TLS AND PKI

CS 361S

Fall 2021

Seth James Nielson

SECURE AUTHENTICATED CHANNEL

(Entity) Authentication –
Assurance of party identity

Confidentiality –
Unreadability of data by unauthorized parties

(Data Origin) Authentication
– Unwritability of data by unauthorized parties

TLS 1.2 HANDSHAKE

Goal — Agree on a cipher suite for encryption, authentication, etc.

Goal — Identify the server (and optionally the client)

Goal — Create session keys for bi-directional communication

TLS 1.2 HANDSHAKE

Step	Client	Direction	Message	Direction	Server
1			Client Hello	>	
2		<	Server Hello		
3		<	Certificate		
4		<	Server Key Exchange		
5		<	Server Hello Done		
6			Client Key Exchange	>	
7			Change Cipher Spec	>	
8			Finished	>	
9		<	Change Cipher Spec		
10		<	Finished		

HANDSHAKE VISUALIZATION 1

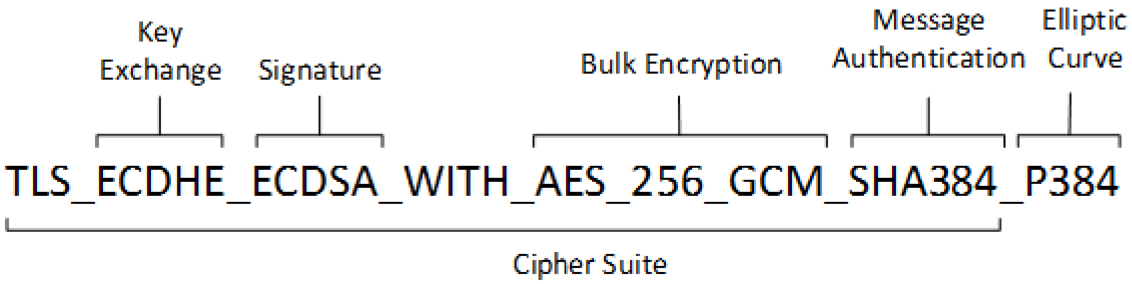
CLIENT HELLO

```
struct {
    ProtocolVersion client_version;
    Random random;
    SessionID session_id;
    CipherSuite cipher_suites<2..2^16-2>;
    CompressionMethod compression_methods<1..2^8-1>;
    select (extensions_present) {
        case false:
            struct {};
        case true:
            Extension extensions<0..2^16-1>;
    };
} ClientHello;
```

```
struct {
    uint32 gmt_unix_time;
    opaque random_bytes[28];
} Random;
```

client_version: 0x303 for TLS 1.2
random: prevents "replay" attacks
cipher_suites: see next slide

CIPHER SUITES



- TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256
- TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384
- TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256
- TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384
- TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256
- TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384
- TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
- TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
- TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256
- TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384
- TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256
- TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384
- TLS_DHE_RSA_WITH_AES_128_GCM_SHA256
- TLS_DHE_RSA_WITH_AES_256_GCM_SHA384
- TLS_DHE_RSA_WITH_AES_128_CBC_SHA

… (there are MANY more!)

SERVER HELLO

```
struct {
    ProtocolVersion server_version;
    Random random;
    SessionID session_id;
    CipherSuite cipher_suite;
    CompressionMethod compression_method;
    select (extensions_present) {
        case false:
            struct {};
        case true:
            Extension extensions<0..2^16-1>;
    };
} ServerHello;
```

Note that the client offers a list of cipher suites and the server picks one

SERVER CERTIFICATE

- A “chain” of certificates
- Usually X509 certificates
- Lowest certificate identifies the server’s name (e.g., “www.amazon.com”)
- Includes a public key such as an RSA public key
- ***The public key must be compatible with the ciphersuite***

SERVER KEY EXCHANGE

```
struct {
    select (KeyExchangeAlgorithm) {
        case dh_anon:
            ServerDHParams params;
        case dhe_dss:
        case dhe_rsa:
            ServerDHParams params;
            digitally-signed struct {
                opaque client_random[32];
                opaque server_random[32];
                ServerDHParams params;
            } signed_params;
        case rsa:
        case dh_dss:
        case dh_rsa:
            struct {} ;
           /* message is omitted for rsa, dh_dss, and dh_rsa */
        /* may be extended, e.g., for ECDH -- see [TLSECC] */
    };
} ServerKeyExchange;
```

- If RSA key transport is used, this message is not sent
- If DHE key agreement is used, this message sends the DHE public key
- **Notice the signature…**

SERVER HELLO DONE

- Server Hello Done carries no extra information
- Marks the end of the Hello part

THE PROBLEM OF TRUST

- How does the browser know the received public key is the server's
- What if BadGuy.com transmitted HIS key and CLAIMED it was Amazon's?
- X509 Certificates bind identity information to a key; this helps
- But it's not enough. BadGuy.com can generate a fake cert
- PKI, Public Key Infrastructure, binds ***all keys to already trusted keys***
- Client validates the chain of server certificates to a common key

CERTIFICATE VERIFICATION

WEB BROWSER

Verify "amazon.com" is the URL
Verify the validity period
(Other Verification)
Who issued the cert?

