A2. Transport layer security (TLS)

Many slides from Jinyuan Sun@U. of Tennessee

http vs https

http

- HyperText Transfer Protocol
- No certificate
- No encryption
- TLS not used
- No privacy

https

- Hypertext Transfer Protocol Secure
- Certificate
- Encryption
- Use TLS
- Privacy

What is SSL/TLS?

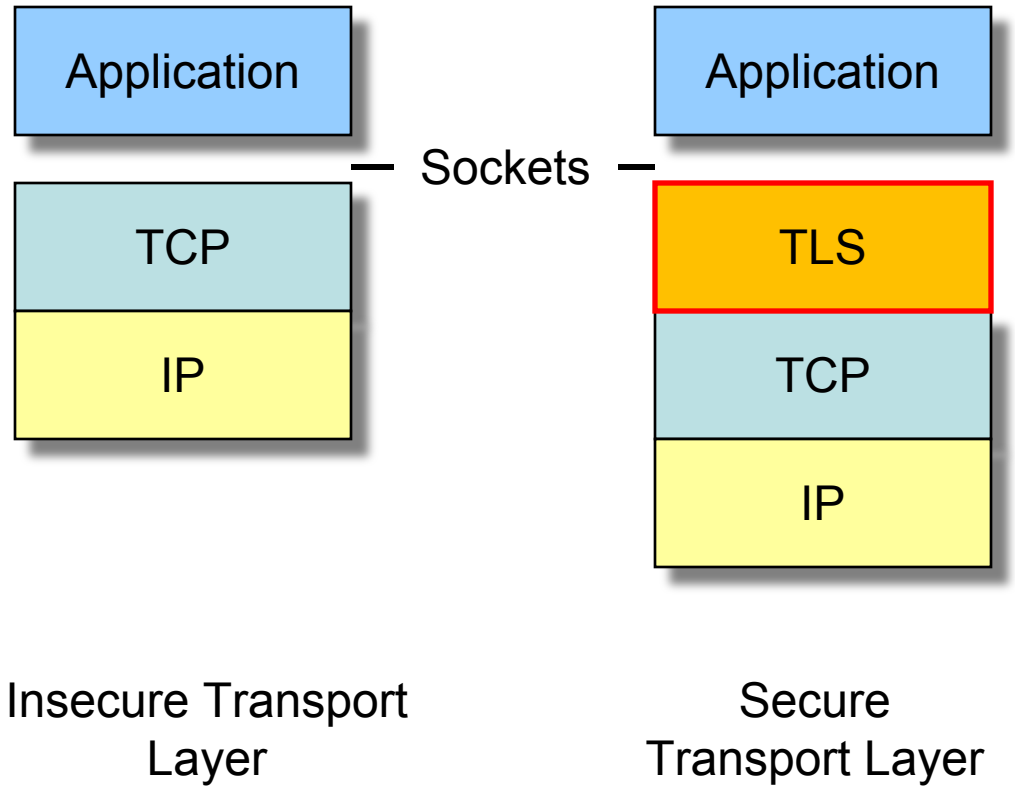
- Transport Layer Security (TLS) protocol
  - De facto standard for Internet security
  - “The primary goal of the TLS protocol is to provide privacy and data integrity between two communicating applications”
  - In practice, used to protect information transmitted between browsers and Web servers
- Based on Secure Sockets Layer (SSL)
  - Same protocol design, different algorithms
  - SSL is the old version and deprecated.
- X.509 certificates are called SSL/TLS certificates
- Deployed in every Web browser

how can I know TLS is used

- look at the address line in the browser

source: https://www.dreamstime.com/ssl-sertificate-website-icon-secure-https-safe-connection-site-browser-url-data-image140285429

Application-Level Protection



Source: Andreas Steffen@ITA

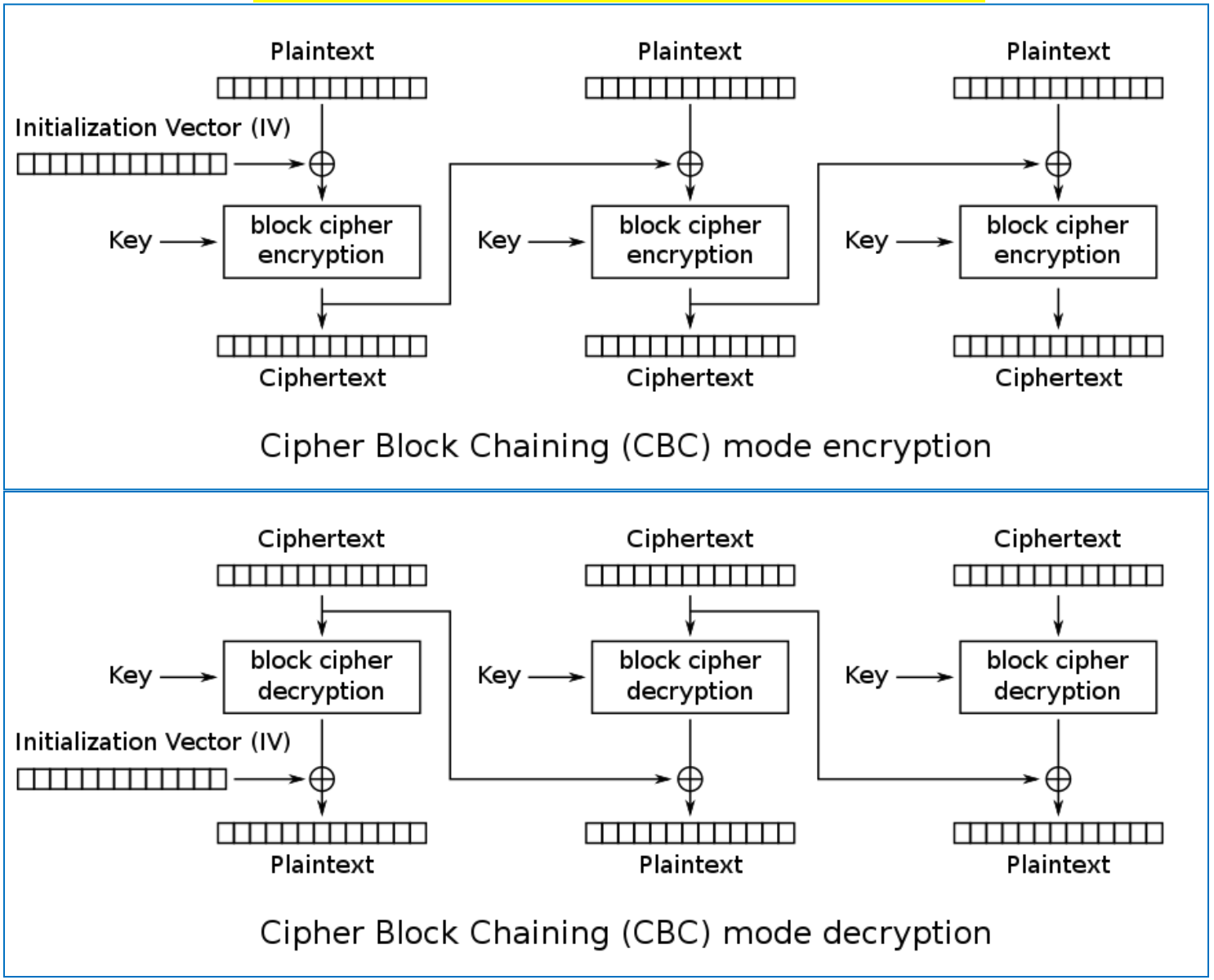
TLS history (1/2)

- SSL 1.0
  - Internal Netscape design, 1994
  - Not publicly released
- SSL 2.0
  - Published by Netscape, 1995
  - Several weaknesses
- SSL 3.0
  - Designed by Netscape and Paul Kocher, 1996
- TLS 1.0
  - IETF makes RFC 2246 based on SSL 3.0, 1999
  - Not interoperable with SSL 3.0
    - TLS uses HMAC instead of MAC; can run on any port

TLS history (2/2)

- TLS 1.1, 2006
  - RFC 4346
  - Protection against cipher-block chaining (CBC) padding attacks
- TLS 1.2, 2008
  - RFC 5246
  - More options in cipher suite
    - Eg. SHA 256, AES-related
- TLS 1.3, 2018
  - RFC 8446
  - Some insecure ciphers removed (RC4, DES,...)
  - streamline RTT handshakes (e.g. 0-RTT mode)

CBC mode of operation



Original CBC: MAC -> padding -> encrypt

CBC: padding oracle attack

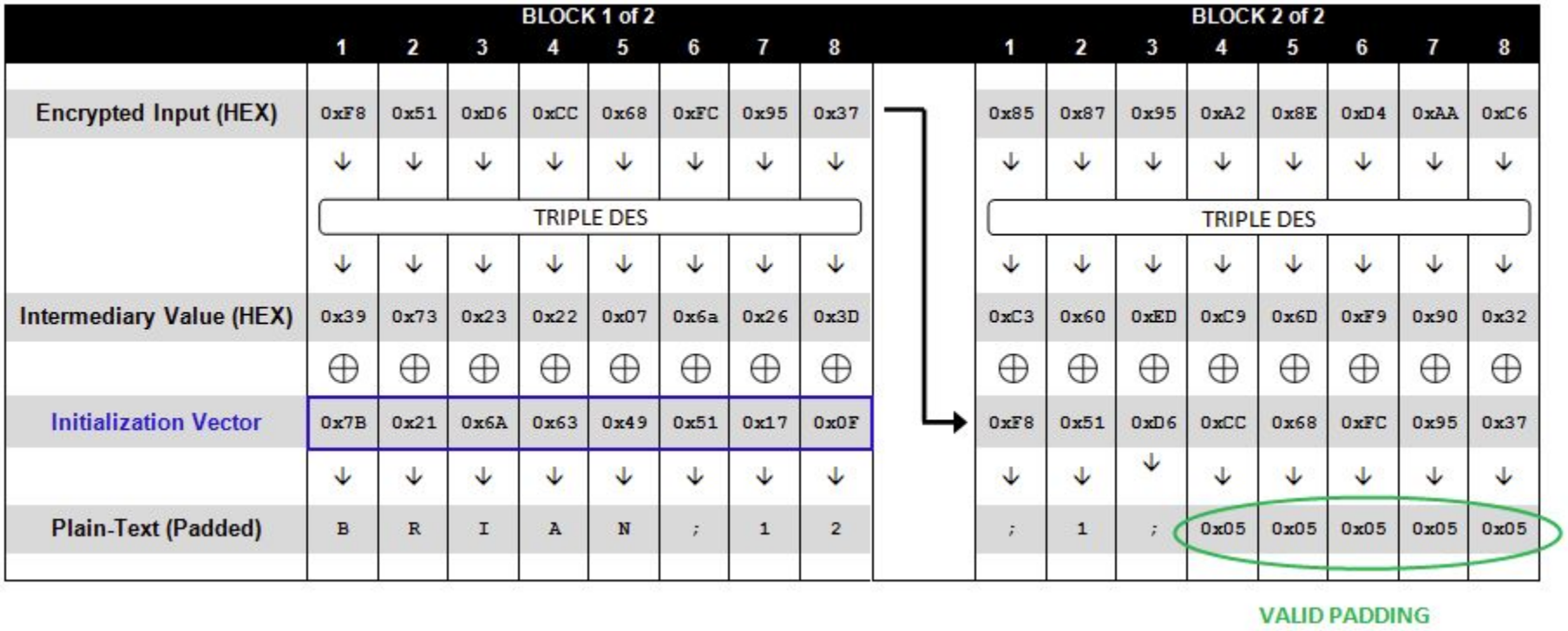
- PKCS#7 padding, assuming 64 bit block

	BLOCK #1								BLOCK #2							
	1	2	3	4	5	6	7	8	1	2	3	4	5	6	7	8
Ex 1	F	I	G													
Ex 1 (Padded)	F	I	G	0x05	0x05	0x05	0x05	0x05								
Ex 2	B	A	N	A	N	A										
Ex 2 (Padded)	B	A	N	A	N	A	0x02	0x02								
Ex 3	A	V	O	C	A	D	O									
Ex 3 (Padded)	A	V	O	C	A	D	O	0x01								
Ex 4	P	L	A	N	T	A	I	N								
Ex 4 (Padded)	P	L	A	N	T	A	I	N	0x08	0x08	0x08	0x08	0x08	0x08	0x08	0x08
Ex 5	P	A	S	S	I	O	N	F	R	U	I	T				
Ex 5 (Padded)	P	A	S	S	I	O	N	F	R	U	I	T	0x04	0x04	0x04	0x04

source: https://kimtruth.github.io/2020/04/12/padding-oracle-attack/

CBC: padding oracle attack

- CBC: decrypt 3 letters (5 byte padding)



