CS 419: Computer Security

Week 3: Recitation

Transport Layer Security (TLS)

Key concepts we covered

- **Symmetric Cryptography**
  - Encrypt and decrypt data using a shared secret key
  - **Example algorithms: AES, ChaCha20**
- **Public Key Cryptography (Asymmetric Cryptography)**
  - Two related keys: public & private
  - Encrypt with a public key; decrypt with the private key
  - Encrypt with your private key; decrypt with a public key
  - **Example algorithms: RSA, Elliptic Curve Cryptography**
- **Hash function**
  - Maps input data to a fixed-length digest in a way that is hard to reverse
  - **Example algorithms: SHA-2 (SHA-256, SHA-512, ...), SHA-3**

Key concepts we covered

- **Message Authentication Code (MAC)**
  - A hash of a {message + secret key}
  - Only someone who knows the secret can create or verify the MAC
  - **Example algorithms: HMAC, CBC-MAC**
- **Digital Signature**
  - A hash encrypted with a private key
  - Only the owner of the private key can create this
  - Anybody can verify it with the user's public key.
  - **Example algorithms: SHA-2+RSA, SHA-2+ECC, SHA-2+DSA, …**
- **Diffie-Hellman Key Exchange (DHKE)**
  - A special-purpose public key algorithm that allows two parties to compute a common key
  - Very quick for key generation.

Key concepts we covered

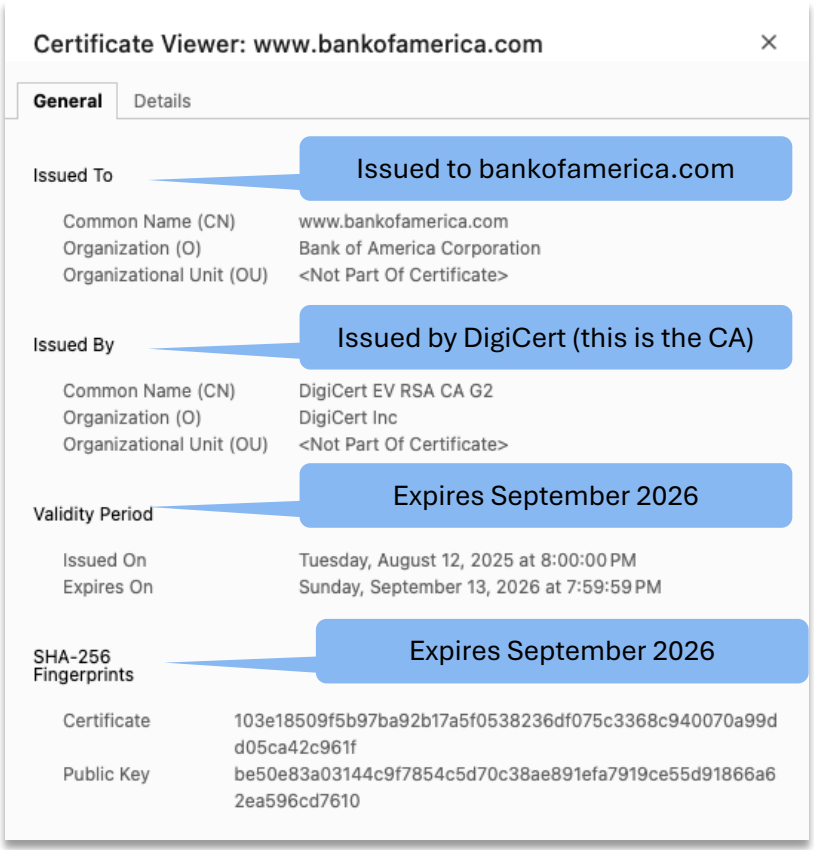
- **Forward Secrecy**
  - There is no secret that someone can steal to allow them to decrypt all your past communication sessions
  - Accomplished by creating unique Diffie-Hellman public keys for each session and then using those to derive any keys we need to communicate
- **Categories of keys**
  - **Long-term key**: public-private keys – usually bound to an identity and used for a long time
  - **Session key**: symmetric key used to encrypt data for one communication session
  - **Ephemeral key**: very short-term key used at the start of a session to set up the session key(s)
- **Digital Certificate (X.509 Certificate)**
  - A standard for storing an identity and its public key
  - Contains a digital signature of the company that certified the identity