CERTIFICATE

Subject CN:	**amazon.com**
Not Valid Before:	**2001**
Not Valid After:	**2030**
Issued By:	**amazon CA**
Signature Blob:	**<sig>**

CERTIFICATE CHAINS

The certificate for the Host may be signed by an INTERMEDIATE Certificate Authority

Because the web browser probably doesn't have this intermediate cert, the TLS handshake includes both certificates.

Subject CN:	**amazon CA**
…	
Issued By:	**GlobalSign**
Signature Blob:	**<sig>**

Subject CN:	**amazon.com**
…	
Issued By:	**amazon CA**
Signature Blob:	**<sig>**

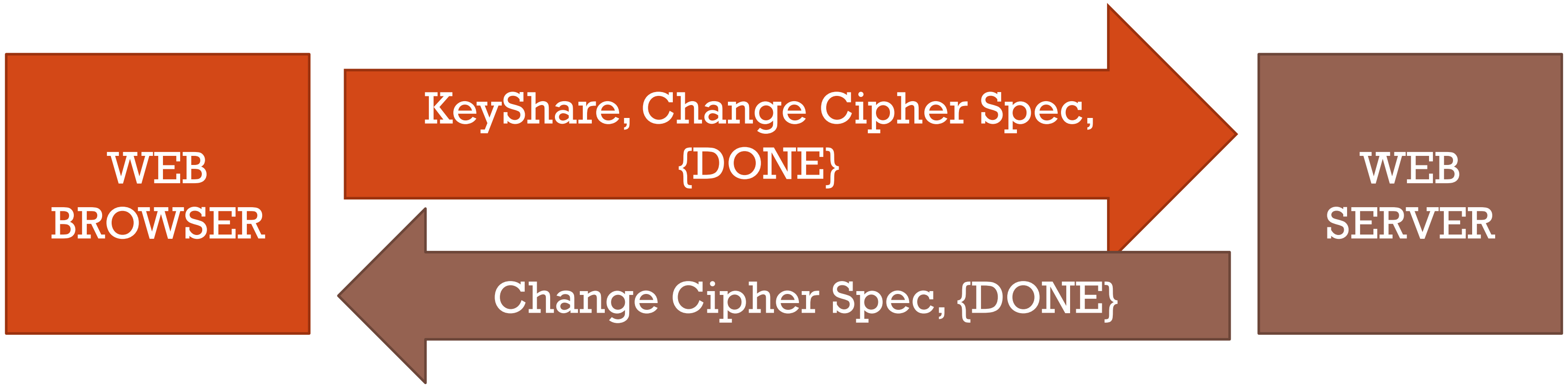
ROOT CA CERTIFICATES

- Certificate chains MUST have a ROOT
- A Root Certificate is SELF SIGNED
- Browsers trust a set of root certificates AXIOMATICALLY
- Certificate chains must have a trust chain to one of these roots.

TRUSTING DIFFIE HELLMAN

- Recall that DH keys are EPHEMERAL
- The Server's cert includes a long-term public key
- The Server's DH key is signed by this key pair
- IF the client trusts the cert, THEN it can validate the DH key

END-TO-END HANDSHAKE VISUALIZATION #2



CLIENT KEY EXCHANGE

- If RSA ***key transport***, sends a "pre master secret" encrypted under the server's RSA public key from the server's certificate
- But, for DHE ***key agreement***, sends DHE public key. "pre master secret" computed

```
struct {
    select (KeyExchangeAlgorithm) {
        case rsa:
            EncryptedPreMasterSecret;
        case dhe_dss:
        case dhe_rsa:
        case dh_dss:
        case dh_rsa:
        case dh_anon:
            ClientDiffieHellmanPublic;
    } exchange_keys;
} ClientKeyExchange;
```

DERIVING KEYS

- For bi-directional communication, EACH SIDE needs its own keys
- Step 1: Compute a master secret from a pre-master secret
- Step 2: Compute a key expansion on the master secret
- Step 3: Split up the key expansion block into the session keys

CIPHER SUITES AND KEY DERIVIATON

- The side of each key depends on the algorithm
- Some cipher suites don't need IV's; some don't need MAC's
- PLEASE NOTE: a client ***WRITE*** key is a server ***READ*** key

