Integrity

I’d swear my life on it

Secrecy Is Not Enough

- Encryption protects someone from reading plaintext
- An adversary can still modify messages
  - Flip a bit in the plaintext
  - Lead you to accept garbage data
- Integrity: protecting messages from tampering and modification
  - Node actually advertised that routing vector
  - Person actually made that bid
  - Endpoint actually sent that message to terminate the connection
- Confidentiality without integrity is rare (and a sign of a poor design), while integrity without confidentiality is common
  - Exception: encrypted storage (not on a network, no MitM)

Two Integrity Examples

- Cryptographic hashes
  - Way to verify that data has not been modified
  - Requires no secrets: anyone can generate one
  - Useful in data storage
- Message authentication codes (MACs)
  - Way to verify that data has not been modified
  - Also verifies generator has secret key: authenticity
  - Useful in networks

Cryptographic Hash

- Hash: computed fixed length output from arbitrary length input
  - Typical sizes 160-412 bits
  - Very cheap to compute, faster than network
- *Cryptographic* hash: also collision-resistant
  - Intractable to find $x \neq y$ such that $H(x) = H(y)$
  - Of course, many such collisions exist: ($2^{(2^{30} - 256)}$ GB blocks have same 256-bit hash)
  - But no-one has been able to find one, even after analyzing the algorithm for years
- Use SHA-256 or SHA-512 today (SHA-2)
  - Historically, most popular hash was SHA-1, but it's nearly broken
  - October 2, 2012: Keccak algorithm chosen by NIST for SHA-3
    - Goal: alternative, dissimilar hash function to SHA-2

Cryptographic Hash

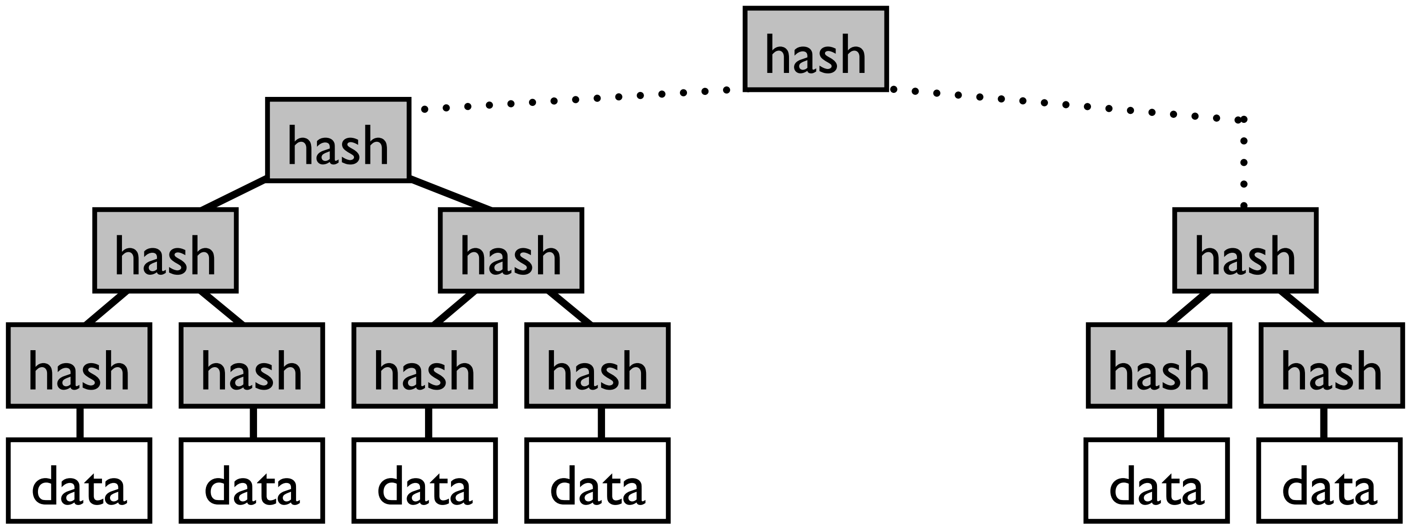
- Hash: computed fixed length output from arbitrary length input
  - Typical sizes 160-412 bits
  - Very cheap to compute, faster than network
- *Cryptographic* hash: also collision-resistant
  - Intractable to find $x \neq y$ such that $H(x) = H(y)$
  - Of course, many such collisions exist: ($2^{(2^{30} - 256)}$ GB blocks have same 256-bit hash)
  - But no-one has been able to find one, even after analyzing the a
- Use SHA-256 or SHA-512 today (SHA-2)
  - Historically, most popular hash was SHA-1, but it's nearly broken
  - October 2, 2012: Keccak algorithm chosen by NIST for SHA-3
    - Goal: alternative, dissimilar hash function to SHA-2

Designed by NSA

Designed by Bertoni et al.

