session identifier

An arbitrary byte sequence chosen by the server to identify an active or resumable session state

peer certificate

X509.v3[X509] certificate of the peer. This element of the state may be null.

compression method

The algorithm used to compress data prior to encryption.

cipher spec

Specifies the bulk data encryption algorithm (such as null, DES, etc.) and a MAC algorithm (such as MD5 or SHA). It also defines cryptographic attributes such as the `hash_size`.

master secret

48-byte secret shared between the client and server.

48 byte = 384 bit

is resumable

server and client random → random # generated by client server

Byte sequences that are chosen by the server and client for each connection.

server write MAC secret

The secret used in MAC operations on data written by the server.

client write MAC secret

The secret used in MAC operations on data written by the client.

server write key

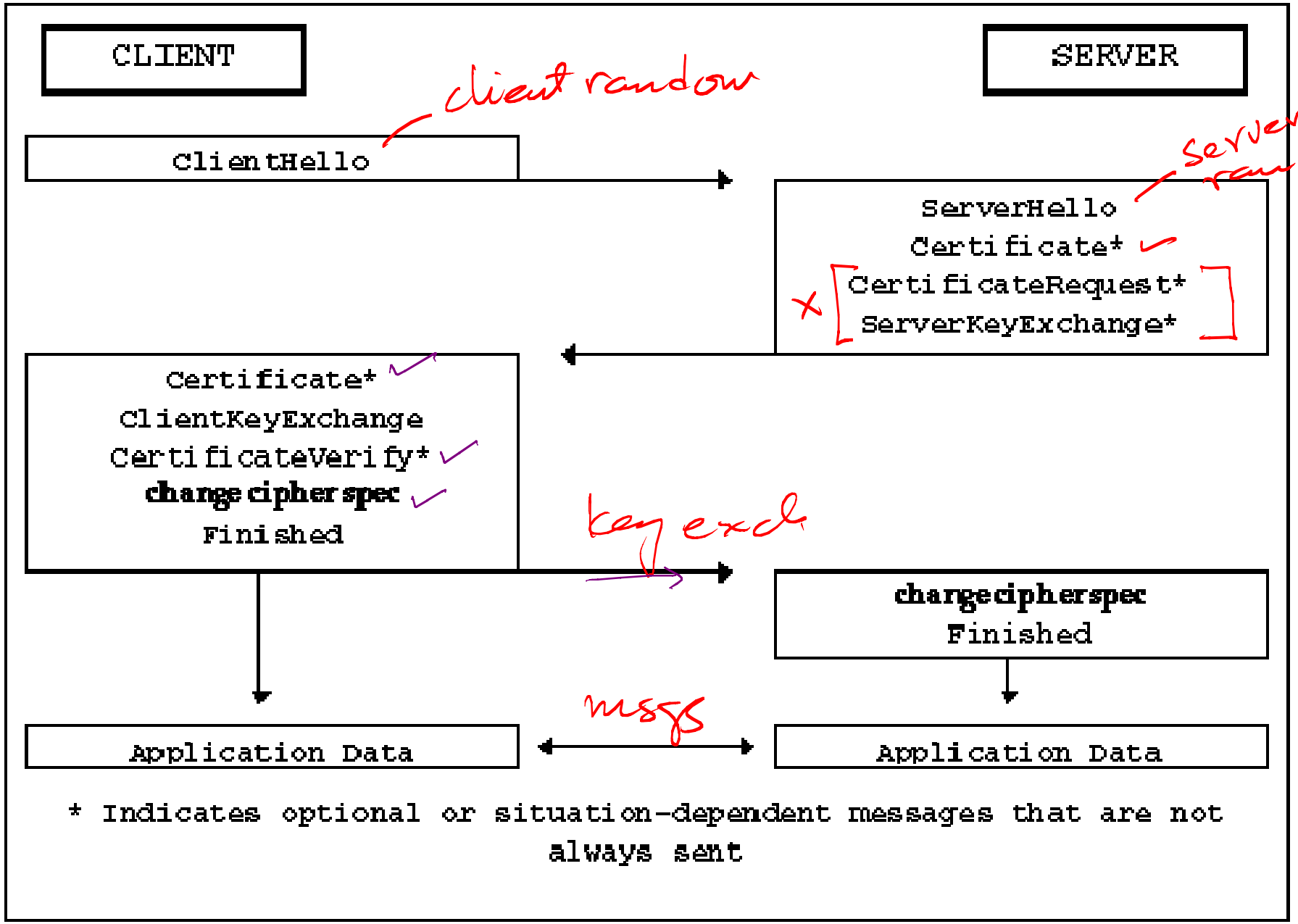
The bulk cipher key for data encrypted by the server and decrypted by the client.

