Combating network adversaries
- Secure Channels
- Signatures
principal
(identifies client on server)

request

server

guard

resource
today we’re going to focus on how to protect packet data from an adversary. in a future lecture, we’ll talk about how you can protect meta-information (e.g., packet headers) from an adversary
**confidentiality:** adversary cannot learn message contents

**integrity:** adversary cannot tamper with message contents
  (if they do, client and/or server will detect it)
encrypt(key, message) → ciphertext
decrypt(key, ciphertext) → message

encrypt(34fbcbd1, “hello, world”) = 0x47348f63a67926cd393d4b93c58f78c
decrypt(34fbcbd1, “0x47348f63a67926cd393d4b93c58f78c”) = hello, world

**property:** given the ciphertext, it is (virtually) impossible to obtain the message without knowing the key

adversary can’t determine message, but might be able to cleverly alter ciphertext so that it decrypts to a different message
encrypt(key, message) → ciphertext
decrypt(key, ciphertext) → message

encrypt(34fbc bd1, “hello, world”) = 0x47348f63a67926cd393d4b93c58f78c
decrypt(34fbc bd1, “0x47348f63a67926cd393d4b93c58f78c”) = hello, world

**property:** given the ciphertext, it is (virtually) impossible to obtain the message without knowing the key

---

no good — if the adversary changes ciphertext, it can also (correctly) update the hash
encrypt(key, message) → ciphertext
decrypt(key, ciphertext) → message

encrypt(34fbcbd1, “hello, world”) = 0x47348f63a67926cd393d4b93c58f78c
decrypt(34fbcbd1, “0x47348f63a67926cd393d4b93c58f78c”) = hello, world

property: given the ciphertext, it is (virtually) impossible to obtain the message without knowing the key

MAC(key, message) → token

MAC(34fbcbd1, “hello, world”) = 0x59cccc95723737f777e62bc756c8da5c

property: given the message, it is (virtually) impossible to obtain the token without knowing the key

(it is also impossible to go in the reverse direction)
c = encrypt(k, m)  

h = MAC(k, m)

in practice, we’d use one key to encrypt and a different one to MAC

m = decrypt(k, c) 
MAC(k, m) == h ?
problem: replay attacks
(adversary could intercept a message, re-send it at a later time)

c = encrypt(k, m)
h = MAC(k, m)
m = decrypt(k, c)
MAC(k, m) == h?
\[c = \text{encrypt}(k, m \mid \text{seq})\]
\[h = \text{MAC}(k, m \mid \text{seq})\]

\[\text{m | seq} = \text{decrypt}(k, c)\]
\[\text{MAC}(k, \text{m | seq}) \equiv h\]
**Problem:** reflection attacks

(adversary could intercept a message, re-send it at a later time in the opposite direction)
\[ c_a = \text{encrypt}(k_a, m_a \mid \text{seq}_a) \]
\[ h_a = \text{MAC}(k_a, m_a \mid \text{seq}_a) \]
\[ m_a \mid \text{seq}_a = \text{decrypt}(k_a, c_a) \]
\[ \text{MAC}(k_a, m_a \mid \text{seq}_a) = h_a? \]

\[ c_b = \text{encrypt}(k_b, m_b \mid \text{seq}_b) \]
\[ h_b = \text{MAC}(k_b, m_b \mid \text{seq}_b) \]
\[ m_b \mid \text{seq}_b = \text{decrypt}(k_b, c_b) \]
\[ \text{MAC}(k_b, m_b \mid \text{seq}_b) = h_b? \]
**problem:** how do the parties know the keys?