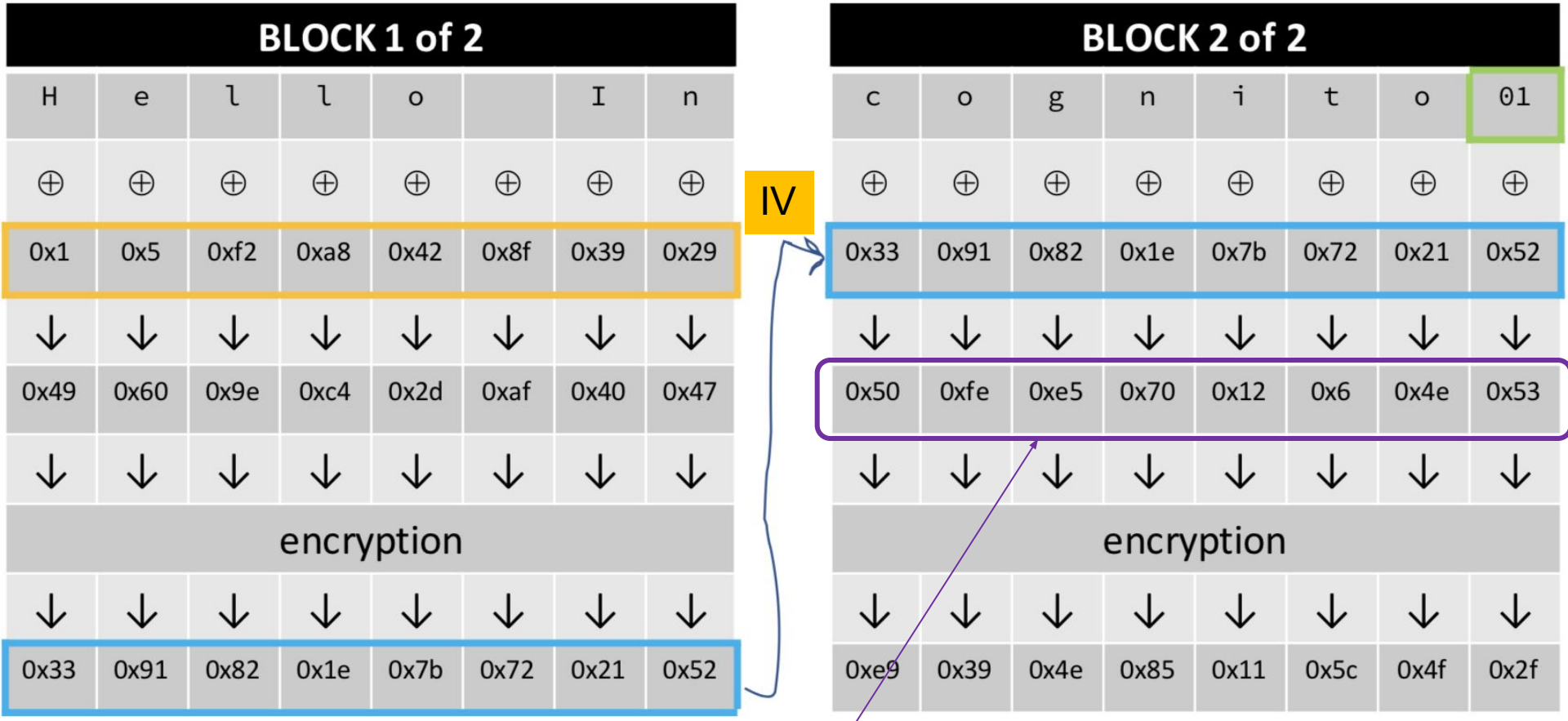
If padding fails, outputs “padding error”
Else if MAC fails, outputs “MAC error”

source: https://kimtruth.github.io/2020/04/12/padding-oracle-attack/

CBC: padding oracle attack

- CBC: encrypt “Hello□Incognito”
  - 1 byte (0x01) padding

I9..I16 = C1..C8 ⊕ P9..P16



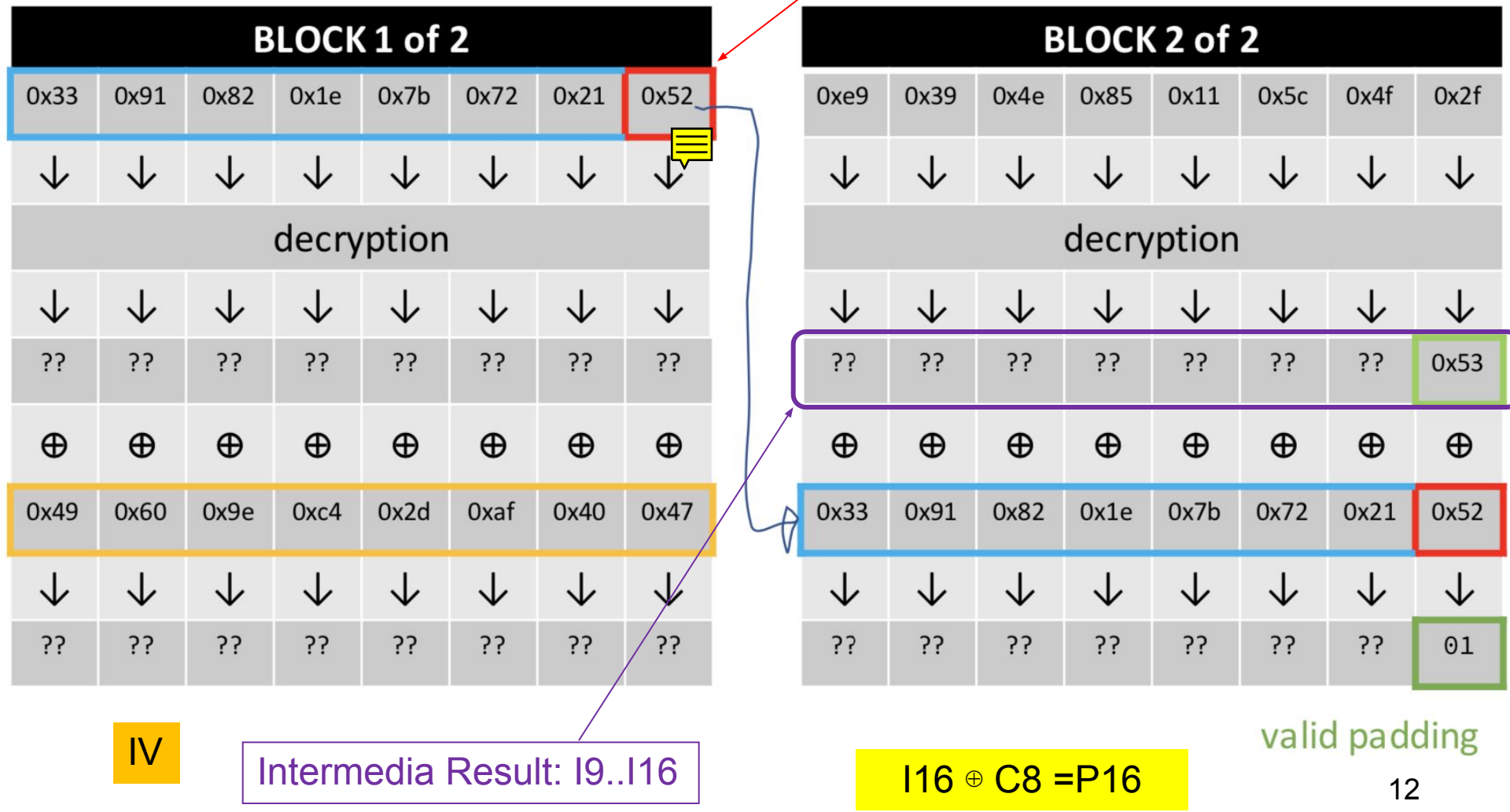
C1..C8

source: https://kimtruth.github.io/2020/04/12/padding-oracle-attack/

Intermedia Result: I9..I16

CBC: padding oracle attack

- CBC: padding attack (CCA)
  - brute-force search for last byte until padding check is fine



source: https://kimtruth.github.io/2020/04/12/padding-oracle-attack/

How to find out P16 by CCA?

- I16 = C8 ⊕ P16: encryption side
- P16 = I16 ⊕ C8: decryption side

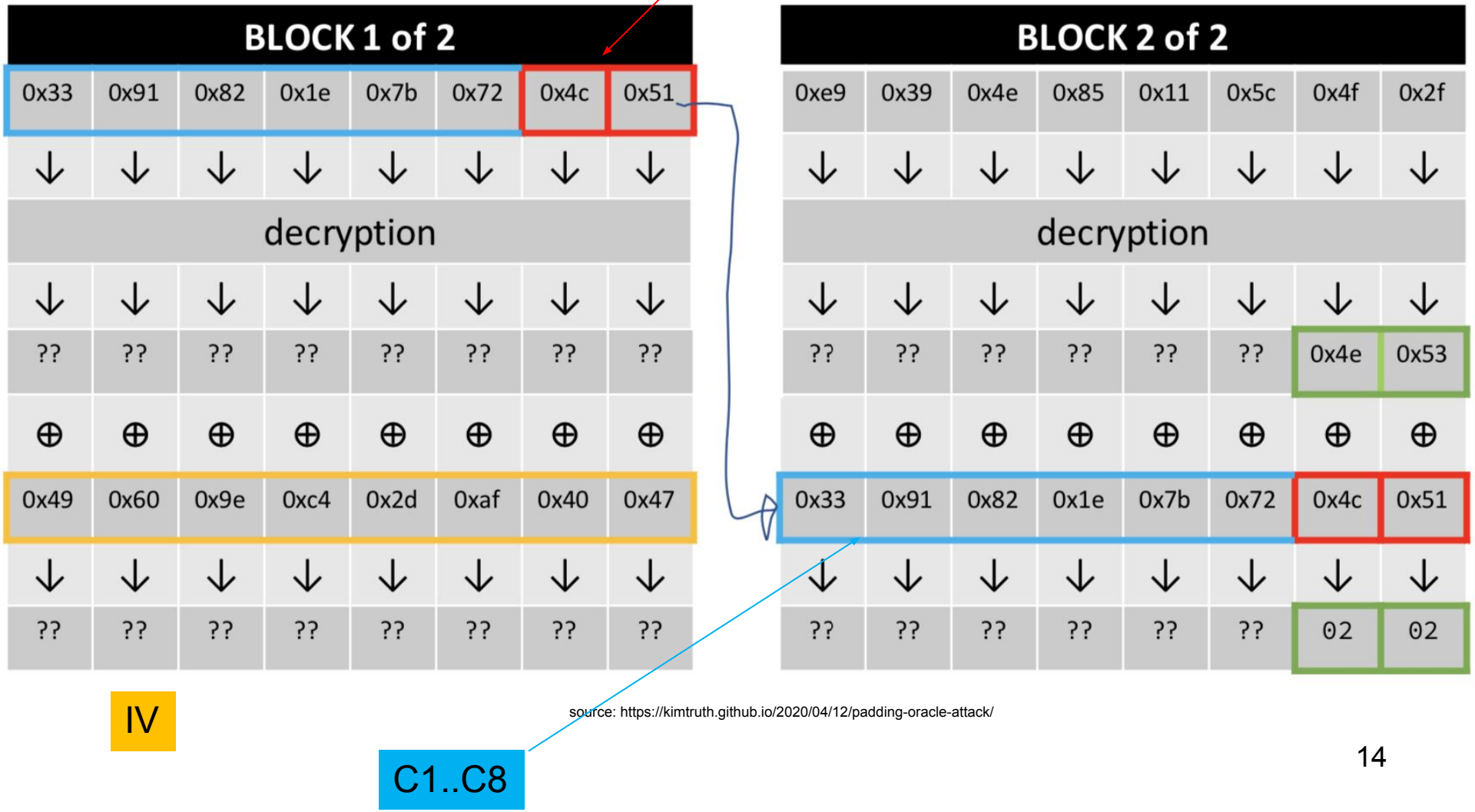
- Which C8' value makes padding success?
  - Try all the 256 values
  - P16' = 0x01
  - P16' = I16 ⊕ C8' = C8 ⊕ P16 ⊕ C8'

