Asymmetric Cryptography

UT CS361S

FALL 2020

LECTURE NOTES

Technology Review

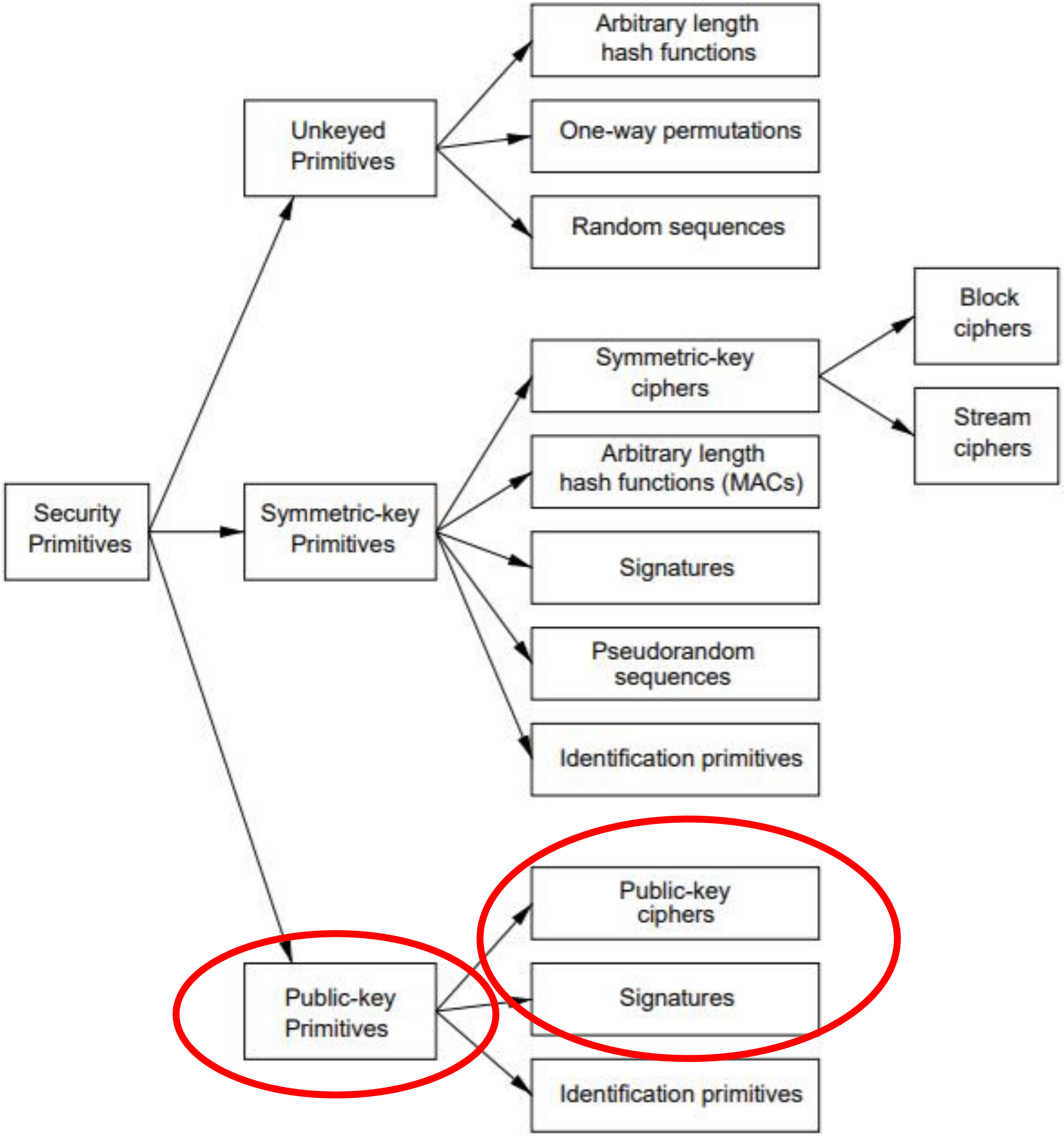


Figure 1.1: *A taxonomy of cryptographic primitives.*

Asymmetric Cryptography

- Keys come in pairs
- Public key can be shared
- Private key MUST be kept secret

