
  - But the ROM gives us a framework for proving security of constructions like HMAC
- You'll encounter "secure in the ROM" in Lectures 05 and 06
  - RSA-FDH signatures (Lecture 06) rely on the ROM for their security proof
  - It's worth understanding what this means and what its limitations are

Authenticated Encryption

Combining secrecy and integrity.

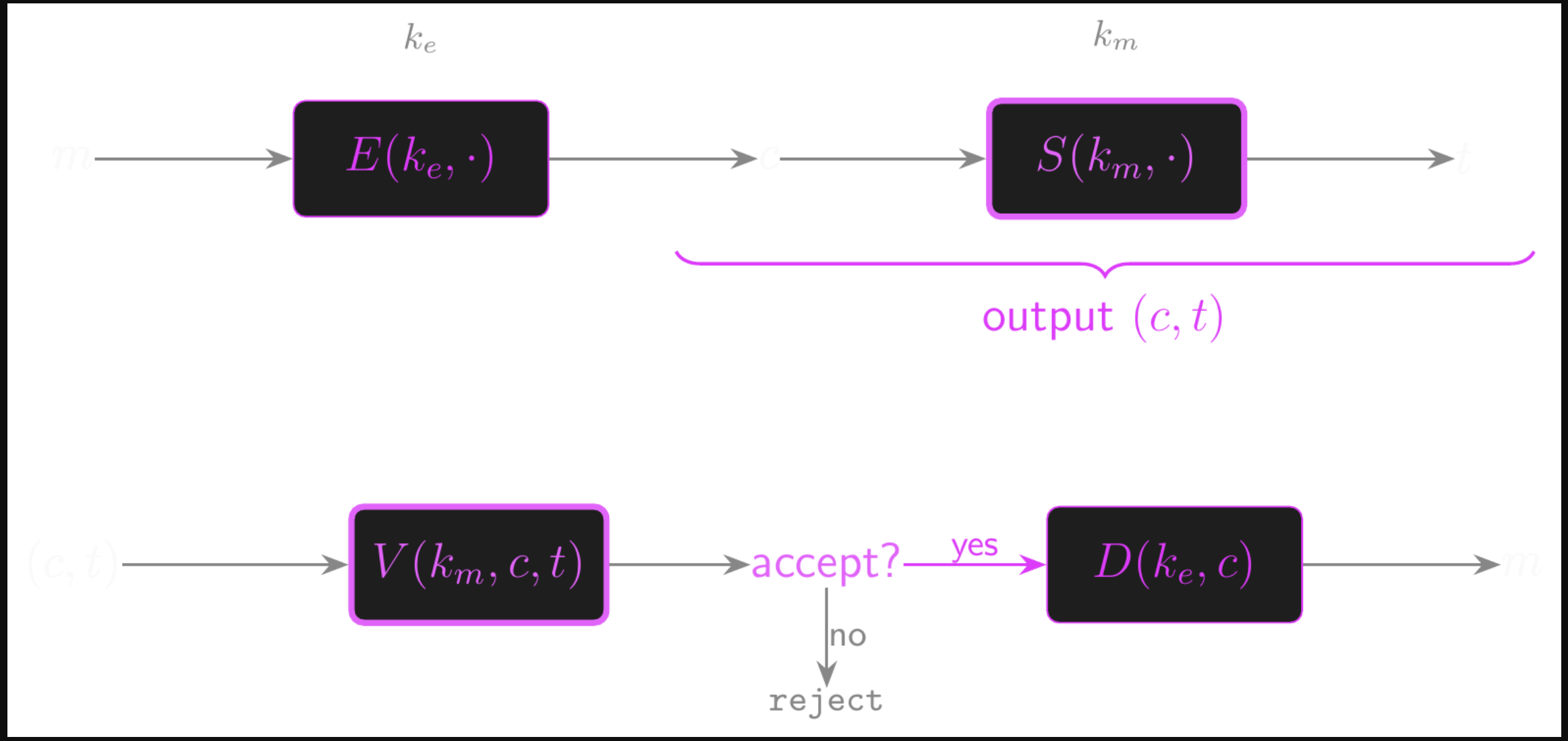
The Composition Problem

- We now have two independent tools:
  - CPA-secure encryption (confidentiality, from Lectures 02 & 03)
  - Secure MACs (integrity, from this lecture)
- Real-world adversaries are active: they can tamper with ciphertexts in transit
  - WiFi, Ethernet and Bluetooth can all be interfered with relatively easily
  - CPA-secure schemes allow undetected tampering with ciphertext
  - But breaking integrity can also break secrecy!
- We should always combine encryption with authentication
  - But the order of composition matters!

Three Compositions

- ***Encrypt-then-MAC***: encrypt m, then MAC the ciphertext
  - $c \leftarrow E(k_{\text{encrypt}}, m), t \leftarrow S(k_{\text{mac}}, c)$, output (c, t)
  - To decrypt: check MAC first, reject if invalid, then decrypt
  - Secure in general! If the MAC is secure, the adversary can't submit valid modified ciphertexts
- ***MAC-then-Encrypt***: MAC the message, then encrypt message and tag together
  - $t \leftarrow S(k_{\text{mac}}, m), c \leftarrow E(k_{\text{encrypt}}, (m, t))$, output c
  - Not generally secure! Padding attacks are devastating
  - The POODLE attack against SSL 3.0 exploits exactly this weakness
  - All TLS 1.0-1.2 CBC cipher suites used MAC-then-Encrypt; TLS 1.3 dropped them entirely
- ***Encrypt-and-MAC***: encrypt m and MAC m independently
  - $c \leftarrow E(k_{\text{encrypt}}, m), t \leftarrow S(k_{\text{mac}}, m)$, output (c, t)
  - Not generally secure! The MAC tag may leak information about m
- Both keys must be independent in all cases: $k_{\text{encrypt}} \neq k_{\text{mac}}$