- then what?
  - We can calculate P16

C8' is one of possible byte values for C8 position
P16' is the changed plaintext for P16 position due to C8'

CBC: padding oracle attack

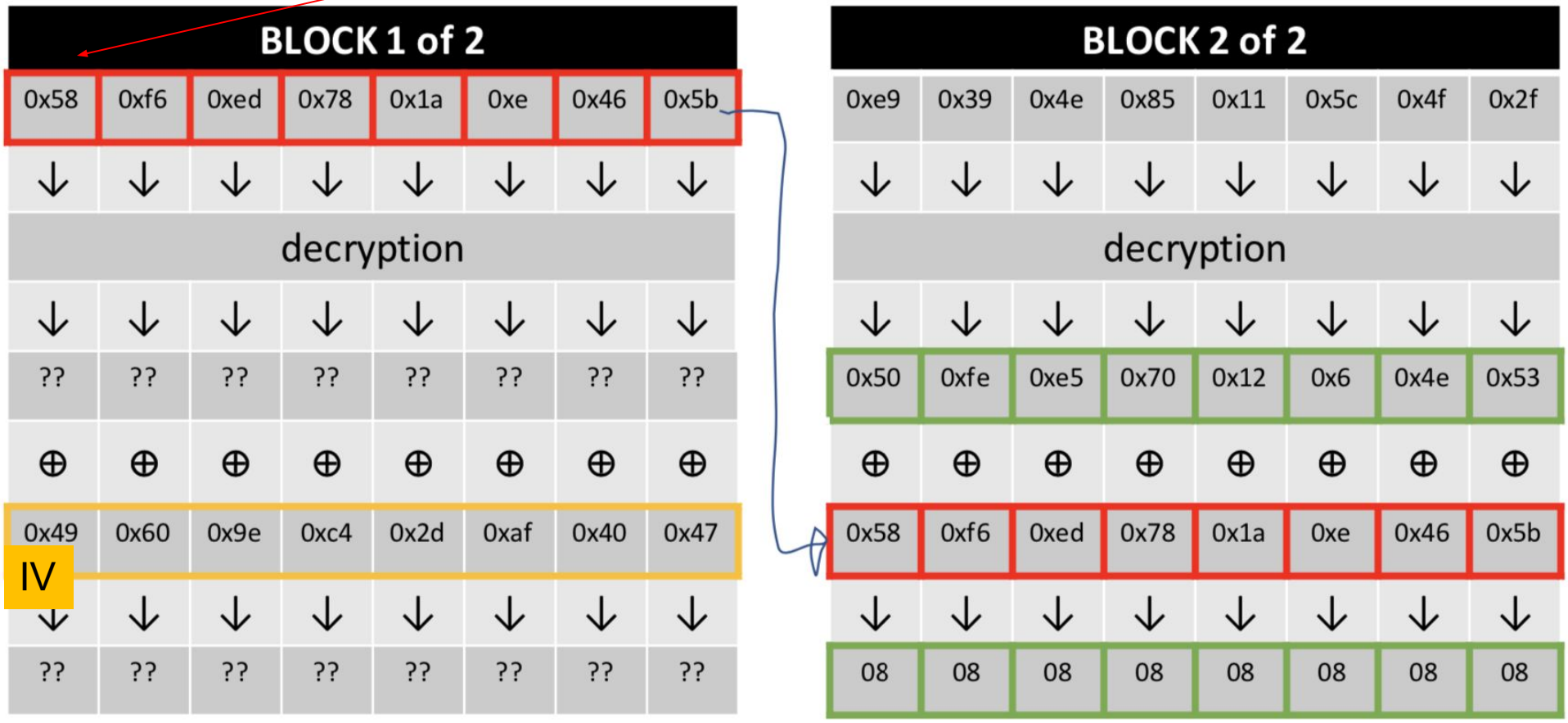
- CBC: padding attack
  - brute-force search for 2nd last byte



source: https://kimtruth.github.io/2020/04/12/padding-oracle-attack/

CBC: padding oracle attack

- CBC: padding attack
  - search continues for the 1st byte



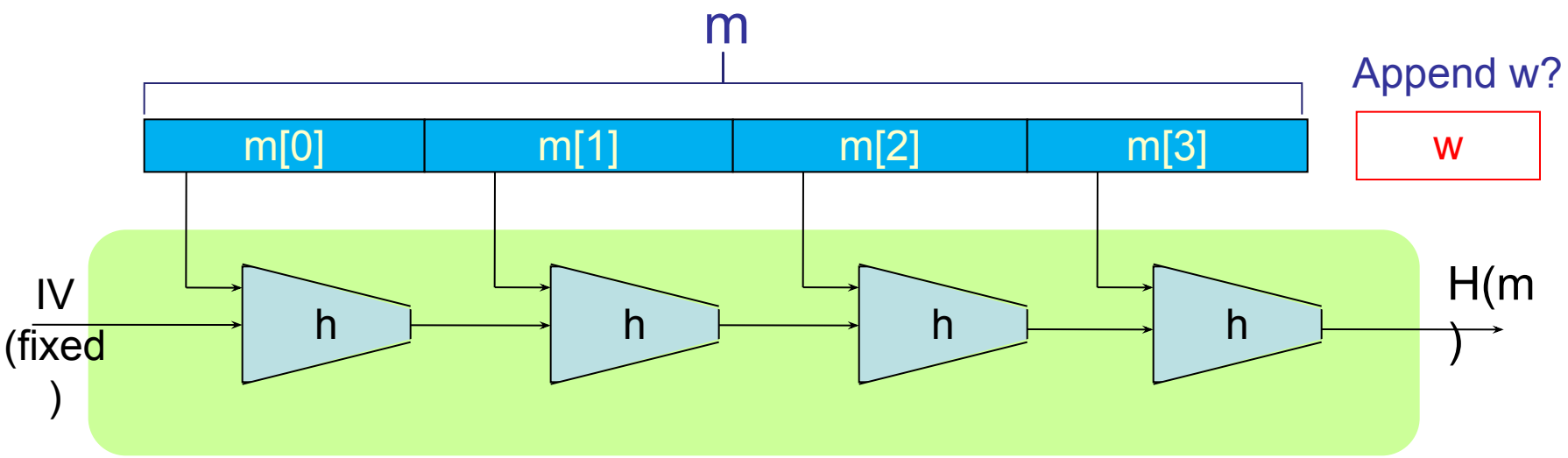
source: https://kimtruth.github.io/2020/04/12/padding-oracle-attack/

defense: encrypt-then-MAC, no CBC, or outputs only MAC error!

HMAC: Constructing MAC from Hash Fn.

- Let H be a hash function
- MAC(K,M) = H(K || M), where || denotes concatenation
  - K is key, M is msg
  - Insecure if H() has Merkle–Damgård construction
  - Length extension attack

Merkel Damgård construction

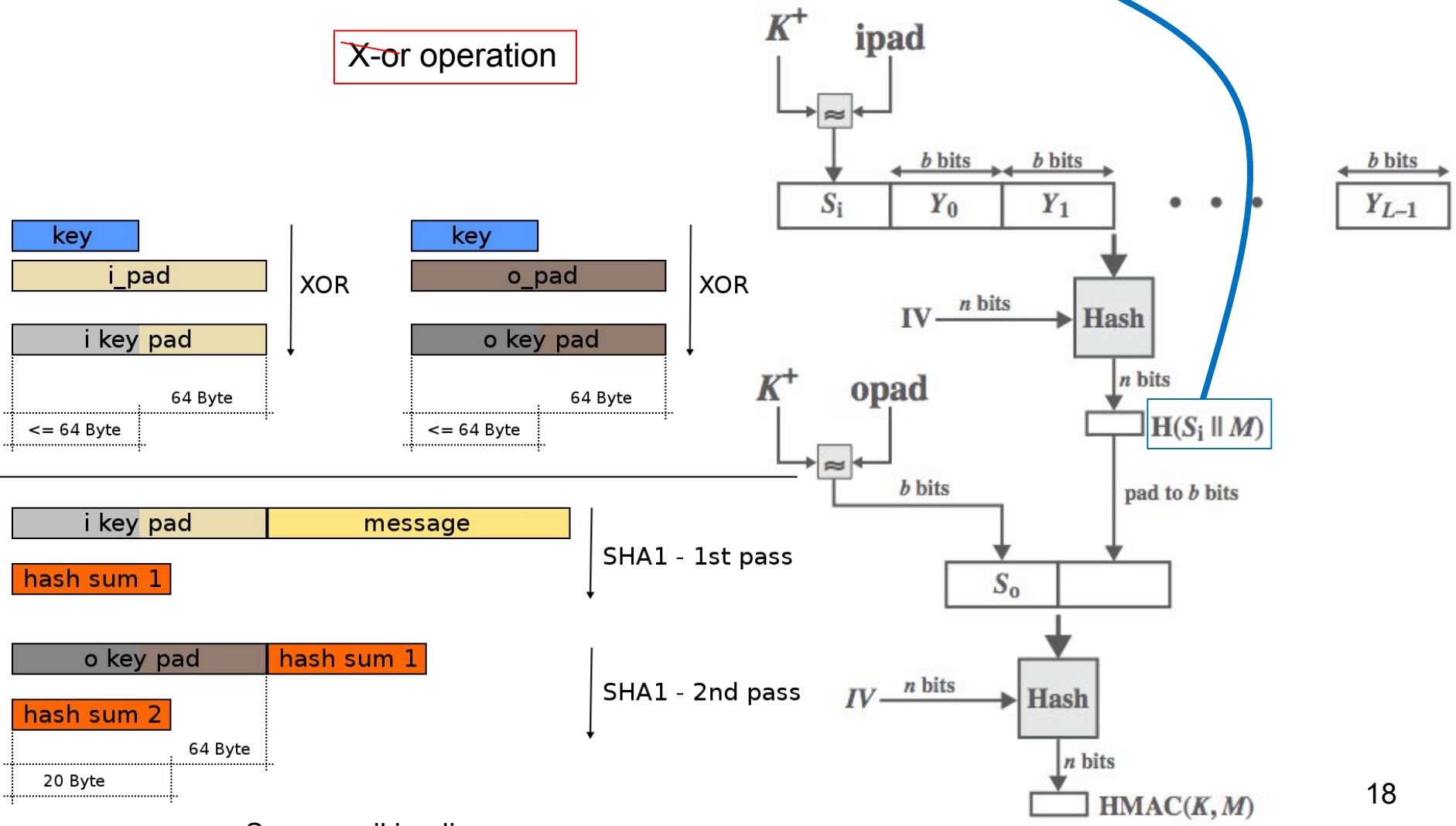


e.g. h is a compression fn. like MD5: 512 bit block → 128 bit hash

- Assume the key is already prepended into m
  - Secret||original_msg = m
- Attacker doesn't know secret or original_msg
- IV and h are publicly available
- Yet he learns H(m) and wishes to append w after m
- What if string w is appended after m?
- h(H(m),w) vs. H(m||w): length extension attack!