Uses of Asymmetric Crypto

Unlike symmetric, what you can **DO** with asymmetric depends greatly on the algorithm

- RSA – encryption (crypto dropbox), signatures
- ECDSA/DSA – signatures
- Diffie Hellman – key agreement

RSA Encryption

Encrypt SHORT MESSAGES with public key, decrypt with private key

Encrypted Communication between A and B

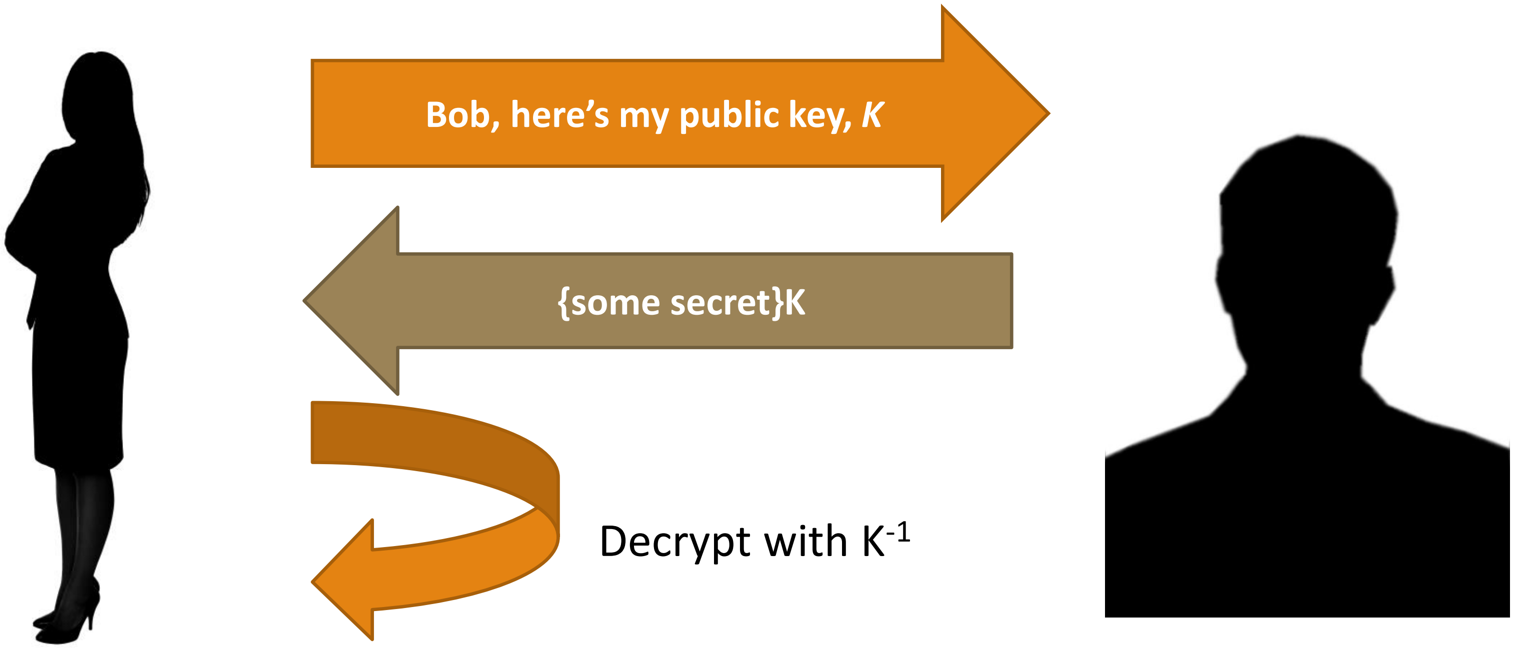
- A and B have each other's public key
- A encrypts a message for B under B's public key
- B responds by sending A a response under A's public key