Encrypt-then-MAC



The CCA Theorem

- Encrypt-then-MAC with a CPA-secure cipher and a secure MAC achieves ***IND-CCA2*** security
  - CCA2 is the strongest standard notion of encryption security
  - We'll define IND-CCA2 formally in Lecture 06 and prove this theorem in Lecture 07
- The intuition: the MAC prevents the adversary from submitting modified ciphertexts
  - Any modification invalidates the tag
  - So the CCA2 decryption oracle is effectively neutered
  - The adversary is left with only chosen-plaintext capability, which CPA handles
- This is why modern protocols exclusively use ***authenticated encryption***
  - There is no good reason to use encryption without authentication

One-Time Security

- Sometimes we can make do with a weaker property instead of AE security
- In some contexts, like encrypted emails, we might use ephemeral keys
  - And each ephemeral key might be used for only a single message
  - In that case, being AE-secure is more than sufficient
- We could weaken the definition to one-time AE security
  - The adversary will only get a single attempt at each key in practice
  - So why not make the definition of security reflect that?
- Typically, we use the same CPA-secure symmetric cipher as usual
  - Remember the polynomial one-time MAC? Combined with a CPA-secure cipher, it gives one-time AE
  - Unforgeable if the adversary only gets a single chosen-message query

CCA Security

- The good news is that an AE-secure symmetric cipher is also CCA-secure
  - So long as we're doing authenticated encryption, we're secure against chosen-ciphertext attacks
- There's little-to-no reason to use any symmetric cipher that isn't AE-secure
- TLS 1.3 includes some gold-standard AE constructs
  - AES-GCM
  - AES-CCM
  - ChaCha20-Poly1305
- These are ***AEAD*** schemes: authenticated encryption with associated data

AEAD in Practice

Associated data and nonce discipline.