**known:** $p$ (prime), $g$

**property:** given $g^r \mod p$, it is (virtually) impossible to determine $r$ even if you know $g$ and $p$

---

**alice**

pick random $a$

calculate $(g^b \mod p)^a \mod p$

key $= g^{ab} \mod p$

---

**bob**

pick random $b$

calculate $(g^a \mod p)^b \mod p$

$(g^x \mod p)^y \mod p = (g^y \mod p)^x \mod p = g^{xy} \mod p$
alice
pick random a
\[ g^a \mod p \]
\[ g^e \mod p \]
\[ k_1 = (g^e)^a \mod p \]

eve
pick random e
\[ g^b \mod p \]
\[ g^e \mod p \]
\[ k_2 = (g^e)^b \mod p \]

eve can calculate
\[ k_1 \text{ and } k_2 \]

bob
pick random b

eve can calculate
\[ k_1 \text{ and } k_2 \]

\[ \text{encrypt}(k_1, m) \rightarrow \]
\[ \text{decrypt } m \]
\[ \text{encrypt}(k_2, m) \rightarrow \]

problem: alice and bob don’t know they’re not communicating directly
cryptographic signatures
allow users to verify identities using public-key cryptography

users generate key pairs
the two keys in the pair are related mathematically

\{\text{public\_key}, \text{secret\_key}\}

\text{sign}(\text{secret\_key}, \text{message}) \rightarrow \text{sig}
\text{verify}(\text{public\_key}, \text{message}, \text{sig}) \rightarrow \text{yes/no}

\textbf{property:} it is (virtually) impossible to compute \text{sig} without \text{secret\_key}
\[ m = \text{original message} \]
\[ c = \text{encrypt}(k_a, m \mid \text{seq}_a) \]
\[ h = \text{MAC}(k_a, m \mid \text{seq}_a) \]
\[ \text{sig} = \text{sign(}\text{secret\_key}_a, m \mid \text{seq}_a) \]

\[ m \mid \text{seq}_a = \text{decrypt}(k_a, c) \]
\[ \text{MAC}(k_a, m \mid \text{seq}_a) == h \ ? \]
\[ \text{verify(}m \mid \text{seq}_a, \text{public\_key}_a, \text{sig}) == \text{yes?} \]

\textbf{verify} will not check out if \textbf{sig} was not created with \textbf{secret\_key}_a, the corresponding secret key to \textbf{public\_key}_a.
m = original message

\(c = \text{encrypt}(k_a, \ m \ | \ seq_a)\)

\(h = \text{MAC}(k_a, \ m \ | \ seq_a)\)

\(\text{sig} = \text{sign}(\text{secret	extunderscore key}_a, \ m \ | \ seq_a)\)

\(m \ | \ seq_a = \text{decrypt}(k_a, \ c)\)

\(\text{MAC}(k_a, \ m \ | \ seq_a) == h \ ?\)

\(\text{verify}(m \ | \ seq_a, \ \text{public	extunderscore key}_a, \ \text{sig}) == \text{yes} \ ?\)

how do we distribute public keys?
ClientHello \{version, seq_c, session_id, cipher suites, compression func\} 

ServerHello \{version, seq_s, session_id, cipher suite, compression func\}  
\{server certificate, CA certificates\}

ServerHelloDone

client verifies authenticity of server

ClientKeyExchange \{encrypt(server_pub_key, pre_master_secret)\}  

\begin{align*}
\text{compute} \\
\text{master_secret} &= \text{PRF}(\text{pre_master_secret}, \text{"master secret"}, seq_c | seq_s) \\
\text{key_block} &= \text{PRF}(\text{master_secret}, \text{"key expansion"}, seq_c | seq_s) \\
&= \{ \text{client_MAC_key}, \\
&\quad \text{server_MAC_key}, \\
&\quad \text{client_encrypt_key}, \\
&\quad \text{server_encrypt_key}, \\
&\quad \ldots \}
\end{align*}

Finished \{\text{sign(client_MAC_key, encrypt(client_encrypt_key, MAC(master_secret, previous_messages)))}\}

Finished \{\text{sign(server_MAC_key, encrypt(server_encrypt_key, MAC(master_secret, previous_messages)))}\}
• **Secure channels** protect us from adversaries that can observer and tamper with packets in the network.

• Encrypting with **symmetric keys** provides secrecy, and using **MACs** provides integrity. **Diffie-Hellman key exchange** lets us exchange the symmetric key securely.

• To verify identities, we use **public-key cryptography** and cryptographic **signatures**. We often distribute public keys with **certificate authorities**, though this method is not perfect.