Works fine but…

- It is very slow (asymmetric encryption/decryption is expensive)

Used almost exclusively for <u>Key Transfer</u> (sending a symmetric session key)

RSA Encrypt Visualization



RSA Signatures

RSA encrypts with the PRIVATE KEY for a signature

Step 1: Publisher produces a message M

Step 2: Publisher takes the hash of M h(M)

Step 3: Publisher encrypts the hash with the private key $\{h(M)\}_{k^{-1}}$

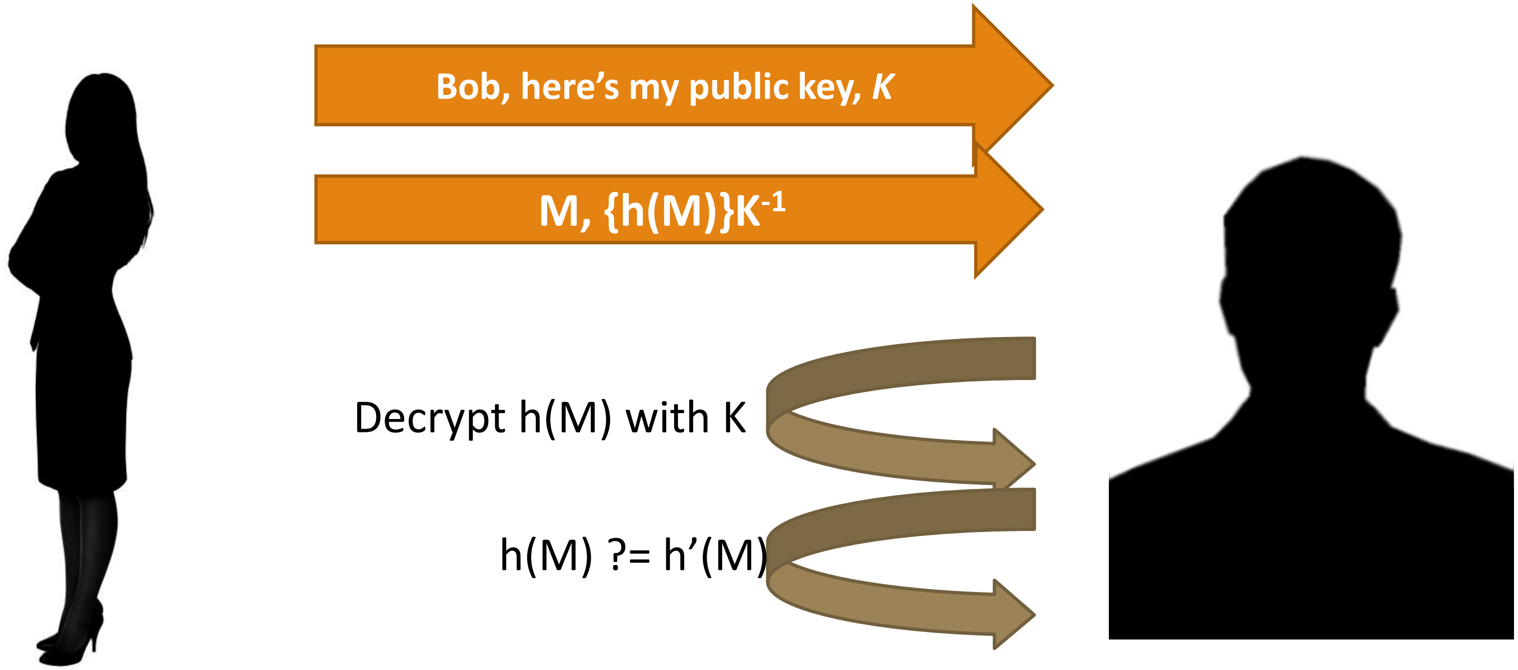
Step 4: Publisher transmits Message M and $\{h(M)\}_{k^{-1}}$ as the signature

Step 5: A verifier decrypts h(M) with Publisher's public key

Step 6. A verifier computes their own hash of M h'(M)

Step 7: A verifier determines the signature is valid if h'(M) = h(M)

RSA Signature



Key Exchange

Asymmetric crypto is not good for “bulk data” encryption

RSA can only encrypt small messages SLOWLY.