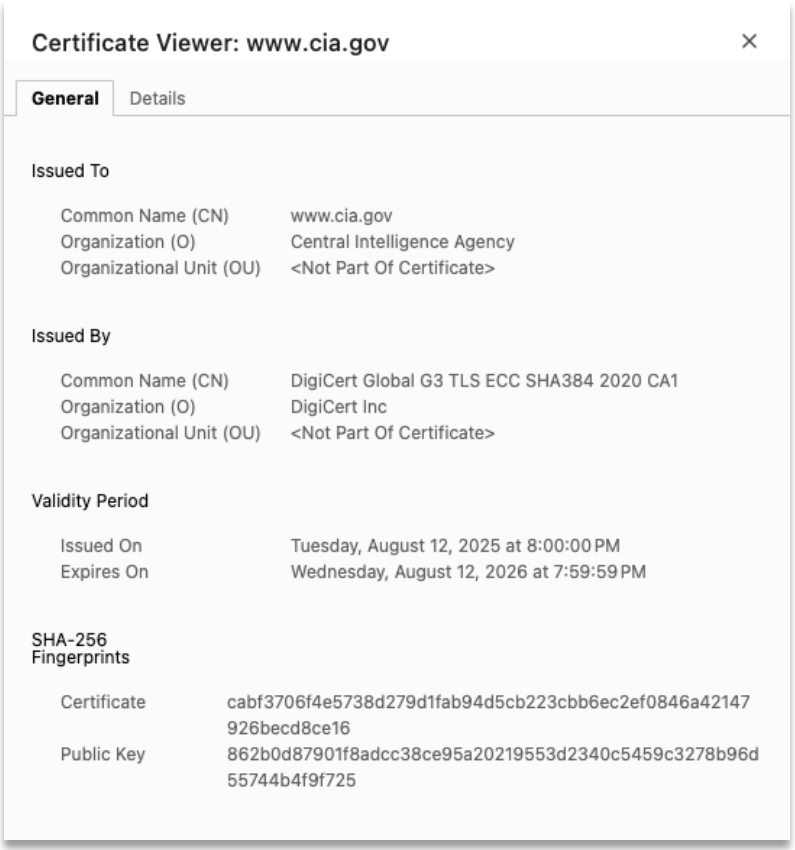
X.509 Certificates (Digital Certificates)

If you get Alice's public key, how do you know it belongs to Alice?

- **An X.509 certificate contains an identity and the public key**
- **It contains:**
  1. **Subject**: who the certificate is issued to (Alice, bankofamerica.com, etc.)
  2. **Public key**: The subject's public key that can be used to
  3. **Issuer**: the Certificate Authority (CA) that is responsible for verifying the identity of the subject and creating the certificate
  4. **Validity period**: start & expiration dates of the certificate
  5. **Digital signature**: the CA's signature to validate that the certificate data isn't modified

Example Certificates (via the Chrome browser)





Secure Communication with TLS

Why Protocols Matter

- **We know know how to do**
  - Public key cryptography
  - Digital signatures
  - Hashes
  - Symmetric cryptography
- **How do two strangers really set up a secure connection?**

Transport Layer Security (TLS)

Goal: provide a *transport layer* security protocol

After setup, applications feel like they are using TCP sockets

- **TLS was originally called SSL (Secure Socket Layer)**
- **It was designed with the web (HTTP) in mind but is used for HTTPS, email, VPNs, and other protocols**

Protocols

- **Protocols are instructions that describe a sequence of operations**
- **Each step has a purpose:**
  - Authenticate, agree on keys, prevent tampering, …

We will look at the latest version of TLS (1.3) –
the high-level steps but not every detail

The basics of setting up a secure channel

The key steps in setting up a secure communication channel:

- **Authentication**: the client knows it's talking to the real server
- **Confidentiality**: traffic is encrypted with symmetric keys
- **Integrity**: traffic cannot be tampered with silently
- **Forward secrecy** (optional goal): old traffic remains safe even if long-term keys are later stolen

Useful components

- **TLS uses:**
  - **Diffie-Hellman**: to generate an initial shared secret
  - **Digital certificates**: for the server to prove its identity (so you know you're connecting to bankofamerica.com)
  - **Public key cryptography**: to validate the certificate
  - **HMAC**: hash-based MAC for message integrity to prove we know the shared secret
  - **Symmetric cryptography**: to encrypt data going back and forth
- **It also relies on:**
  - **AEAD** (Authenticated Encryption with Associated Data)
  - **HKDF** (HMAC-based Key Derivation Function)

AEAD: Combine Confidentiality + Integrity

AEAD: Authenticated Encryption with Associated Data = encryption + MAC

- Algorithm that combines encryption with message authentication
- Avoids having to scan the message twice: once for encrypting and again for hashing
- **Examples**
  - AES-GCM (AES – Galois Counter Mode), ChaCha20-Poly1305
    - Both produce a 16-byte authentication tag in addition to the encrypted message