GENERATING THE KEY BLOCK

DHE → Pre-master Secret

PRF(pms, “master secret”, client_random, server_random) → Master Secret

PRF(ms, “key expansion”, server_random, client_random) → Key Block

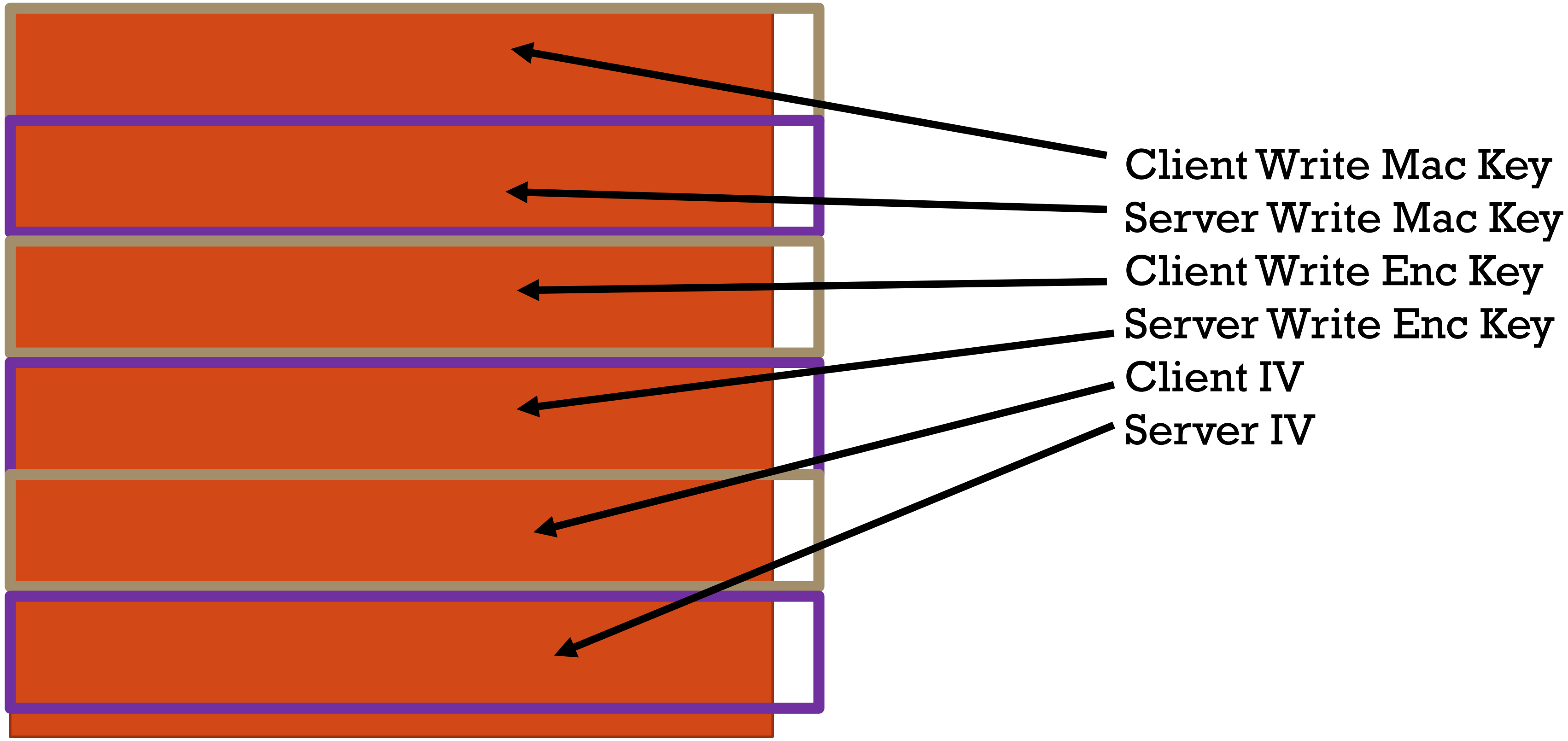
DATA ORIGIN AUTHENTICATION

- After entity authentication, ensure data received is from that party
- Ensure that data is unaltered
- Can use something like MAC – Message Authentication Code

HMAC

- Hash Message Authentication Code
- Details are outside this lecture
- Basic concept:
  - Hash + key
  - E.g., h(key + data)

SPLITTING KEY BLOCK INTO KEYS



CHANGE CIPHER SPEC

- Sent by the client after Client Key Exchange
- Indicates that all future messages will be encrypted and MAC'd

CLIENT FINISHED

- Includes a hash of all handshake messages sent so far
- Excludes Change Cipher Spec, which ***is not a handshake message***
- Excludes the current message (client finished)
- Hash is computed as:
  - PRF(master_secret, "client finished", Hash(handshake_messages))

SERVER CHANGE CIPHER SPEC

- Server verifies the client’s finished message
- Remember, this message is AFTER change cipher spec, so encrypted
- Server sends its own change cipher spec

SERVER FINISHED

- Server sends its own encrypted Finished message
- Hash of handshake messages includes client's finished message
  - PRF(master_secret, "server finished", Hash(handshake_messages))

IT ALL DEPENDS ON THE CERT

IF a browser trusts MY certificate to be Amazon's certificate	IF the browser trusts my DH public key	IF the browser derives the same MAC key I do
• THEN the browser will trust my DH public key	• THEN the browser will derive the same MAC key I do	• THEN the browser will believe my messages are from Amazon

TLS VISIBILITY

- Typically, a browser/client MUST have a new root CA installed
- This root CA is a self-signed certificate from the Visibility appliance
- The appliance can now generate ANY cert and the browser believes it!
- We will discuss the huge security concerns in a later lecture

TLS VISIBILITY HANDSHAKE VISUALIZATION