HMAC Properties

- HMAC(K, M) = $H[(K^+ \oplus opad) \,||\, H((K^+ \oplus ipad) \,||\, M]]$
- HMAC is a hash function, so it has the properties of the underlying hash too
  - It is collision resistant
  - Given HMAC(K, M), an attacker can't learn M – *one way*
  - If the underlying hash is secure, HMAC doesn't reveal M, but it is still deterministic
- You can't verify a tag T if you don't have K
- This means that an attacker can't brute-force the message M without knowing K

MACs: Summary

- Inputs: a secret key and a message
- Output: a tag on the message
- A secure MAC is unforgeable: Even if David can trick Alice into creating MACs for messages that David chooses, David cannot create a valid MAC on a message that she hasn't seen before
  - Example: HMAC(K, M) = $H((K^+ \oplus opad) \,||\, H((K^+ \oplus ipad) \,||\, M))$
- MACs do not provide confidentiality

Do MACs provide integrity?

- Do MACs provide integrity?
  - Yes. An attacker cannot tamper with the message without being detected
- Do MACs provide authenticity?
  - It depends on your threat model
  - If only two people have the secret key, MACs provide authenticity: it has a valid MAC, and it's not from me, so it must be from the other person
  - More than one secret key, If a message has a valid MAC, you can be sure it came from *someone with the secret key*, but you can't narrow it down to one person
- Do MACs provide confidentiality?

Authenticated Encryption: Definition

- **Authenticated encryption** (**AE**): A scheme that simultaneously guarantees confidentiality and integrity (and authenticity, depending on your threat model) on a message
- Two ways of achieving authenticated encryption:
  - Combine schemes that provide confidentiality with schemes that provide integrity
  - Use a scheme that is designed to provide confidentiality and integrity

Scratchpad: Let's design it together

- You can use:
  - An encryption scheme (e.g. AES-CBC): Enc(K, M) and Dec(K, M)
  - An unforgeable MAC scheme (e.g. HMAC): MAC(K, M)
- First attempt: Alice sends Enc(K_1, M) and MAC(K_2, M)
  - Integrity? Yes, attacker can't tamper with the MAC
  - Confidentiality? No, the MAC is not secure
- Idea 1: Let's compute the MAC on the *ciphertext* instead of the plaintext: Enc(K_1, M) and MAC(k2, Enc(K_1, M))
  - Integrity? Yes, attacker can't tamper with the MAC
  - Confidentiality? Yes, the MAC might leak info about the ciphertext, but that's okay
- Idea 2: Let's encrypt the MAC too: Enc(K_1, M || MAC(K_2, M))
  - Integrity? Yes, attacker can't tamper with the MAC
  - Confidentiality? Yes, everything is encrypted

MAC-then-Encrypt or Encrypt-then-MAC?

- Method 1: Encrypt-then-MAC
  - First compute Enc(K_1, M)
  - Then MAC the ciphertext: MAC(K_2, Enc(K_1, M))
- Method 2: MAC-then-encrypt
  - First compute MAC(K_2, M)
  - Then encrypt the message and the MAC together: Enc(k_1, M || MAC(K_2, M))
- Which is better?
  - In theory, both are secure if applied properly
  - MAC-then-encrypt has a flaw: You don't know if tampering has occurred until after decrypting
    - Attacker can supply arbitrary tampered input, and you always have to decrypt it
    - Passing attacker-chosen input through the decryption function can cause side-channel leaks
- **Always use encrypt-then-MAC** because it's more robust to mistakes

TLS 1.0 “Lucky 13” Attack

- TLS: A protocol for sending encrypted and authenticated messages over the Internet
- TLS 1.0 uses MAC-then-encrypt: $Enc(k_1, M \,||\, MAC(k_2, M))$
  - The encryption algorithm is AES-CBC
- The Lucky 13 attack abuses MAC-then-encrypt to read encrypted messages
  - Guess a byte of plaintext and change the ciphertext accordingly
  - The MAC will error, but the time it takes to error is different depending on if the guess is correct
  - Attacker measures how long it takes to error in order to learn information about plaintext
  - TLS will send the message again if the MAC errors, so the attacker can guess repeatedly
- Takeaways
  - Side channel attack: The algorithm is proved secure, but poor implementation made it vulnerable
  - Always encrypt-then-MAC
- https://medium.com/@c0D3M/lucky-13-attack-explained-dd9a9fd42fa6