

TLS AND TLS MITM

CS 361S

Spring 2020

Seth James Nielson

SECURE COMMUNICATIONS



(Entity) Authentication –
Assurance of party identity



Confidentiality –
Unreadability of data by
unauthorized parties



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

ALTERNATE TERMS

CIA – Confidentiality, Integrity, Availability

- Integrity is more-or-less equivalent to Data Origin Authentication
- Availability means data cannot be denied to authorized parties

Data origin authentication means that data came from an authorized party

Data origin authentication implies integrity

POLICY AND MECHANISM ARE DIFFERENT!

Policy	Mechanism
Confidentiality	Encryption
Entity Authenticaiton	Public Key Infrastructure (Certificates)
Data Origin Authentication	Message Authentication Code

SYMMETRIC KEY CRYPTOGRAPHY

One key both
encrypts and
decrypts data

Typically fast and
useful for bulk
encryption

Can provide
confidentiality,
but generally not
integrity

ASYMMETRIC CRYPTOGRAPHY

Algorithms use
TWO keys: one
public, one private

Different algorithms
support different
operations

- Key agreement
- Signatures
- Encryption

Many algorithms do
not support
encryption

Encryption, when
available, is slow and
for short messages

RSA



RSA public-key cryptography DOES encrypt



Public key crypto is often explained using RSA



This can be helpful because RSA is very easy to understand

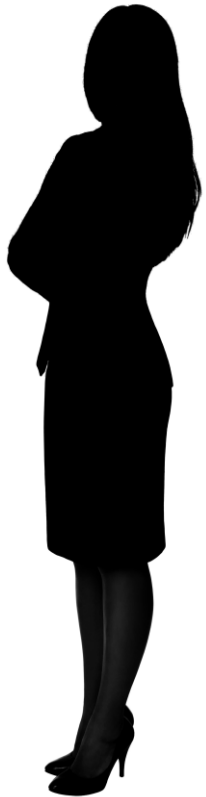


But remember, other algorithms work differently!

RSA ENCRYPTION

- RSA provides encryption of short messages
- Anyone can encrypt using the public key
- But only the owner of the private key can decrypt
- Moreover, the encrypting party knows **ONLY** the owner can decrypt
- This provides a secure, ***authenticated*** dropbox

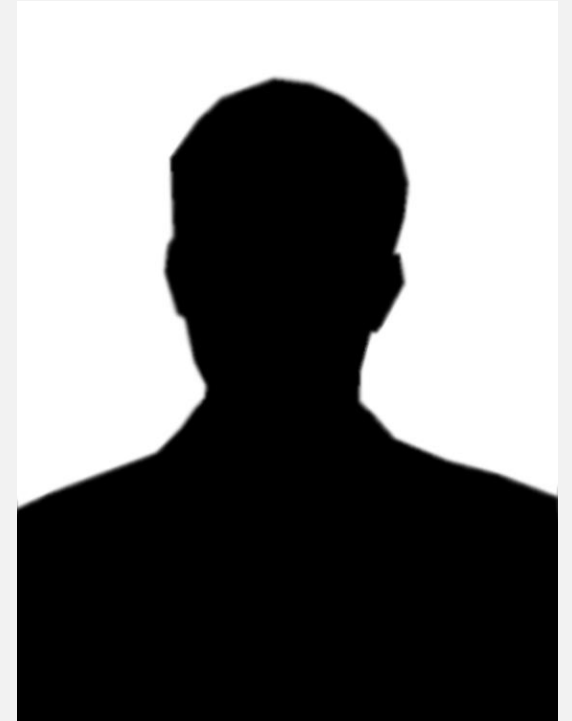
RSA ENCRYPT VISUALIZATION



Bob, here's my public key, K

{some secret} $_K$

Decrypt with K^{-1}



WHAT CAN RSA ENCRYPT?

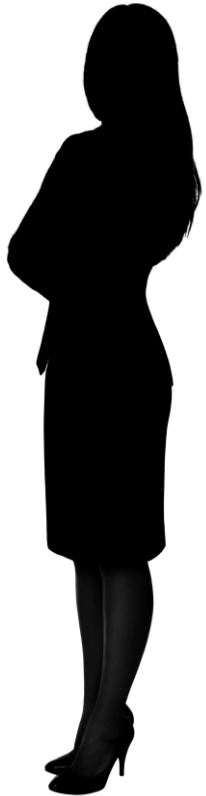
Messages must be smaller than
key size (e.g., 2048)

Hundreds of times slower than
symmetric key algorithms

So what good is it?