AEAD: Authenticated Encryption with Associated Data

- Real-world protocols often need to authenticate metadata alongside the message
  - Packet headers must be readable for routing but protected from tampering
  - Metadata in secure file systems should be visible but not malleable
- A ***nonce-based AEAD cipher*** $\mathcal{E} = (E, D)$ takes four inputs:
  - Key $k \in \mathcal{K}$, message $m \in \mathcal{M}$, associated data $d \in \mathcal{D}$, nonce $n \in \mathcal{N}$
- Encryption: $c = E(k, m, d, n)$
- Decryption: $m = D(k, c, d, n)$, or `reject` if integrity check fails
- The authentication tag covers both the ciphertext and the associated data
  - Tampering with either one is detected on decryption
- If d is empty, behaves like ordinary authenticated encryption
- If m is empty, behaves like a MAC over the associated data

AES-GCM

- ***Galois Counter Mode (GCM)*** is the most widely deployed AEAD cipher
  - Standardized by NIST in 2007; used in TLS 1.3, IPsec, and many other protocols
- GCM follows encrypt-then-MAC internally:
  - Encryption: AES in counter mode (CTR)
  - Authentication: a polynomial MAC (GHASH) applied to the ciphertext and associated data
- Hardware support makes AES-GCM extremely fast on modern processors
  - AES-NI for encryption, `PCLMULQDQ` for Galois field multiplication

ChaCha20-Poly1305

- The alternative when AES hardware support is unavailable
  - ChaCha20 is a stream cipher (from Lecture 02); Poly1305 is a polynomial one-time MAC (recall the construction from earlier)
  - Combined as an AEAD cipher in RFC 7539
- Both AES-GCM and ChaCha20-Poly1305 are available in TLS 1.3
  - Either is a good choice for the EPIC project's secure messaging component

Nonce Discipline: The Operational Cliff Edge

- AEAD schemes require a unique nonce for every encryption under the same key
  - The nonce doesn't have to be secret, just unique
  - Can be transmitted in the clear alongside the ciphertext
  - TLS 1.3 derives a per-connection IV and XORs it with the record sequence number, guaranteeing uniqueness without randomness
- ***Nonce reuse*** in AES-GCM is catastrophic
  - Two messages encrypted with the same key and nonce: plaintexts are leaked via XOR
  - The authentication key k_m is exposed, breaking ciphertext integrity entirely
  - Not just the two affected messages: all messages under that key are compromised
- AES-GCM-SIV provides some nonce misuse resistance at the cost of performance
  - Still leaks whether two plaintexts are equal under nonce reuse
  - But doesn't catastrophically break authentication
- GCM uses a 32-bit internal block counter, limiting each key to 2^{32} encrypted blocks
  - Beyond this, the authentication guarantee degrades
  - Once the limit is approached, keys must be rotated

Conclusion

What did we learn?

Where do we go from here?

- We've solved the integrity problem that CTR mode and CPA-secure ciphers left open
  - Message authentication via MAC systems (UF-CMA security)
  - Building MACs from PRFs (with a full security reduction)
  - Building MACs from hash functions (HMAC and the failure cascade)
  - Combining encryption and authentication (authenticated encryption)
- But every symmetric primitive assumes a pre-shared key
  - How did Alice and Bob get the key in the first place?
  - Kerckhoffs's principle means the key is the only secret
  - Key establishment is the existential problem of symmetric cryptography
- Next lecture: public key cryptography and key exchange

For next time...

- “Knowledge cutoff” for the midterm is now!
  - If you can explain/teach the material, then you understand it
  - Attend, review and revise the material to get maximum benefit
- Bloom’s taxonomy is a good guide for learning and exam prep
  - Remember, Understand, Apply, Analyze, Evaluate, and Create
- No formally assigned reading for Week 5, focus on the midterm!
- But if you’d like more detail, then look at the relevant sections of Crypto 101
- Chapters 6, 7 & 8 of A Graduate Course in Applied Cryptography
  - Obviously much more detailed and proof-heavy than today’s lecture
  - About 120 pages, so skim for value-add sections
  - But reading proofs can help intuitive understanding. Up to you!

Questions?

Ask now, catch me after class, or email eoin@eoin.ai