TLS 1.3 Key Derivation

- **Both sides have a common key after the handshake**
  - Use that to create all the keys we need – client and server can derive the same sets
- **HKDF - HMAC-based Extract-and-Expand Key Derivation Function (RFC5869)**
  - Specification to create any # of keys starting from one secret key
  - It uses an initialization function and then HMAC hashing to create additional keys
- **Key Derivation Function**
  - **Initialization:** Extracts a fixed-length pseudorandom key, PRK, from the initial secret:

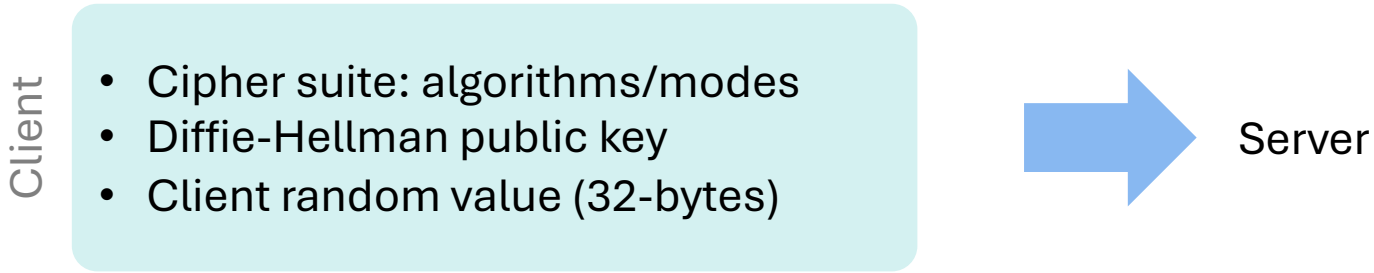
    $PRK = hash(non\text{-}secret\text{-}salt, key)$
  - **Key derivation**: Expands *K* into any number of additional keys

    $Key_0 = null$

    $Key_n = HMAC(PRK, Key_{n-1}, n)$

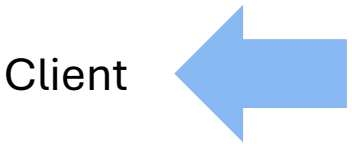
TLS 1.3 Handshake Walkthrough

Step 1. ClientHello



- **The client connects to the server and sends:**
  - A list of cipher and key exchange algorithms that it supports
  - An ephemeral Diffie-Hellman public key that it just created
  - A 32-byte random value
    - This ensures every session is unique, even if the client & server talk twice
    - It will be mixed into the key derivation process for the session key

Step 2. ServerHello

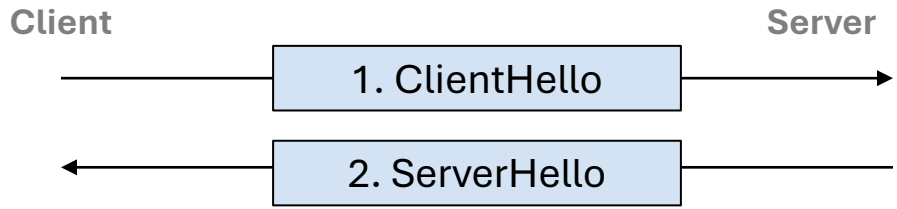


- Choice of cipher suite
- Diffie-Hellman public key
- Server random value

Server

- **The server responds:**
  - The chosen set of algorithms from the list sent by the client
  - An ephemeral Diffie-Hellman public key that it just created
  - A 32-byte random value created by the server
    - Both the client & server random values will be used as input to the key generator
    - This ensures that keys will be different even if the same Diffie-Hellman keys were accidentally used

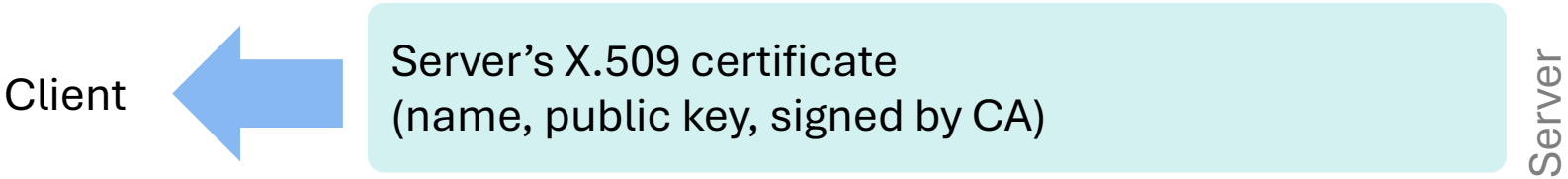
After Step 2



- **Each side now has the other side's Diffie-Hellman public key**
  - Compute the shared Diffie-Hellman secret (common key)
- **Using a key derivation function (HKDF), both sides combine:**
  - The DH secret
  - The client random
  - The server random
  - **handshake transcript:** a running hash of all handshake messages