Hash-based MAC (HMAC)

- HMAC = H(($K^+ \oplus$ opad) || H(($K^+ \oplus$ ipad)||m))

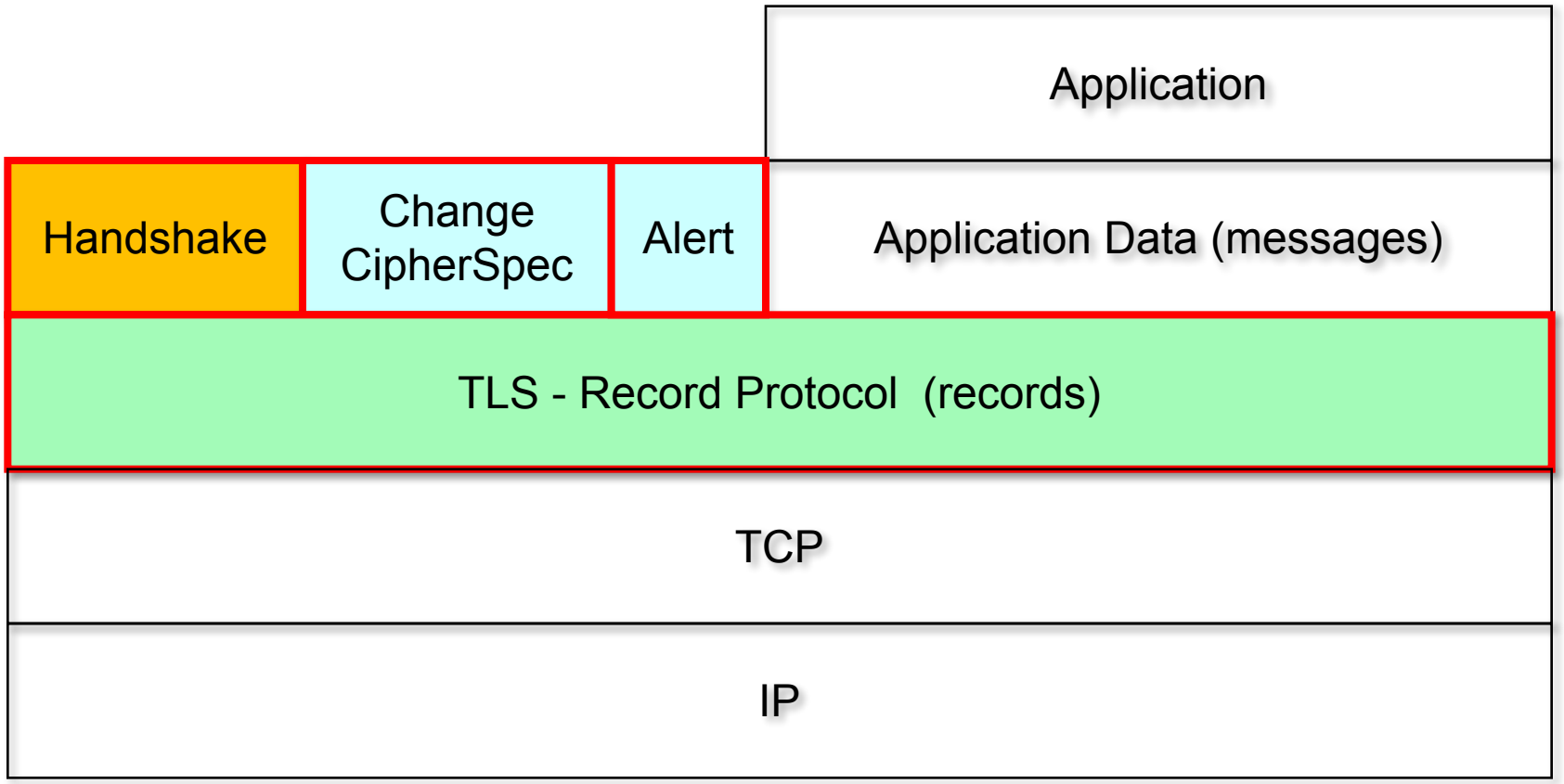


Source: wikipedia

TLS Basics

- TLS consists of two main protocols
  - total 4 protocols
- Handshake protocol
  - Use public-key cryptography to establish a shared secret key between the client and the server
- Record protocol
  - Use the secret key established in the handshake protocol to protect communication between the client and the server
- We will focus on the handshake protocol

TLS Protocol Architecture



Source: Andreas Steffen@ITA

TLS Handshake Protocol

- Two parties: client and server
- Negotiate version of the protocol and the set of cryptographic algorithms to be used
  - Interoperability between different implementations of the protocol
- Authenticate server and client
  - Use digital certificates to learn each other's public keys and verify each other's identity
  - authenticating the client is optional
- Use public keys to establish a shared secret
- Symmetric key is generated from the secret
- The following is based on TLS 1.2

Handshake + ChangeCipherSpec

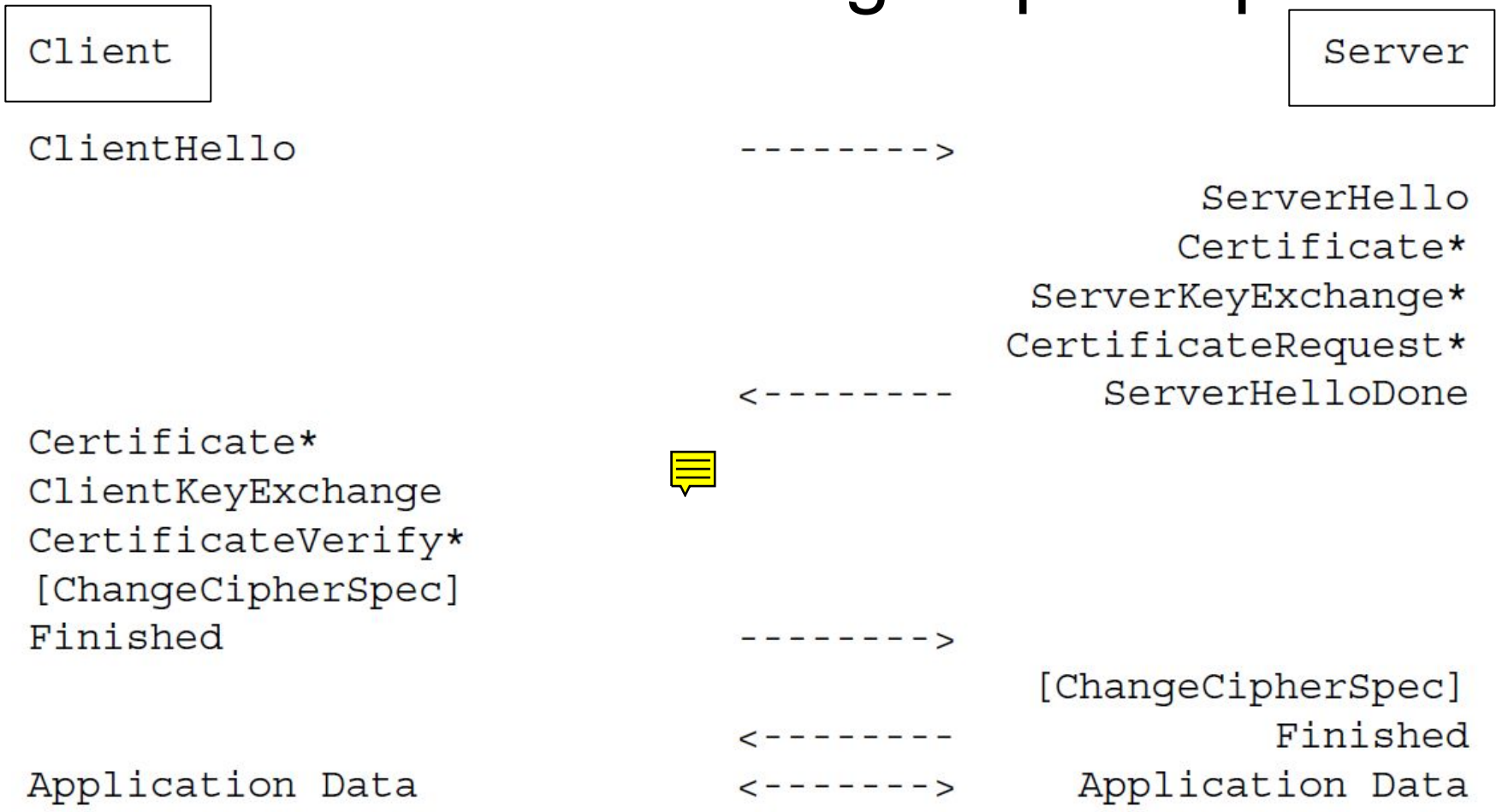
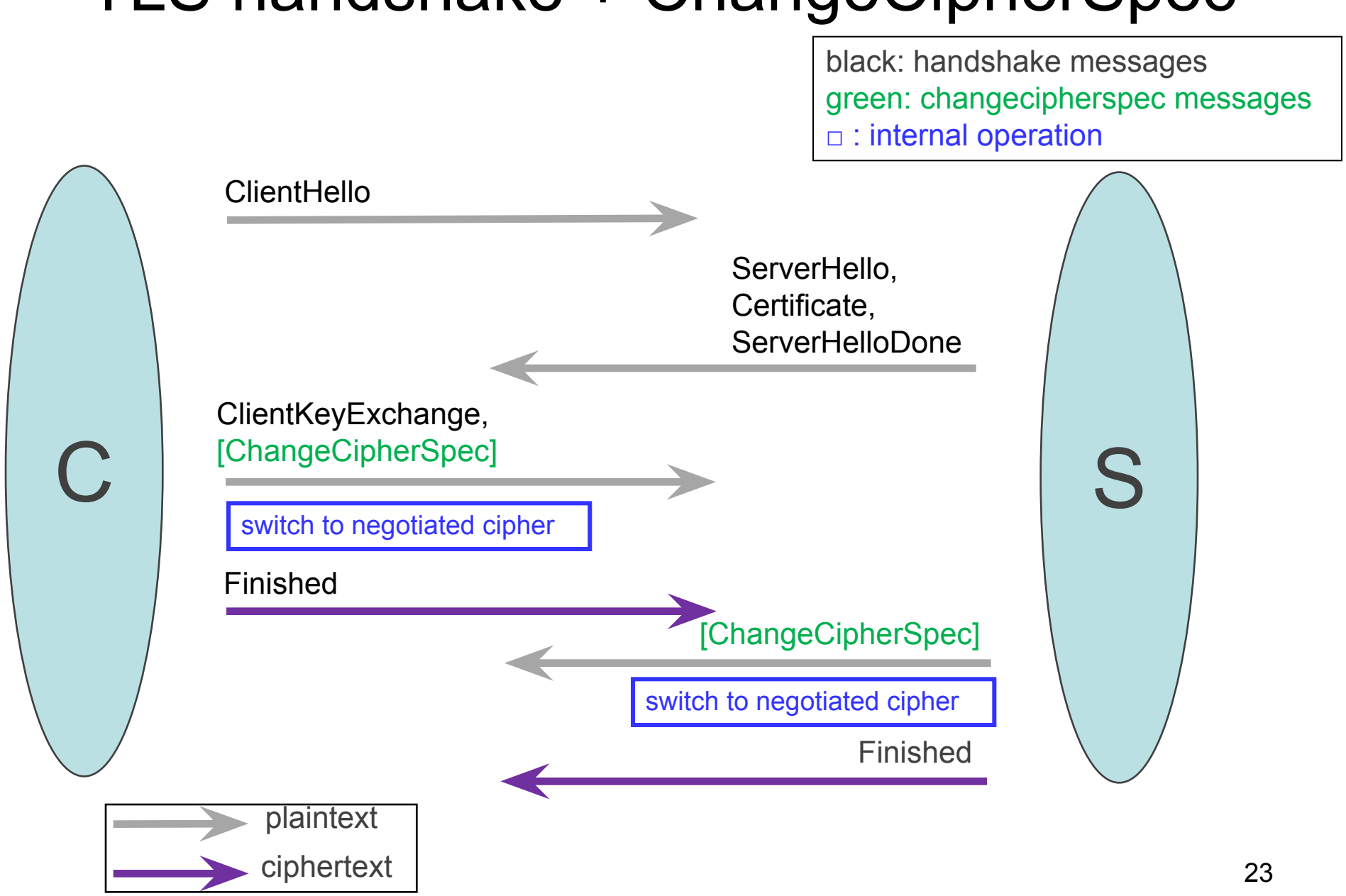


