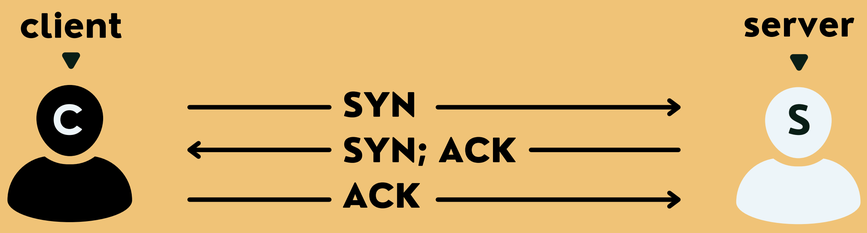
Implementing an SSH Client

Securely sending commands over an unsecure network

By Ruben Boero & Liam Keane, Advised by Jeff Ondich

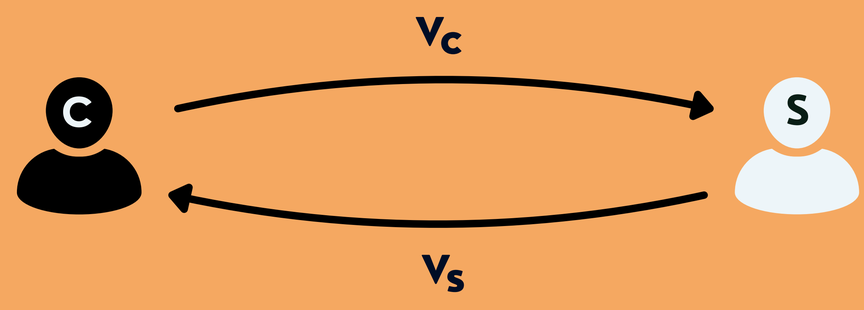
REFERENCES

1) TCP Connection



- SSH runs on top of TCP
- Enures messages arrive in correct order

2) Protocol Version Exchange



- Client and server identification string
- Clarifies the versions both parties are using
- Form: SSH-2.0-OpenSSH_9.7p1 Debian-7

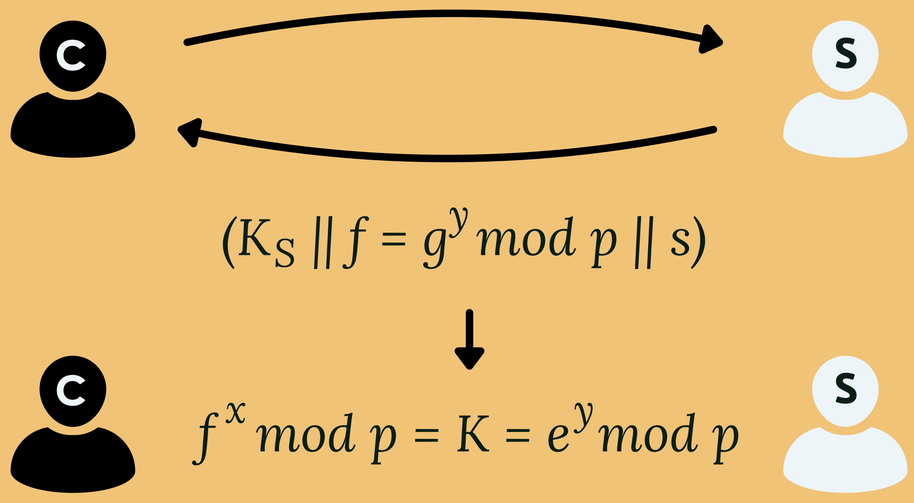
3) Algorithm Negotiation

- Key exchange: diffie-hellman-group14-sha256
- Server host key: ssh-ed25519
- Encryption algorithms: aes-128-ctr
- MAC algorithms: hmac-sha1

4) Diffie-Hellman Key Exchange

1. Derive a shared secret (K):

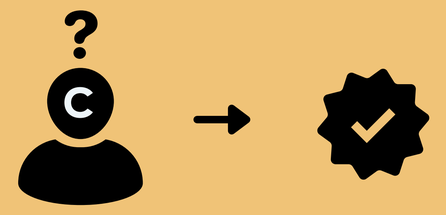
$$e = g^x \bmod p$$



2. Verify server legitimacy:

Does

$unsign(s, K_S)$



Equal

$$H = hash(V_C \| V_S \| I_C \| I_S \| K_S \| e \| f \| K)$$

5) Encryption, Integrity, and Authenticity

Symmetric encryption using Diffie-Hellman shared secret and hash from previous step

$$K_E = hash(K \| H \| \text{“}x\text{”} \| sessionID)$$

$$IV = hash(K \| H \| \text{“}x\text{”} \| sessionID)$$

$$ciphertext = E(K_E, IV, plaintext)$$

$$plaintext = E(K_E, IV, ciphertext)$$

Message authentication code (MAC) provides authenticity of sender and integrity of message

$$MAC = mac(K, seq\text{-}num \| plaintext)$$

$$packet = (ciphertext \| MAC)$$

6) Secure Communication

1. Authenticate client
2. Open session channel
3. Request command execution
4. Receive server's output