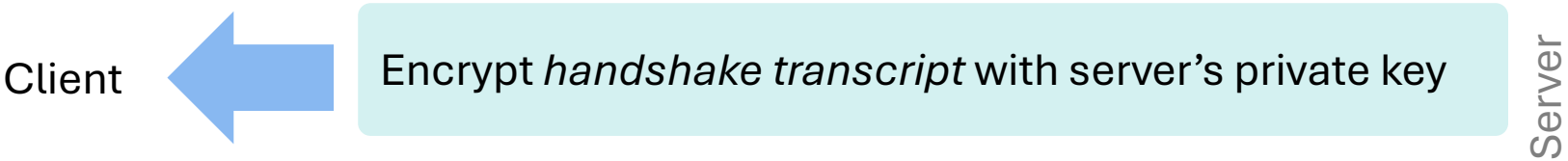
The key derivation function is a pseudorandom number generator that can generate as many keys as you need

Step 3. Certificate



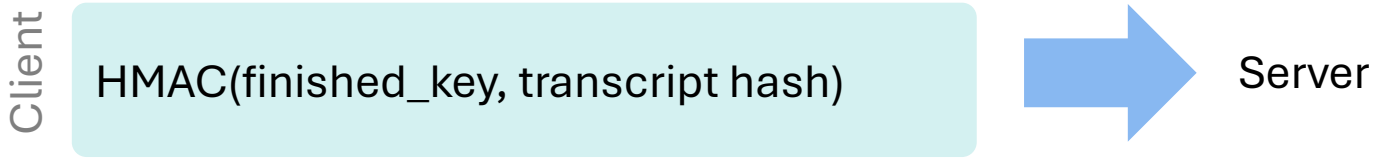
- **The server sends its digital certificate**
  - This contains:
    - The server's identity (e.g., the domain name that the client can compare)
    - The server's public key (this is the server's **long-term key**)
    - The signature of the certificate authority (CA) that will allow the client to verify the certificate

Step 4. Certificate Verify



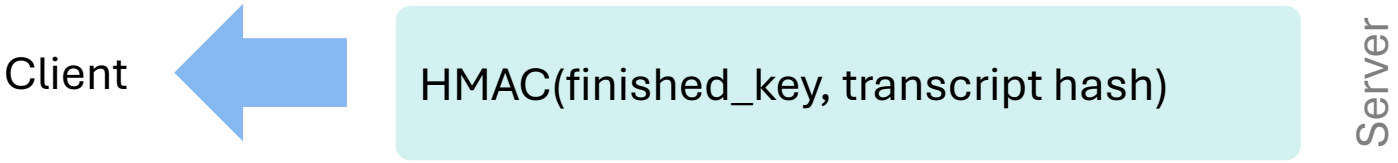
- **The server proves that it has the private key for the public key that was in the certificate it sent**
  - The handshake transcript is a hash of all the messages sent and received so far
  - Server signs the handshake transcript with its private key
    (encrypts the hash with whatever algorithm its private key is associated with)
  - Prevents an attacker from replaying or combining messages
    - The signature is a function of all messages sent so far

Step 5. Finished (Client)



- **The client sends a Finished MAC**
  - HMAC algorithm is used = SHA-256 or SHA-384 hash = hash(key, data)
  - Input to the hash is:
    - Transcript hash = hash of all handshake messages up to this point
    - Finished key = from the HKDF – derived from the shared secret

Step 6. Finished (Server)