Figure 1. Message flow for a full handshake

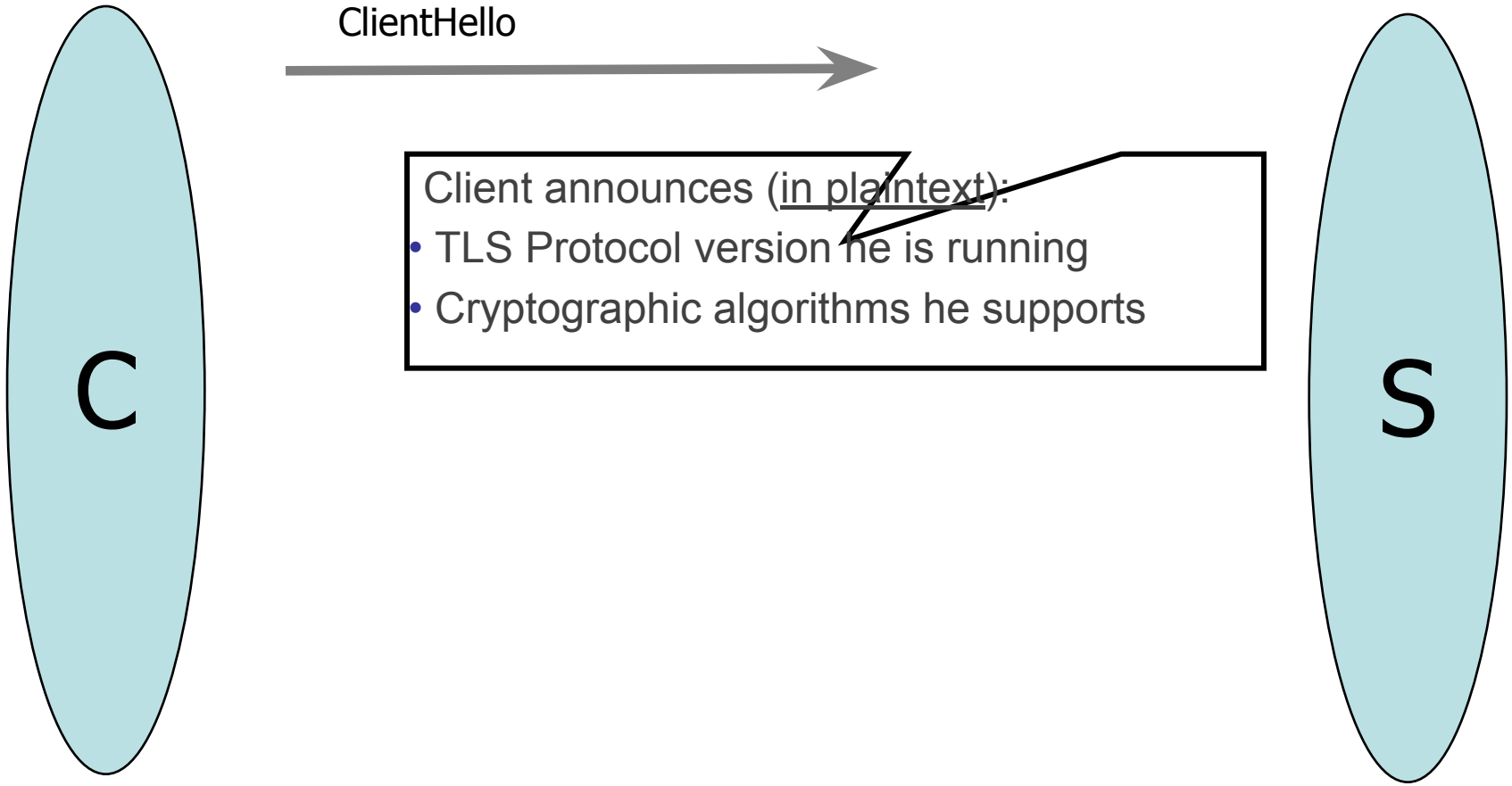
* Indicates optional or situation-dependent messages that are not always sent.

We simplify the TLS message flow in the following slides. There are many variations in the message flow depending on crypto modes.

TLS handshake + ChangeCipherSpec



ClientHello



ClientHello

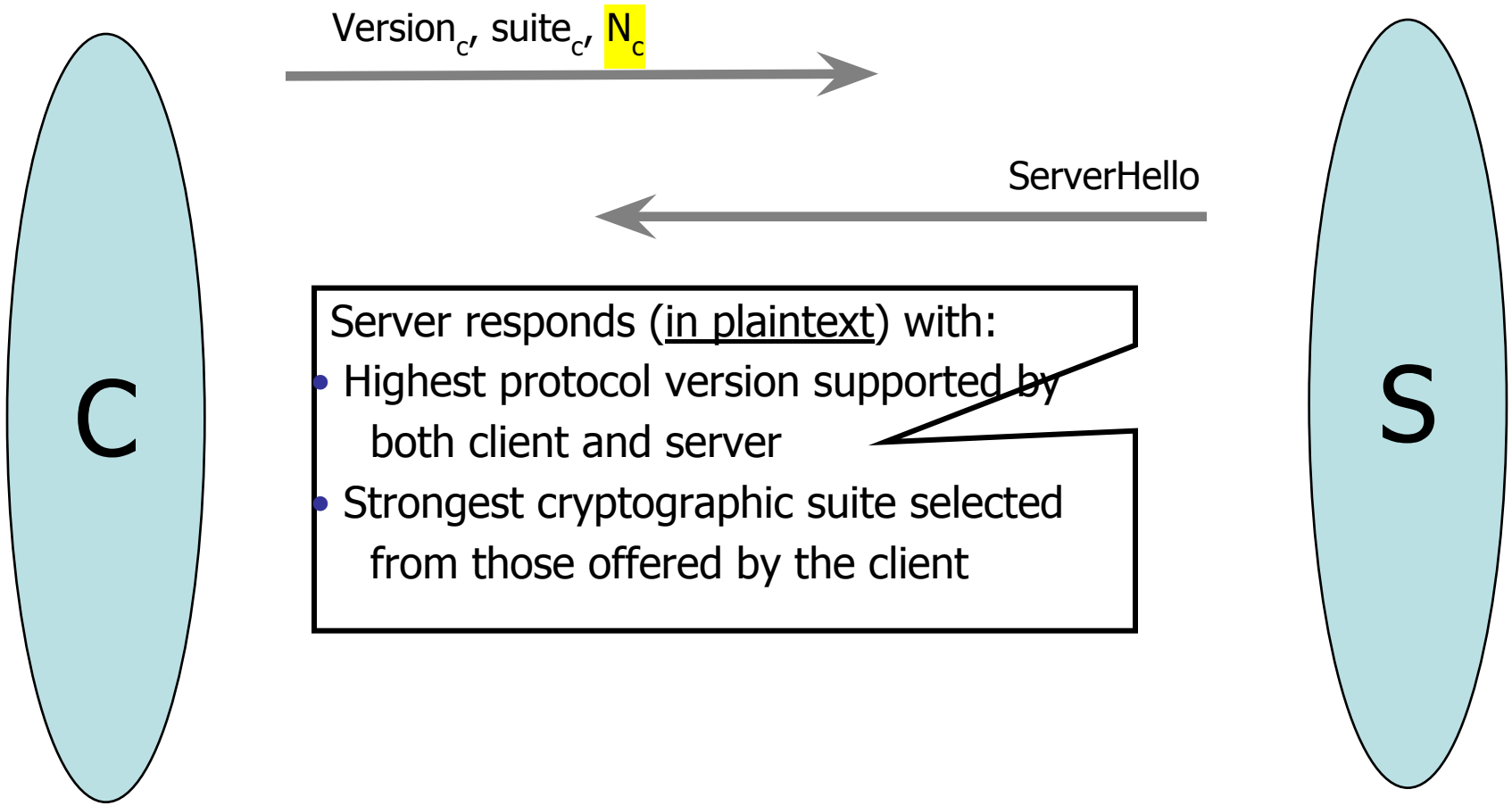
```
struct {
   ProtocolVersion client_version;
   Random random;
   SessionID session_id;
   CipherSuite cipher_suites;
   CompressionMethod
    compression_methods;
} ClientHello
```

Highest version of the protocol supported by the client

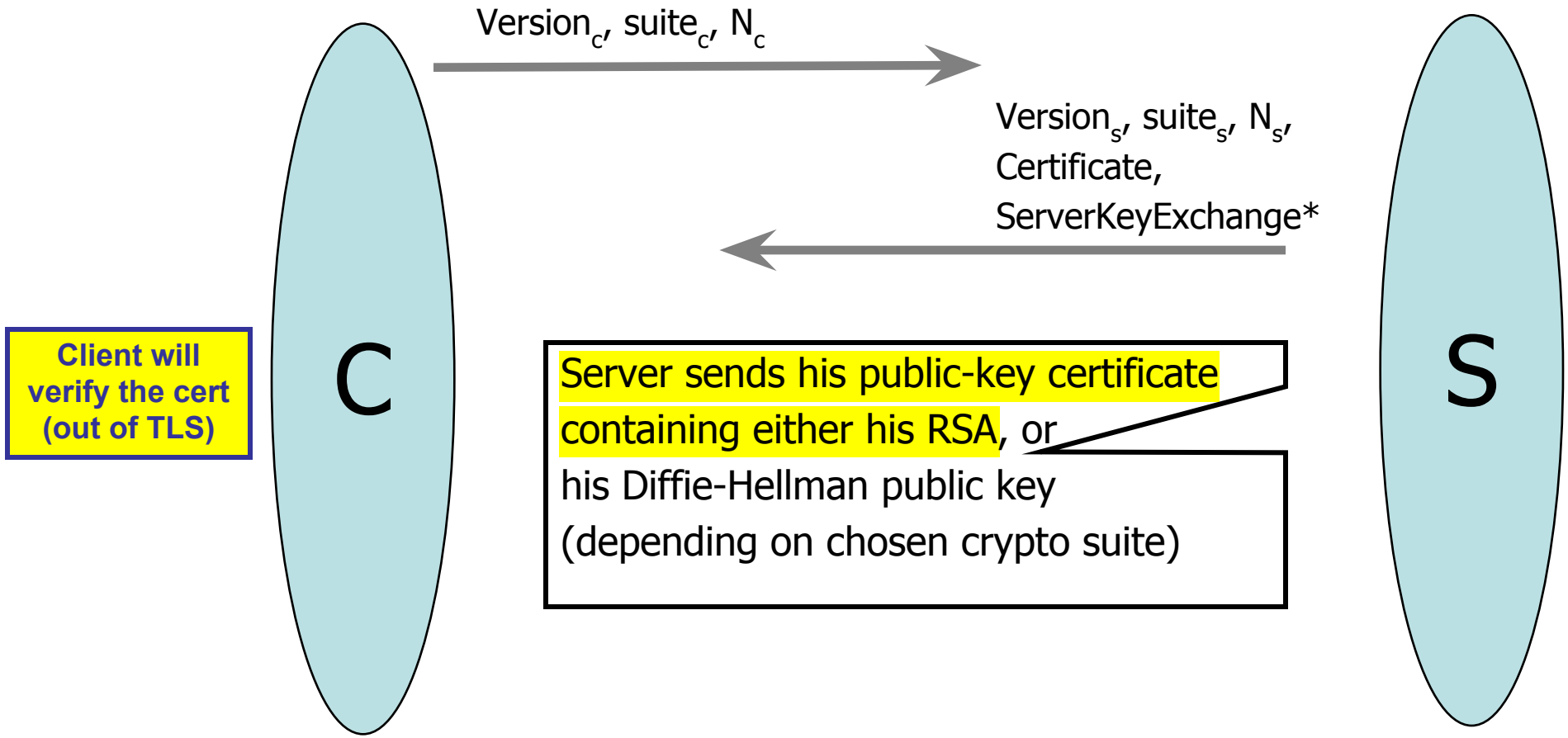
Session id (if the client wants to resume an old session)