KEY TRANSPORT!

RSA KEY TRANSPORT



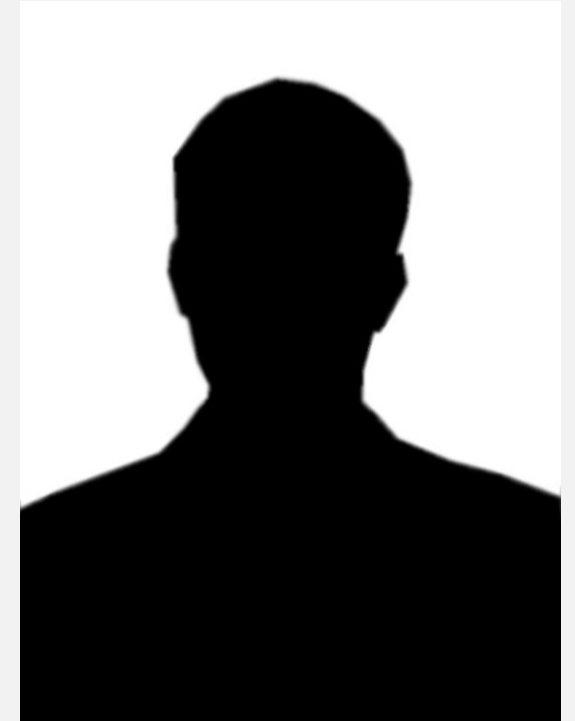
Bob, here's my public key, K

$\{\text{session key } sk\}K$

Decrypt with K^{-1}

$\{\text{Message 1}\}sk$

$\{\text{Message 2}\}sk$



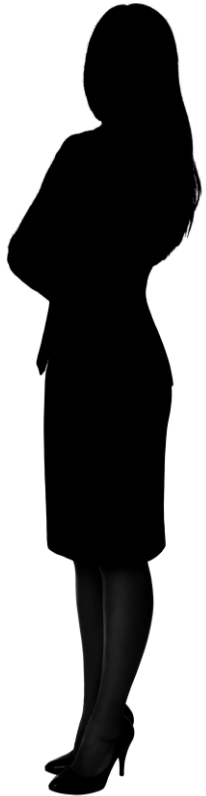
RSA SIGNATURES

- RSA signatures ALSO use encryption, but in the other direction!
- The private key owner encrypts with the private key
- Anyone can decrypt with the public key
- But it must have been encrypted by the private key owner
- This provides entity authentication, not confidentiality.

RSA SIGNATURES

- Alice publishes her public key K
- Alice wants to PROVE that she is the author of message M
- Alice produces $h(M)$, the hash of M
- Alice produces $\{h(M)\}K^{-1}$, the RSA encryption of $h(M)$
- Alice transmits M , and $\{h(M)\}K^{-1}$ to Bob.
- Bob decrypts $h(M)$ with K
- Bob compares $h(M)$ with $h'(M)$, the recomputed hash of M