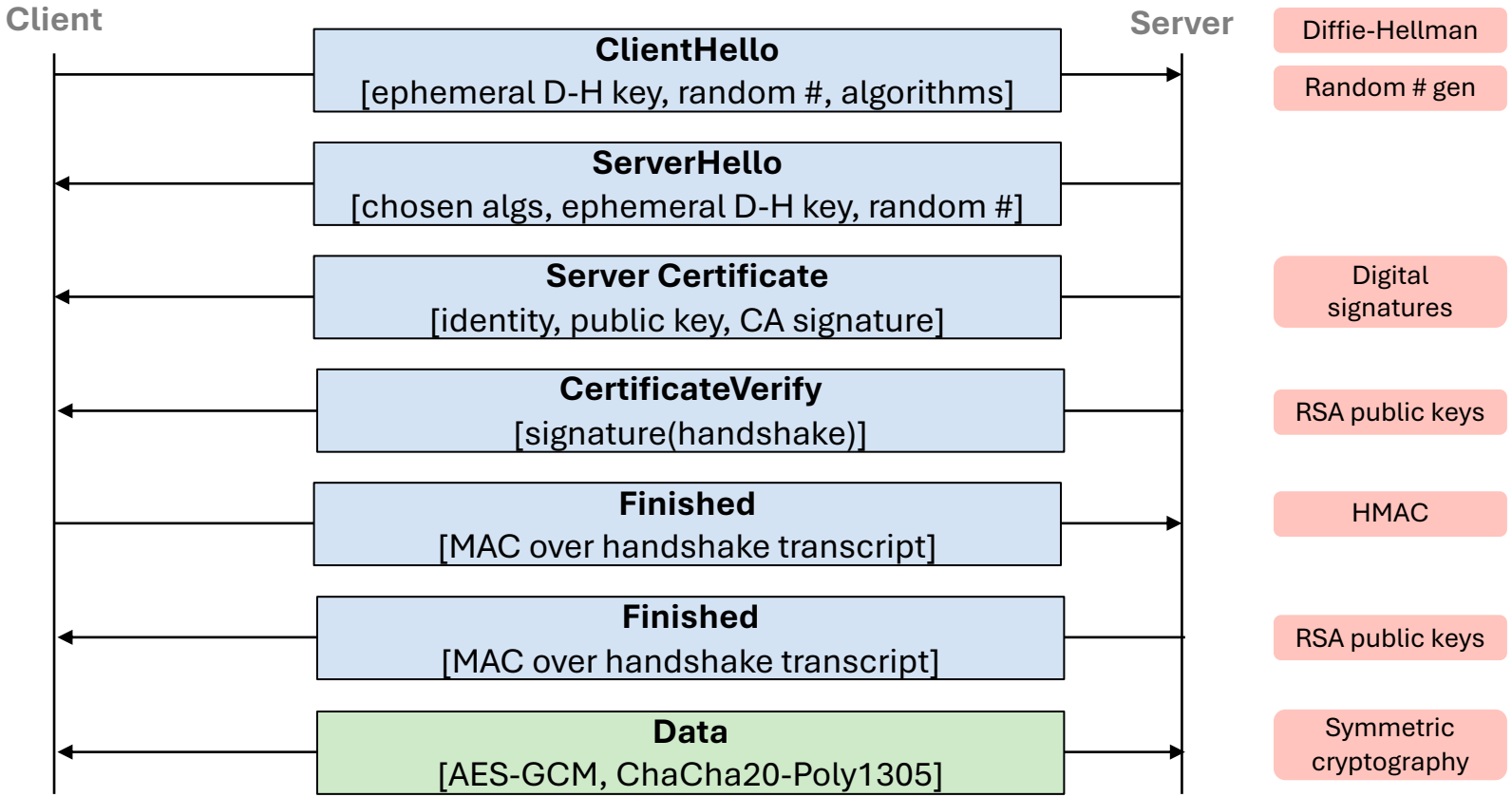


- **The server verifies the client's *Finished* message**
  - Computes an HMAC(finished_key, transcript_hash)
  - It uses the same key derivation function (HKDF) and can generate the same key
  - It saw the same messages, so it has the same transcript_hash
  - If the HMAC matches then the server is convinced that:
    - This message is part of the session
    - The client knows the shared secret (that was used to set up HKDF)
- **The server computes the Finished HMAC and sends it to the client**
  - The HMAC input is different because the transcript hash has been updated

After Step 6. Application Data

- **The client verifies the Finished HMAC it received from the server**
- **Both sides know:**
  - The handshake has not been tampered with
  - Both sides have the same cryptographic state (same initial key)
- **Now we're ready to send data back and forth!**
- **Both sides switch to symmetric encryption**
  - Each side uses HKDF to get two different session keys:
    1. A key for client-to-server encryption
    2. A key for server-to-client encryption

TLS 1.3 Protocol Summary



Summary

- **There are more details to TLS**
  - Different cipher suites might use HMAC instead of AEAD
  - There's a protocol for restarting broken connections efficiently
  - The protocol can request clients and server to update keys during the session
- **This protocol combines many of the components we studied**
  - Symmetric cryptography
  - Public key cryptography
  - Hash functions
  - Digital signatures
  - Message authentication codes
  - Random number generation
  - Diffie-Hellman key exchange

The End