Set of cryptographic algorithms supported by the client (e.g., RSA or Diffie-Hellman)

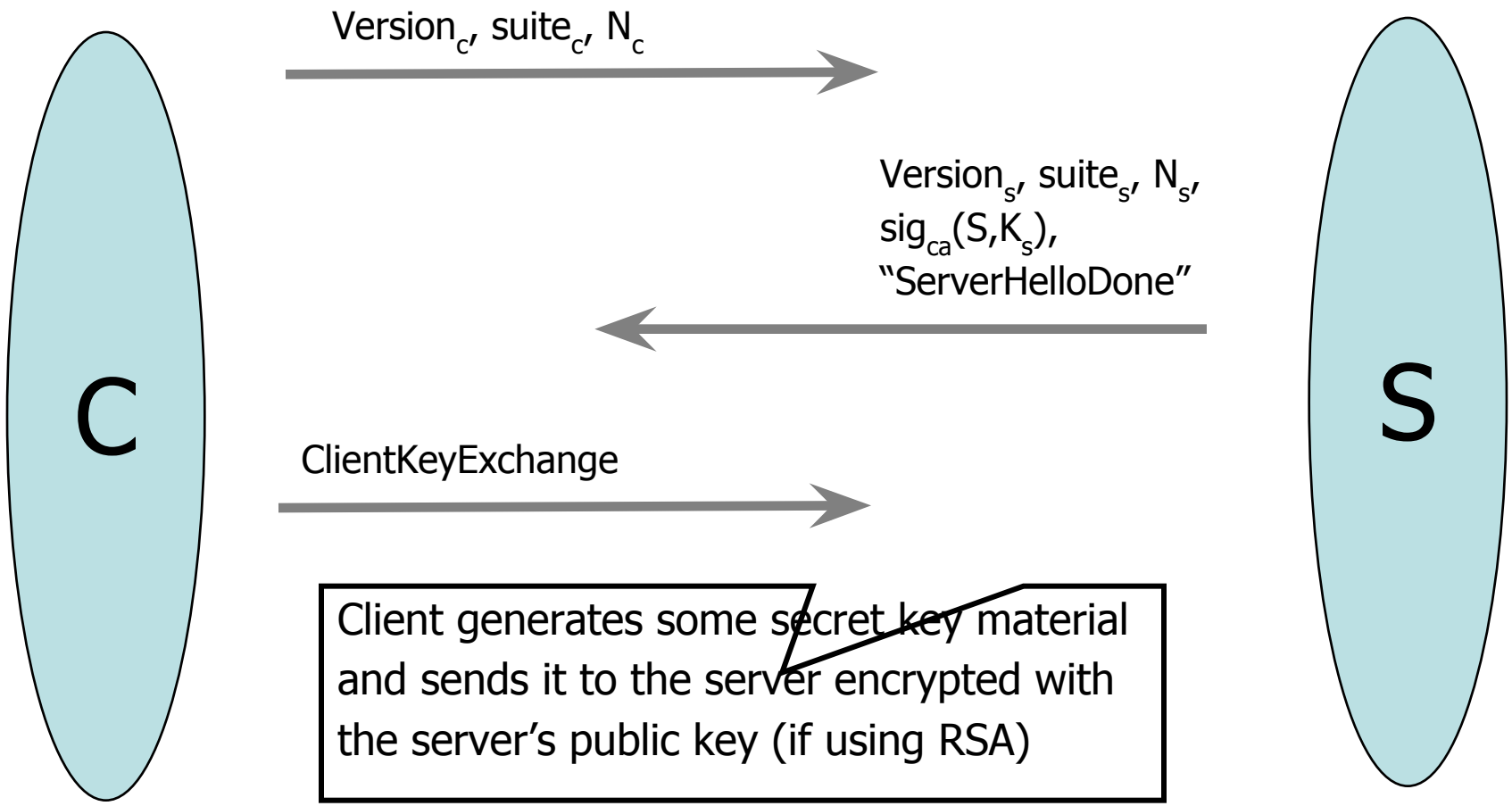
ServerHello



Certificate and/or ServerKeyExchange

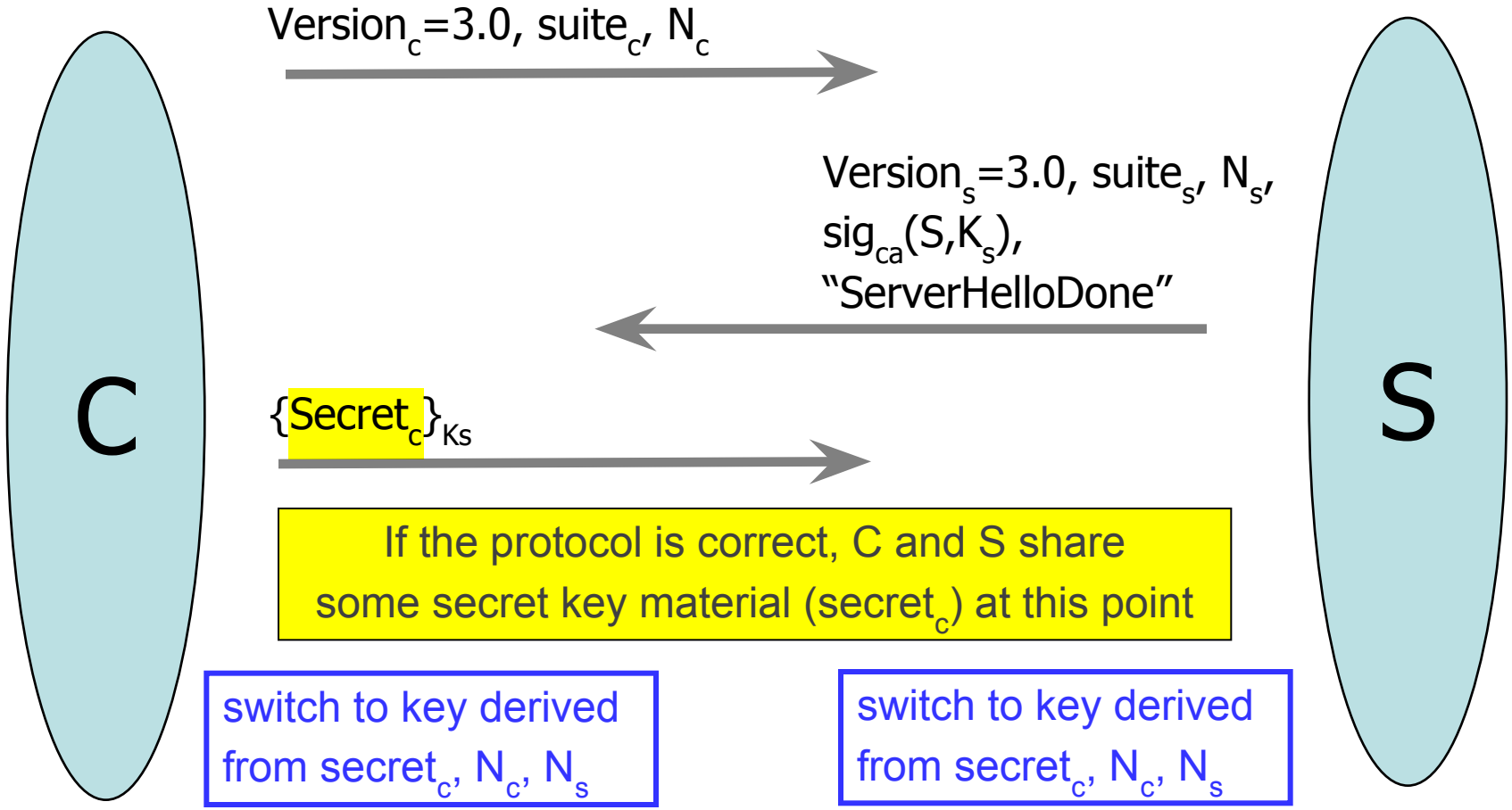


ClientKeyExchange



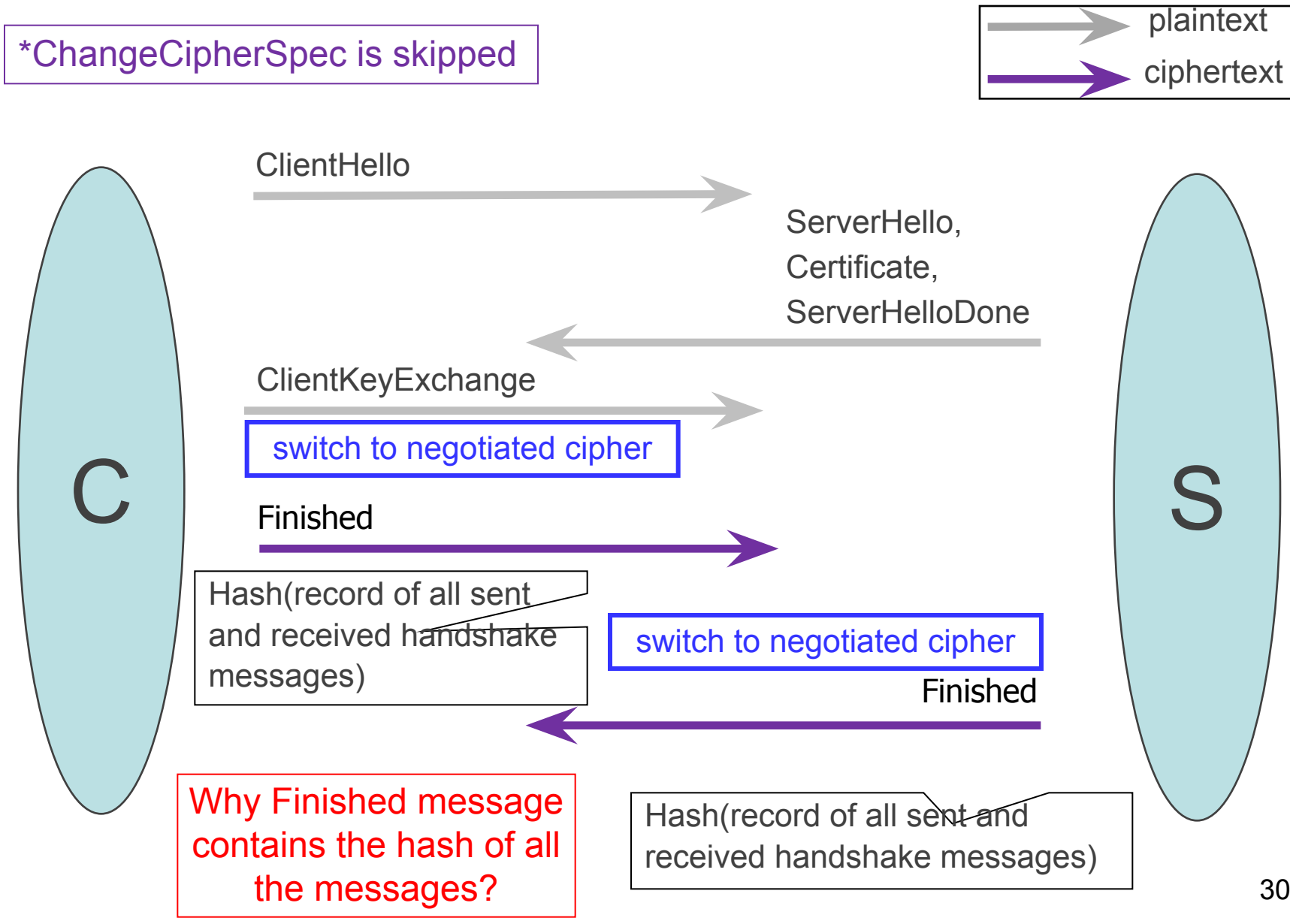
S: server identity, K_s: server's public key, sig_{ca}( ): CA's signature cert