RSA SIGNATURE

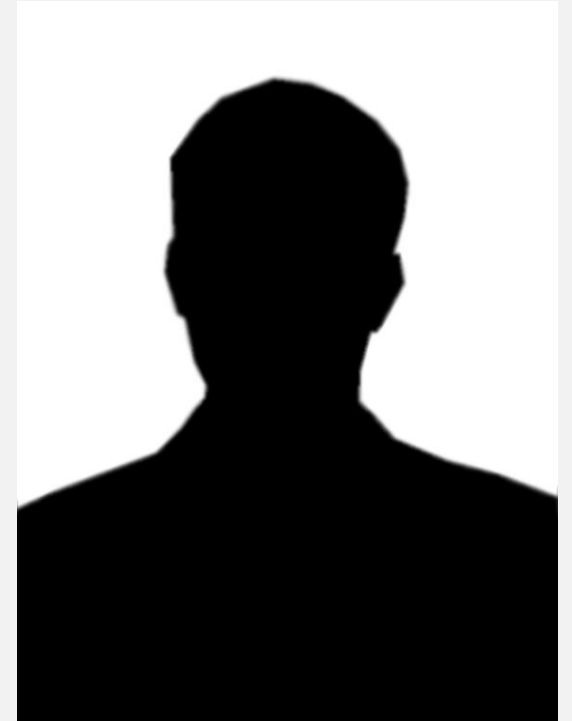


Bob, here's my public key, K

$M, \{h(M)\}K^{-1}$

Decrypt $h(M)$ with K

$h(M) \stackrel{?}{=} h'(M)$



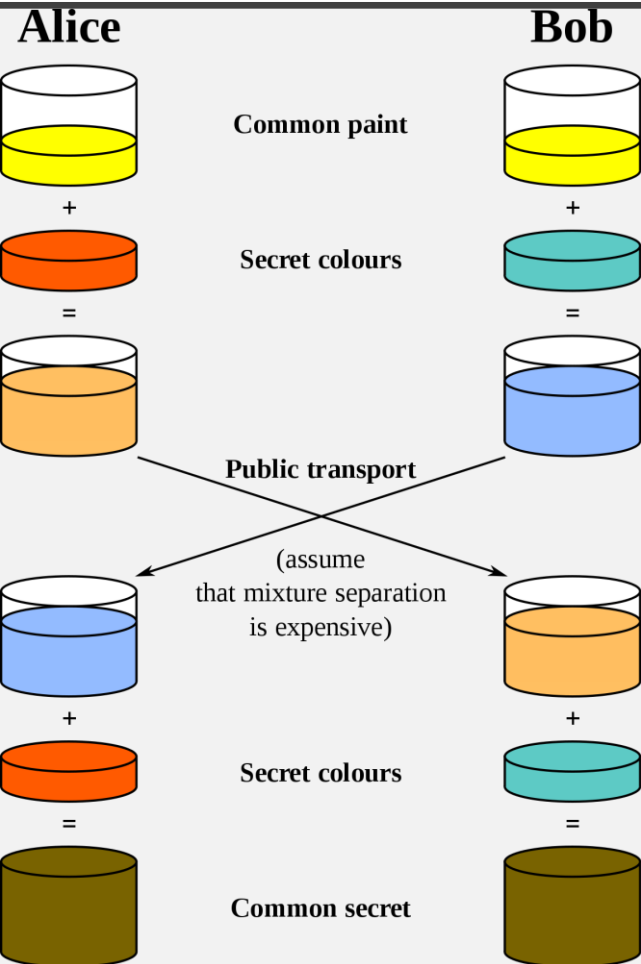
REMINDER

- Most Public Key algorithms do NOT work like RSA does
- DSA, ECDSA, for example, have signatures, but ***do not use encryption!***
- Diffie Hellman (DH), only does ***key exchange***

KEY EXCHANGE

- Key agreement is where parties create a session key between them
- RSA provides Key Transport, as already described
- Key Exchange uses public key computations to COMPUTE a shared key

DIFFIE HELLMAN VISUALIZATION



FORWARD SECRECY

- Forward secrecy = a lost private key only compromises ONE session
- Requires a unique public/private key EACH SESSION
- RSA keys are slow to generate; keys must be used long term
- DH keys are fast to generate; “ephemeral” keys can be used
- DHE = Diffie Hellman Ephemeral, provides forward secrecy

THE PROBLEM OF TRUST

- How do you know that Alice's key is Alice's?
- What if Malfoy transmitted HIS key and CLAIMED it was Alice's?
- X509 Certificates bind identity information to a key; this helps
- But it's not enough. Malfoy can generate a fake cert
- PKI, Public Key Infrastructure, binds all keys to already trusted keys

DHE + RSA

DH can quickly generate a public key, but how to trust it?