client write key

The bulk cipher key for data encrypted by the client and decrypted by the server.

initialization vectors

When a block cipher in CBC mode is used, an initialization vector (IV) is


CLIENT
SERVER
client random
ClientHello
server rando
ServerHello
Certificate*
CertificateRequest*
ServerKeyExchange*
Certificate*
ClientKeyExchange
CertificateVerify*
change cipher spec
Finished
key exch
change cipher spec
Finished
msgs
Application Data
Application Data
* Indicates optional or situation-dependent messages that are not always sent

The **client hello** message includes a random structure, which is used later in the protocol.

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

`gmt_unix_time`

The current time and date in standard UNIX 32-bit format according to the sender's internal clock. Clocks are not required to be set correctly by the basic SSL Protocol; higher level or application protocols may define additional requirements.

`random_bytes`

28 bytes generated by a secure random number generator.

The server processes the **client hello** message and responds with either a `handshake_failure` **alert** or **server hello** message.

```
struct {
        ProtocolVersion server_version;
        Random random;
        SessionID session_id;
        CipherSuite cipher_suite;
        CompressionMethod compression_method;
} ServerHello;
```

If RSA is being used for key agreement and authentication, the client generates a 48-byte pre-master secret, encrypts it with server public key.

```
struct {
        ProtocolVersion client_version;
        opaque random[48];
} PreMasterSecret;
```

48 bytes = 384 bits

Sent encrypted → with server pub key

This random value is generated by the client and is used to generate the *master secret*.

```
master_secret =
        MD5(pre_master_secret +
            SHA('A' + pre_master_secret +
            ClientHello.random +
                 ServerHello.random)) +
        MD5(pre_master_secret +
            SHA('BB' + pre_master_secret +
            ClientHello.random +
                 ServerHello.random))   +
        MD5(pre_master_secret +
            SHA('CCC' + pre_master_secret +
            ClientHello.random +
                   ServerHello.random));
```

To generate the key material, compute

```
key_block =
        MD5(master_secret +
            SHA('A' + master_secret +
            ServerHello.random +
            ClientHello.random)) +
        MD5(master_secret +
           SHA('BB' + master_secret +
           ServerHello.random +
           ClientHello.random)) +
        MD5(master_secret +
           SHA('CCC' + master_secret       +
           ServerHello.random +
           ClientHello.random)) + [...];
```

until enough output has been generated.

Then the `key_block` is partitioned as follows.

```
client_write_MAC_secret[CipherSpec.hash_size]

server_write_MAC_secret[CipherSpec.hash_size]

client_write_key[CipherSpec.key_material]

server_write_key[CipherSPec.key_material]
```

The MAC is generated as:

```
hash(MAC_write_secret + pad_2 +
        hash (MAC_write_secret + pad_1 +
              seq_num + length + content) );
```

`pad_1`

The character 0x36 repeated 48 time for MD5 or 40 times for SHA.

`pad_2`

The character 0x5c repeated the same number of times.

`seq_num`

The sequence number for this message.

`hash`

The hashing algorithm derived from the cipher suite.