Handshake: server certificate with RSA



secret$_c$: premaster secret, K$_s$: server's public key

Handshake Protocol Structure

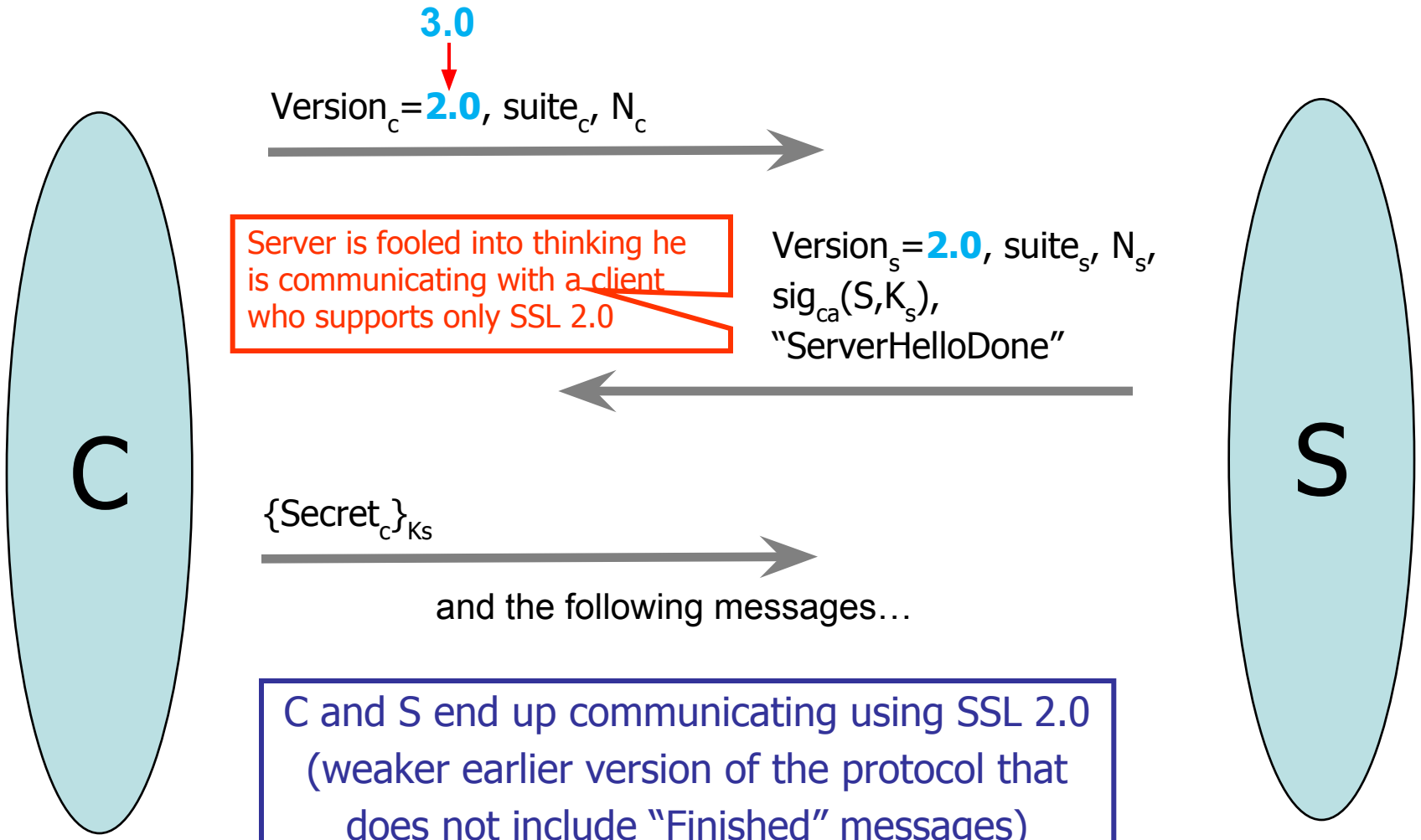


Generating master secret & keys

```
master_secret = PRF(pre_master_secret, "master secret",
                    ClientHello.random + ServerHello.random)
                    [0..47];
```

```
key_block = PRF(SecurityParameters.master_secret,
                "key expansion",
                SecurityParameters.server_random +
                SecurityParameters.client_random);
```

Version Rollback Attack (SSL case)

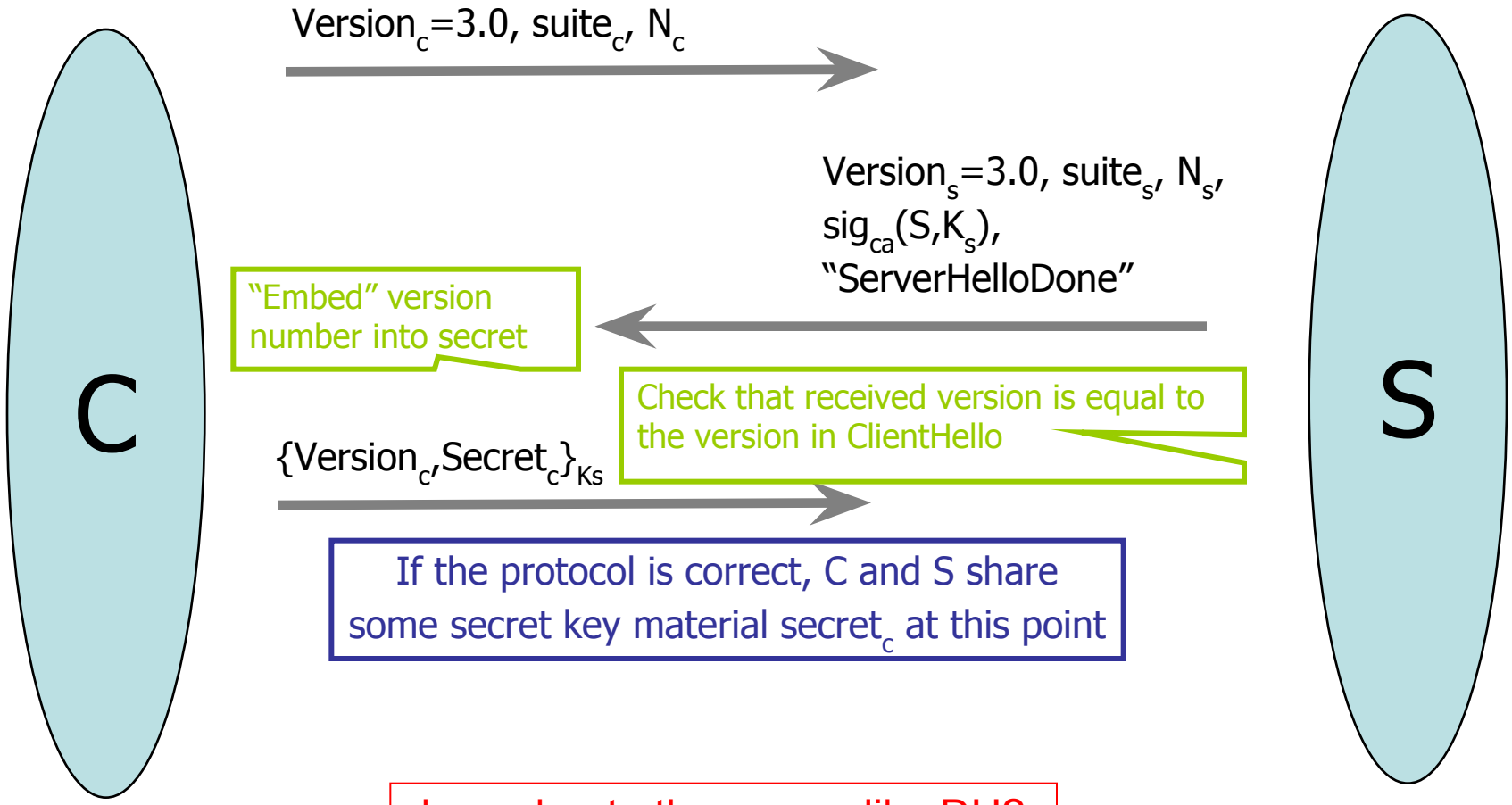


C and S end up communicating using SSL 2.0 (weaker earlier version of the protocol that does not include "Finished" messages)

aka“Chosen-Protocol” Attacks

- Why do people release new versions of security protocols? Because the old version got broken!
- New version must be backward-compatible
  - Not everybody upgrades right away
- Attacker can fool someone into using the old, broken version and exploit known vulnerability
  - fool a victim into using weak crypto algorithms
- Defense is hard: must authenticate version early
- aka downgrade attacks
- Many protocols had “version rollback” attacks
  - SSL, SSH, GSM

Version Check in SSL 3.0



Verify all the plaintext messages

- add “finished” messages
- Client and Server exchange “finished” messages in the end of handshake
  - It contains the hash of all the exchanged messages
- To thwart any attempt of tampering messages in the middle
- If the two hashes do not match, there has been message modification during the TLS handshake

forward secrecy

- session keys will not be compromised even if long-term secrets used in the session key exchange are compromised.
- Prevents an NSA-style attack
  - Store all the TLS traffic starting from TLS handshake
  - Get the server's private key later with a court order, or a bribe, or by hacking in
  - Decrypt all the stored traffic
- no forward secrecy until TLS 1.2
- Solution: use only DHE_* ciphers in TLS 1.3
  - Diffie-Hellman ephemeral (DHE)