In TLS, the DH ephemeral public key is SIGNED by RSA key from cert

So, there are actually TWO sets of public/private keys

DH ephemeral keys for
key agreement

RSA key for signing the
DH ephemeral key

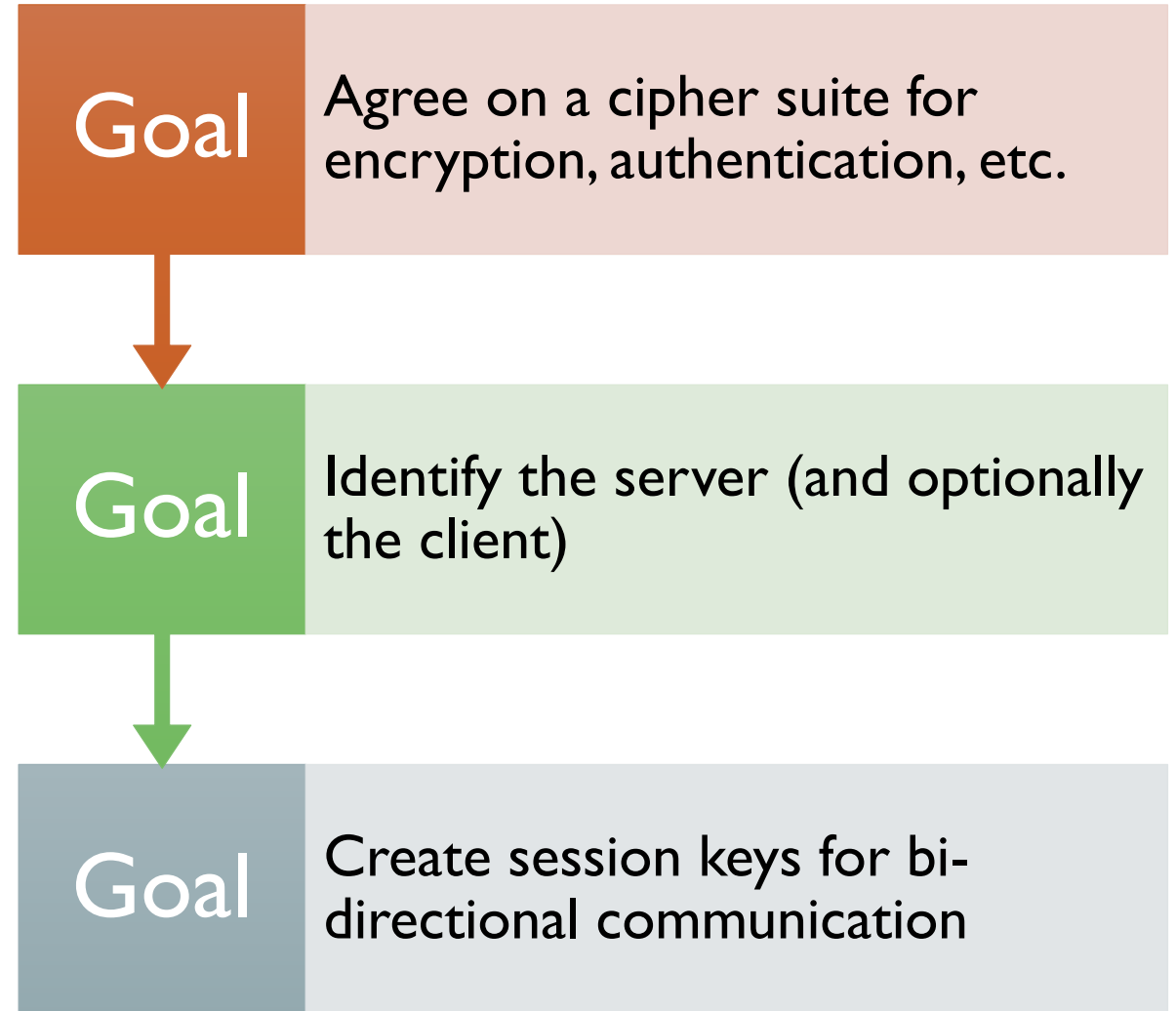
DATA ORIGIN AUTHENTICATION

- 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., $h(\text{key} + \text{data})$