Other asymmetric algorithms CANT ENCRYPT AT ALL

So, asymmetric is used to authenticate ***KEY EXCHANGE***

There are two forms:

- Key Transfer
- Key Agreement

Key Transfer

Requires asymmetric encryption (e.g., RSA)

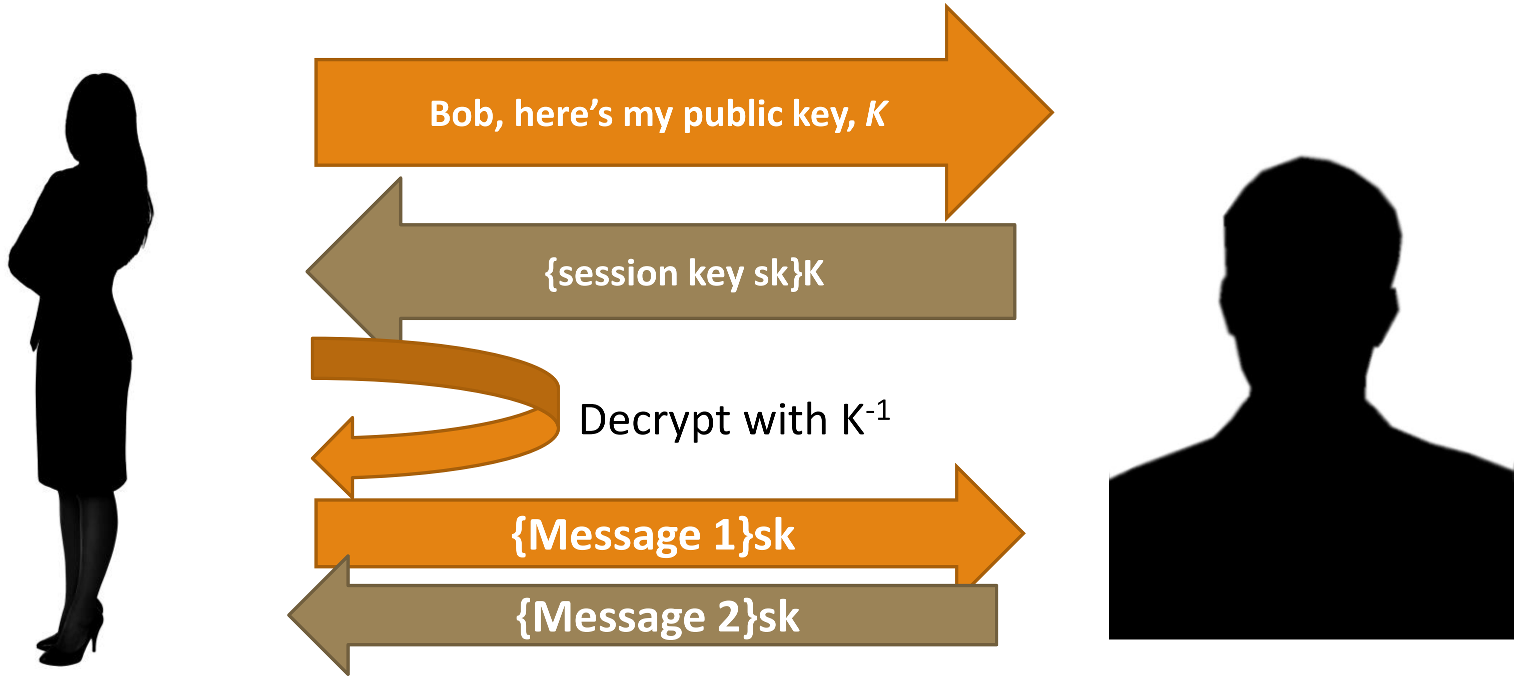
Create a session key

Send session key encrypted with public key

Only party poessing the private key can decrypt it

(Automatically authenticated)