TLS 1.3 has only DHE, ECDHE and PSK for generating session keys

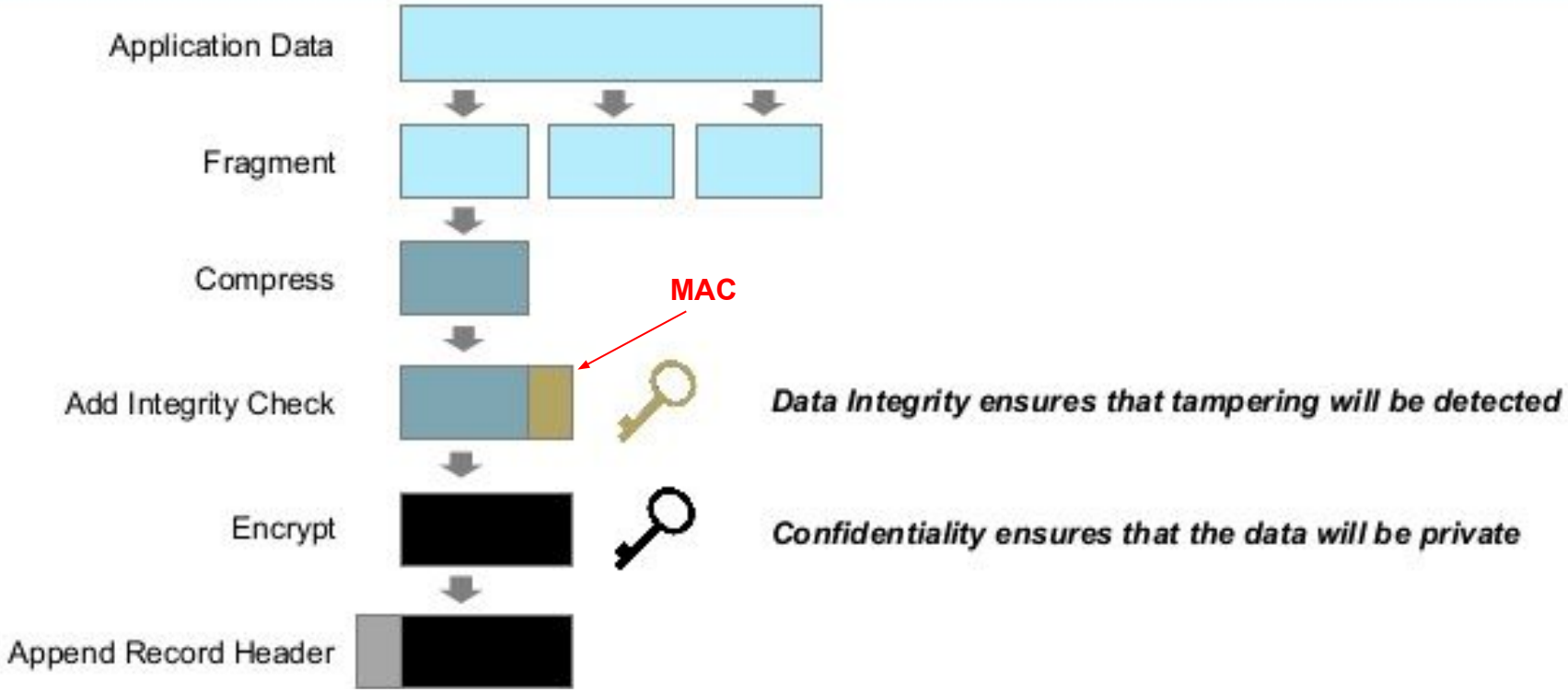
RSA, DH_RSA, DHE_RSA

- RSA
  - In the prior message flow
- DH_RSA

  - Server's "permanent" key pair is a DH key pair
    - certificate should have server's DH public key
      - g^S mod p, where s is the server's private key
    - cert has CA's signature using RSA
  - Client's sends her DH public key (g^C mod p)
- DHE_RSA
  - server generates a random number S (for each TLS session)
  - g^S mod p from server
    - signed by server's private key in cert (say, RSA): prevent MITM
  - g^C mod p from client
  - Ephemeral keys (S, g^{CS} mod p) are discarded in the server after session
  - Forward secrecy is achieved!

TLS Record protocol (until TLS 1.2)

- keys for MAC and for encryption are different



Source: Philippe De Ryck, https://www.websec.be

Other TLS/SSL Protocols

- Alert protocol.
  - Management of SSL/TLS session, error messages.
  - Fatal errors and warnings.
- Change cipher spec protocol.
  - Not part of Handshake Protocol.
  - Used to indicate that entity is changing to recently agreed ciphersuite.
- Both protocols run over Record Protocol

TLS 1.3

- faster speeds
  - reduced RTTs
  - no compression
- improved security
  - some handshake messages are encrypted
  - forward secrecy
  - remove insecure cipher suites
    - DES, RC4,...

TLS1.3 handshake

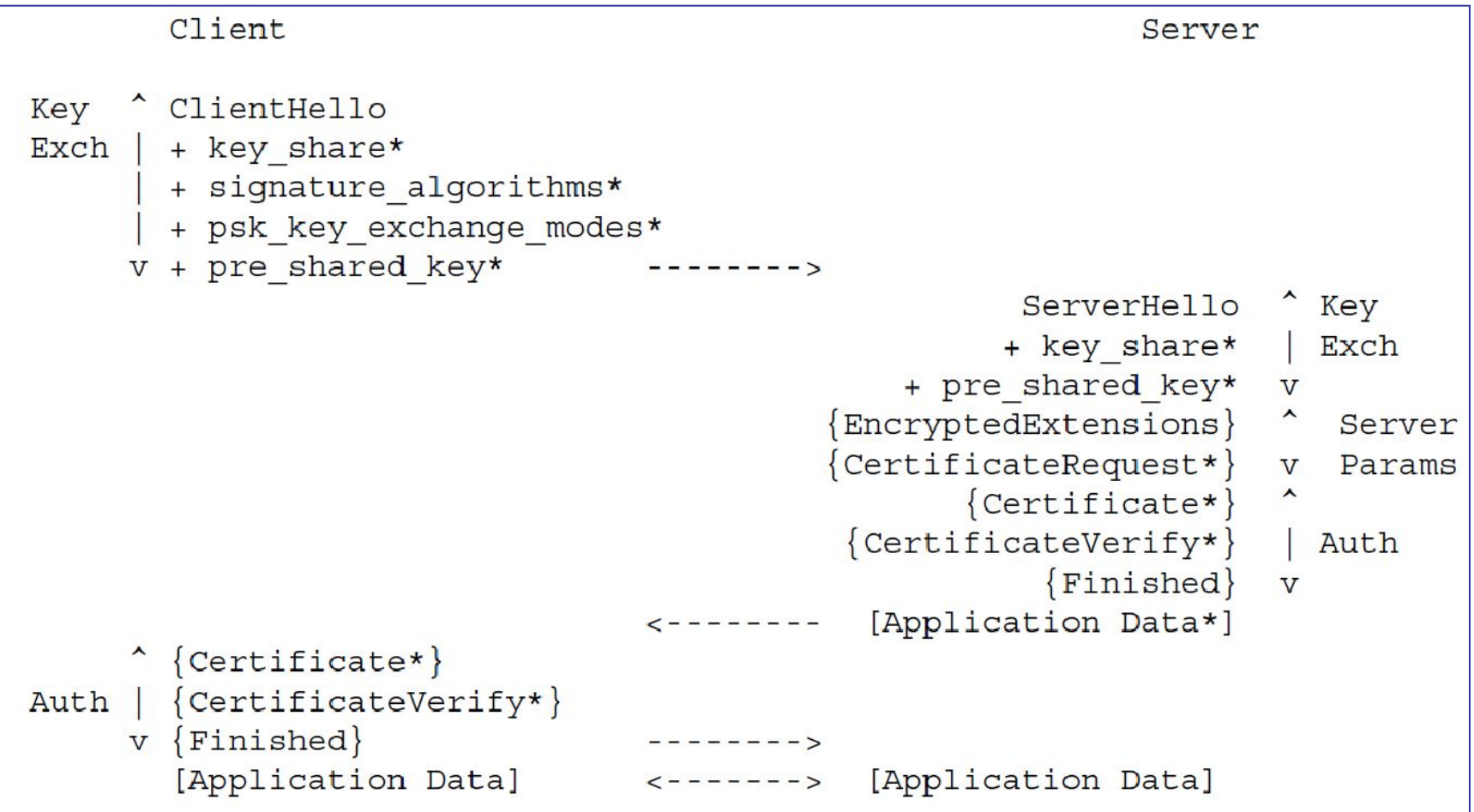
- 2 RTTs → 1 RTT

```
+  Indicates noteworthy extensions sent in the
   previously noted message.

*  Indicates optional or situation-dependent
   messages/extensions that are not always sent.

{} Indicates messages protected using keys
   derived from a [sender]_handshake_traffic_secret.

[] Indicates messages protected using keys
   derived from [sender]_application_traffic_secret_N
```



0-RTT

- resumption
- replay attack!

```
+   Indicates noteworthy extensions sent in the
    previously noted message.

*   Indicates optional or situation-dependent
    messages/extensions that are not always sent.

()  Indicates messages protected using keys
    derived from a client_early_traffic_secret.

{}  Indicates messages protected using keys
    derived from a [sender]_handshake_traffic_secret.

[]  Indicates messages protected using keys
    derived from [sender]_application_traffic_secret_N
```

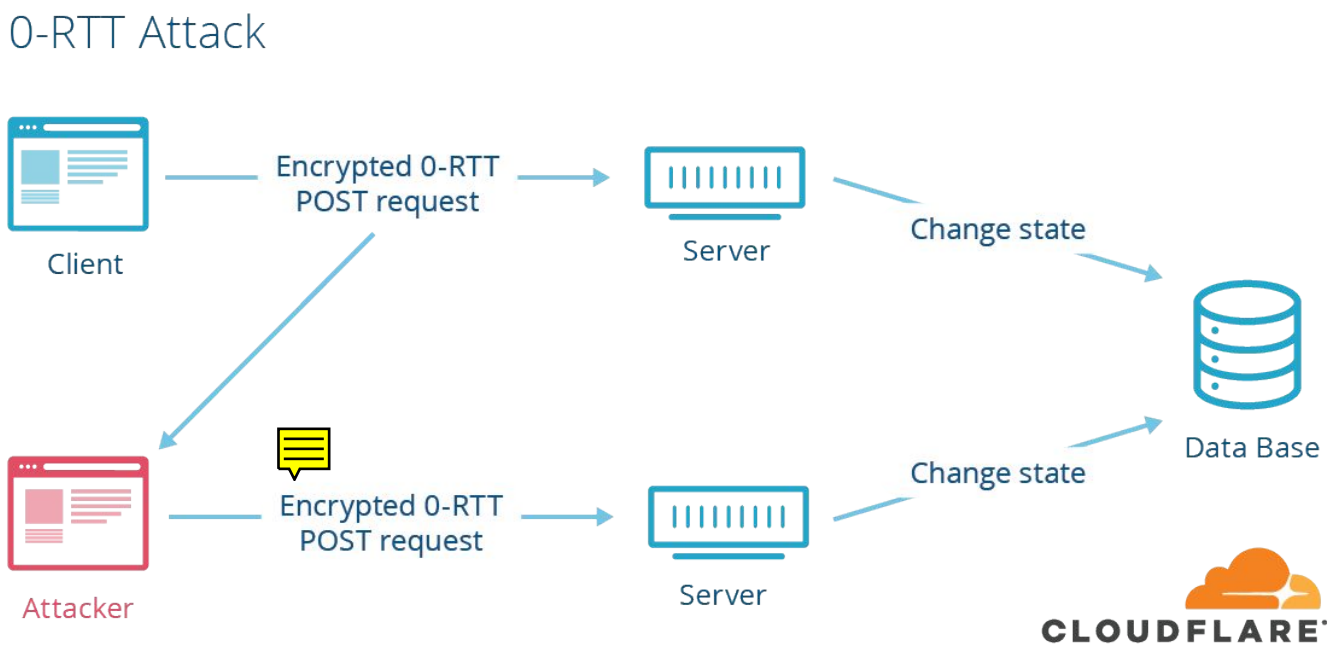
```
ClientHello
+ early_data
+ key_share*
+ psk_key_exchange_modes
+ pre_shared_key
(Application Data*)     -------->

                                        ServerHello
                                   + pre_shared_key
                                       + key_share*
                              {EncryptedExtensions}
                                   + early_data*
                                          {Finished}
                        <--------      [Application Data*]

(EndOfEarlyData)
{Finished}              -------->
[Application Data]      <-------->      [Application Data]
```

0-RTT: replay attack

- Different servers cannot catch reply attacks!
- Initial data in TLS resumption should be carefully handled
  - Say, only HTTP GET messages without any parameters are accepted



Source: https://blog.cloudflare.com/rfc-8446-aka-tls-1-3/