TLS 1.2 HANDSHAKE



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		

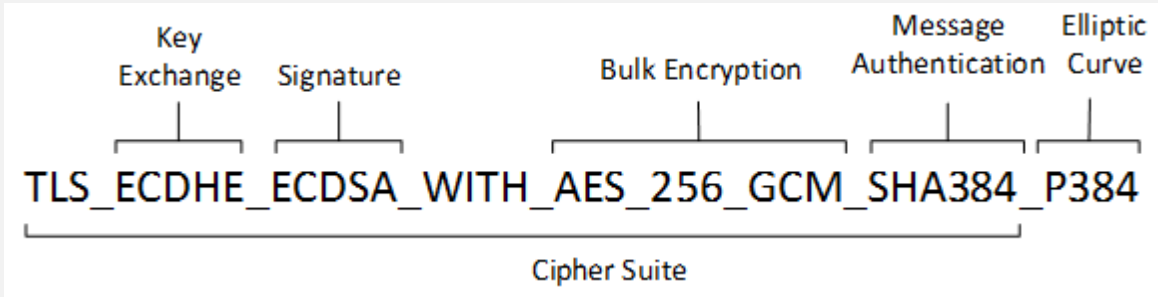
CLIENT HELLO

```
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

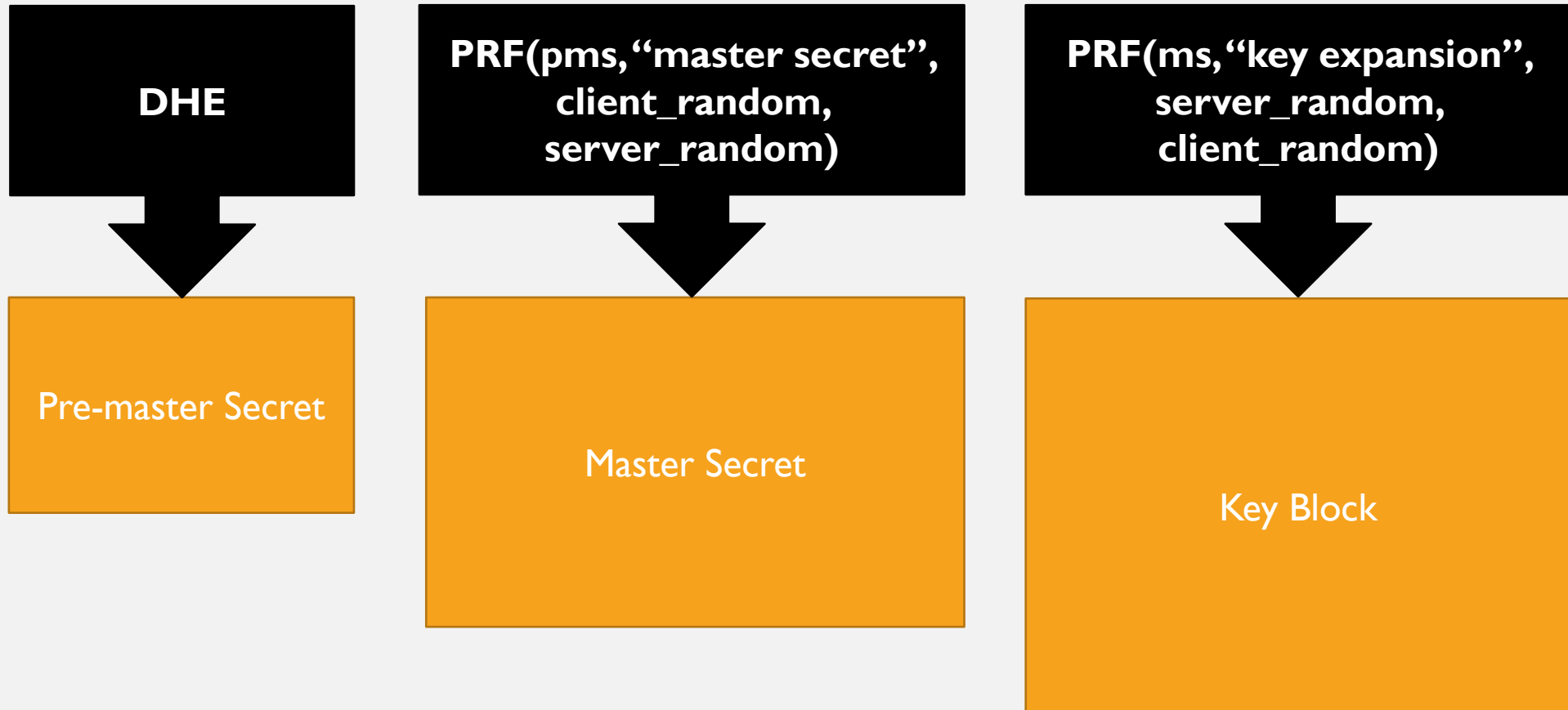
- If RSA **key transport**, sends a “pre master secret” encrypted under the server’s RSA public key from the server’s certificate
- But, for DHE **key agreement**, sends DHE public key. “pre master secret” computed