RSA Key Transport



RSA Weaknesses

Insecure when NO PADDING IS USED

Encryption padding schemes

- PKCS 1.5 (***BROKEN!)***
- OAEP

Signature padding schemes

- PKCS 1.5 (***BROKEN!***)
- PSS

Even though there are non-broken versions, RSA is being phased out

Also, key transfer does not have “forward secrecy”

Catastrophic Loss of RSA Key

Assume A and B want to communicate, E is eavesdropping

A and B use RSA key transfer to exchange session keys

E records thousands of sessions between A and B

After 5 years, A disposes her computer and buys a new one

E steals her computer from the junkyard, finds the private key

ALL PREVIOUSLY RECORDED MESSAGES ARE EXPOSED!

Diffie Hellman Key Exchange

The math version has to do with ***commutative properties.***

Using modulo computations over *p* which is a prime with certain properties:

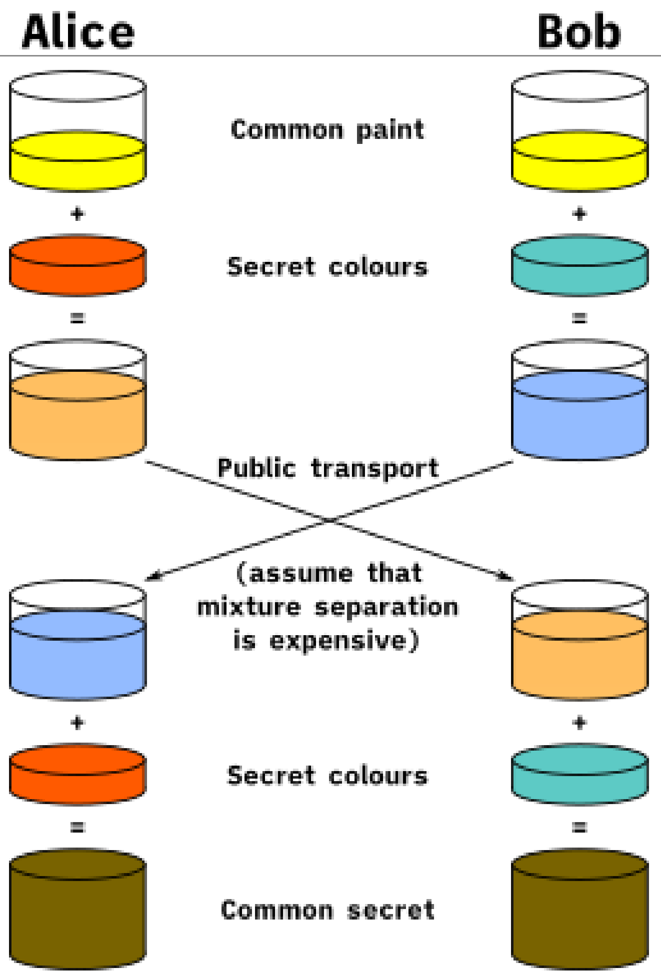
- $A \rightarrow B : g^{RA} \pmod{p}$
- $B \rightarrow A : g^{RB} \pmod{p}$
- $A \rightarrow B : \{M\}g^{RARB}$

A and B are the DH private keys

- Can't be extracted from $g^{RA} \pmod{p}$
- But, because commutative, can be combined by either side into g^{RARB}

In short, to create a key, exchange DH public keys + parameters

Wikipedia Visualization



DHE and Forward Secrecy

Diffie Hellman Ephemeral (DHE)

New Private Key used for EACH KEY AGREEMENT (session)

RSA key is used to SIGN the DH private key

DHE private key never stored outside of RAM

Now if E steals A’s computer, no messages exposed

Compromising a single key exposes only that session

This is “Forward Secrecy”

No DHE Authentication

Next class: how to prove authenticity of a public key

But, spoiler alert!, it HAS to be a long-term key

So, with DHE, you can create keys on the fly (“out of thin air”)

BUT, you have no idea who they’re coming from!!!