Using a Cryptographic Hash

- Small hash uniquely describes large data
  - Hash a file, remember hash value h_1
  - Compute hash h_2 later on same file: if $h_1 = h_2$, file hasn't been tampered with
  - Hashes often published with a software distribution
- Hash tree (Merkle hash tree) lets you check small piece of huge data with a logarithmic number of steps



HMAC

- Anyone can generate a cryptographic hash: MACs want to also provide assurance generator has shared secret
- Simple approach: generate MAC from hash and secret,
  - HMAC(K,M) = SHA-2(K, M)
  - Send {M, HMAC(K,M)}
- Simple: we have a cryptographically strong MAC!

WRONG

- Anyon… provide
  assura…
- Simple…
  - HMA…
  - Send…

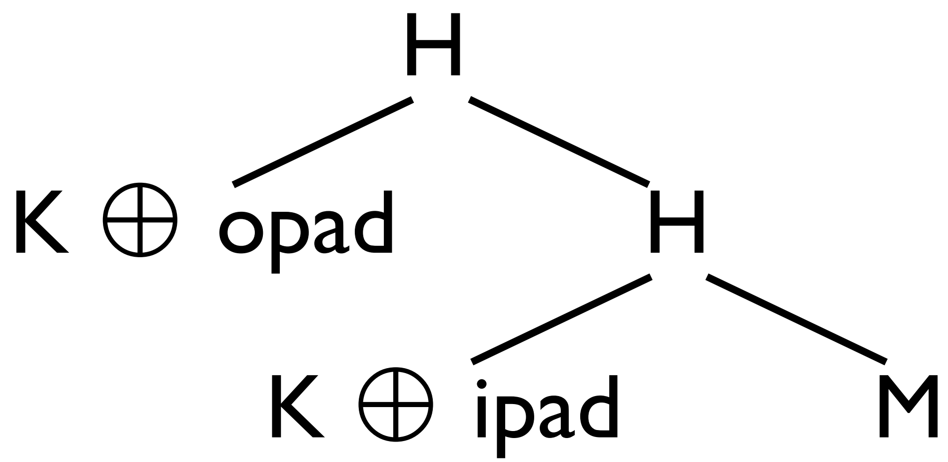
If I have {M, HMAC(K,M)},
I can generate {M', HMAC(K,M')},
where M' has data appended to M

- Simple: we have a cryptographically strong MAC!

HMAC, Revisited

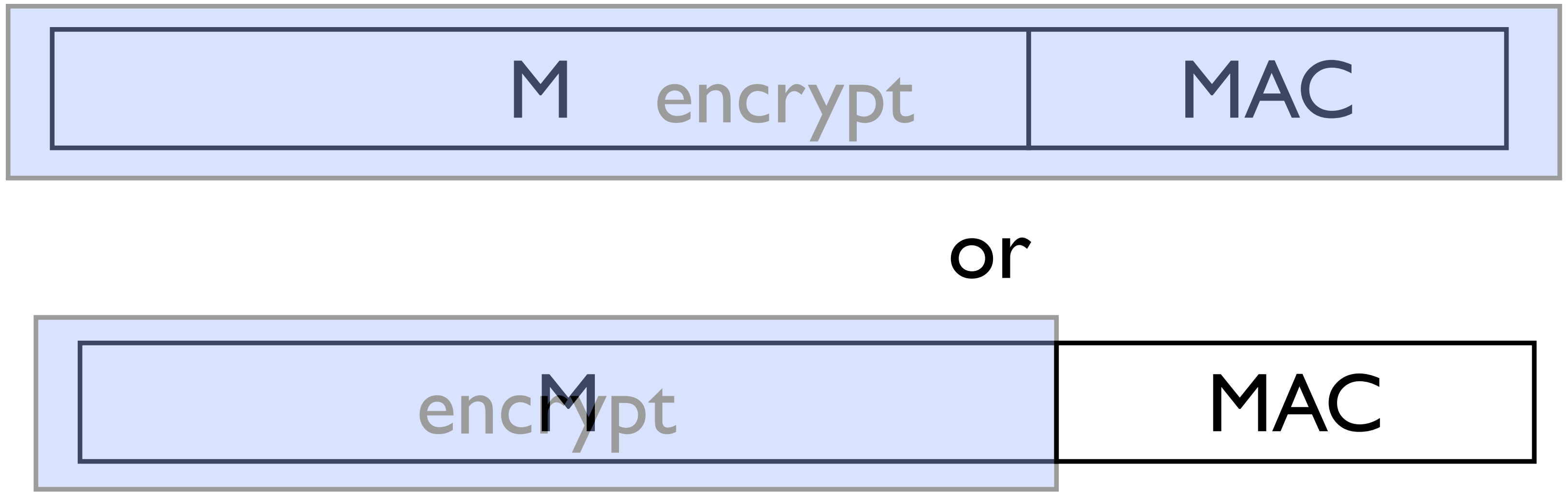
- HMAC(K,M) = H(K $\oplus$ opad, H(K $\oplus$ ipad, M))
  - H is a cryptographic hash such as SHA-3
  - ipad is 0x36 repeated 64 times, opad 0x5c repeated 64 times

- Why does previous attack not work?



Encryption and MACs

- Should I encrypt the MAC, or MAC the encrypted data?



- Encrypting the MAC is not always secure
- <u>MACing encrypted data is always secure</u>