```
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;
```

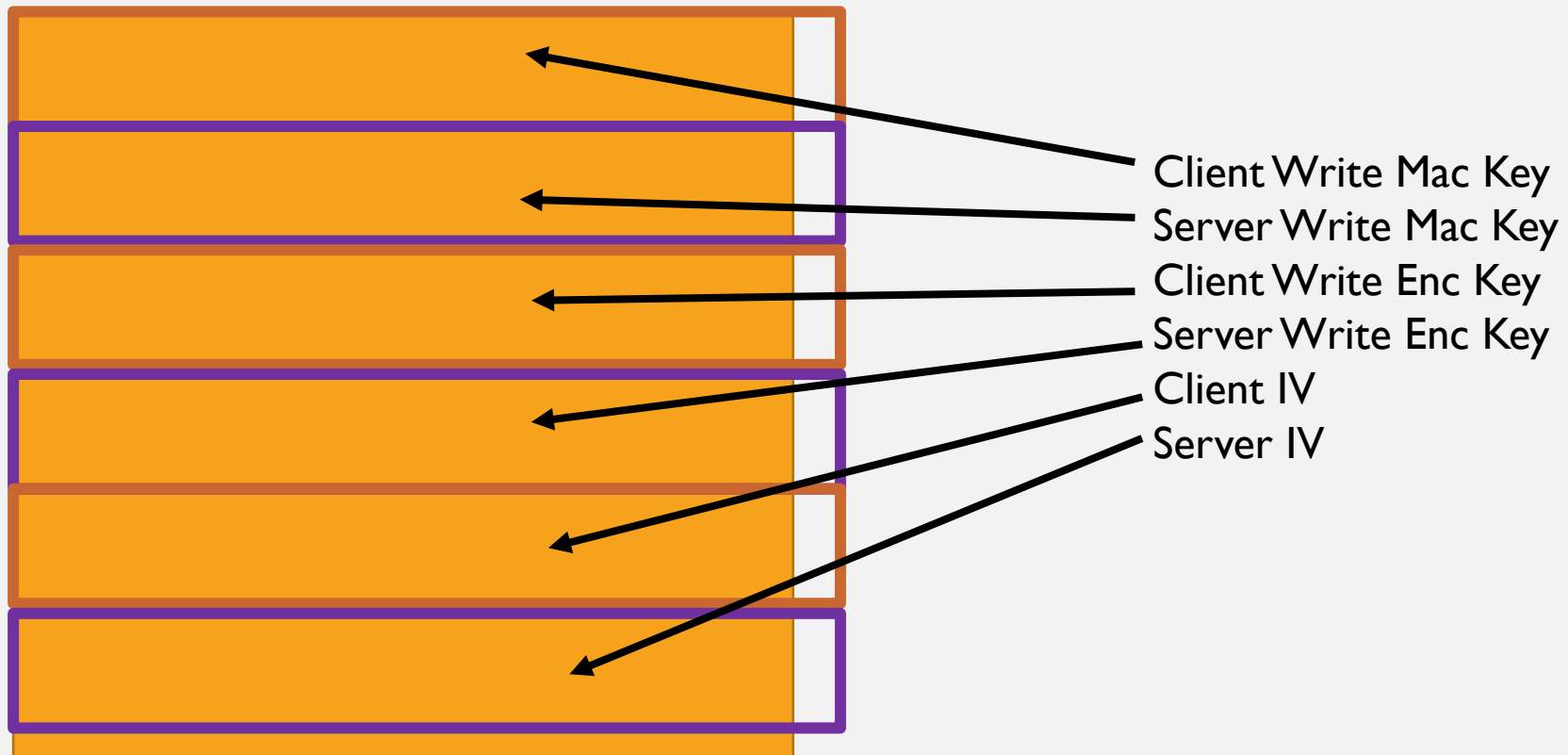
DERIVING KEYS

- 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 DERIVATION

- The size 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:
 - $\text{PRF}(\text{master_secret}, \text{"client finished"}, \text{Hash}(\text{handshake_messages}))$

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
 - $\text{PRF}(\text{master_secret}, \text{"server finished"}, \text{Hash}(\text{handshake_messages}))$