Two Asymmetric Steps

You caught that there were TWO asymmetric steps for DHE?

First, the DHE is used for key generation

Second, RSA is used to sign (authenticate) the DH public key

There are two asymmetric steps, algorithms, and public keys

Why not RSA Ephemeral?

Why not have a long-term RSA key for signing

And an ephemeral RSA key for each key transfer?

You could create a new RSA key pair each session, just like DH

The problem is that RSA is slow; DH keys are quickly generated

Other Asymmetric Algorithms

DSA – Just used for signing

ECDH – Elliptic Curve Diffie Hellman (just like DH)

ECDSA – Elliptic Curve DSA (just like DSA)

RSA, DH, DSA, ECDH, ECDSA are the most common I’ve seen

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

Handshake Visualization

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		

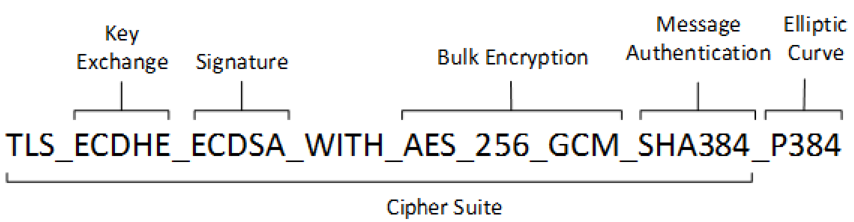
Client Hello

```
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

Certificate

We'll talk about this more next time

Really needs its own lesson

Short Version:

- Usually an X509 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

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

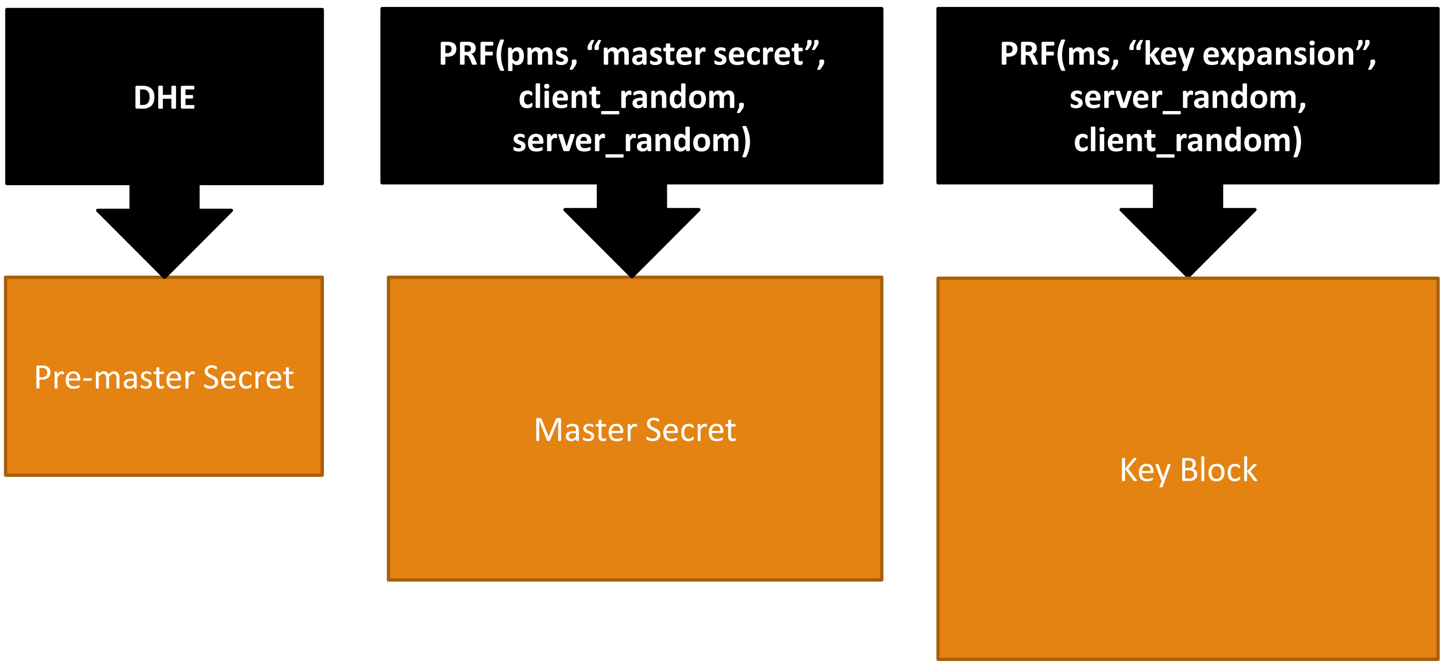
For bi-directional communication, EACH SIDE needs its own keys

Step 1: Compute a master secret from a pre-master secret

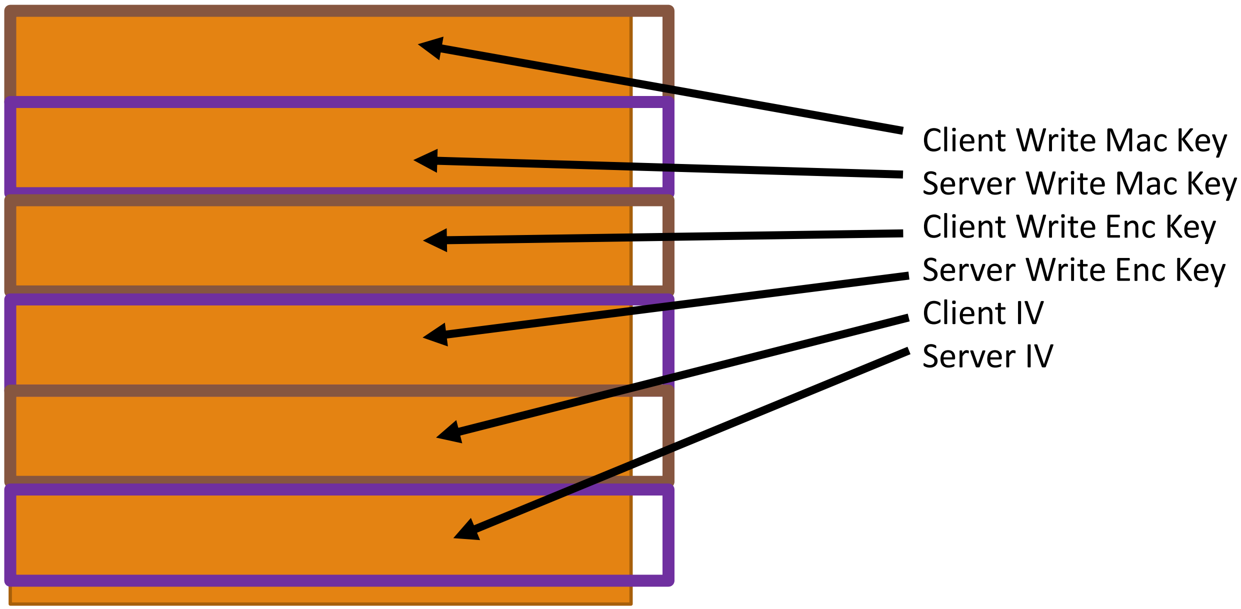
Step 2: Compute a key expansion on the master secret

Step 3: Split up the key expansion block into the session keys

Generating the Key Block



Splitting Key Block into 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

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))