5. Transport layer security (TLS)

Many slides from Jinyuan Sun@U. of Tennessee

http vs https

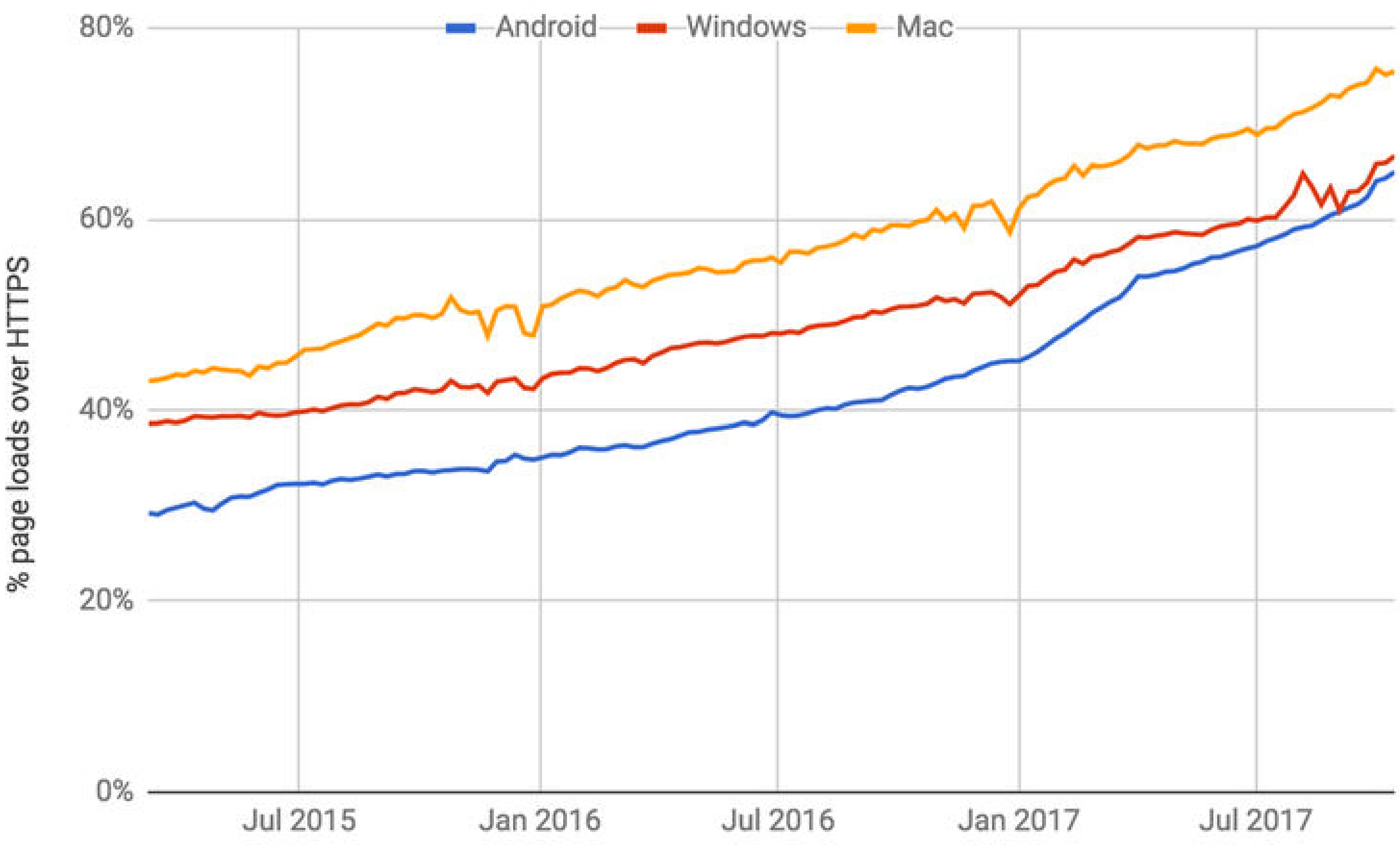
http

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

https

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

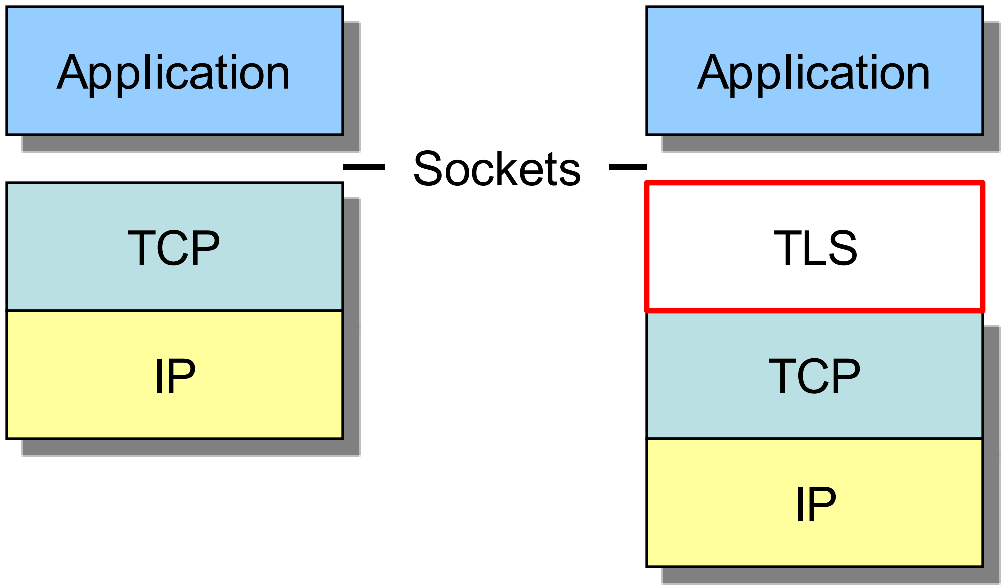
Portion of https traffic



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
- Deployed in every Web browser

Application-Level Protection



Insecure Transport Layer

Secure Transport Layer

Source: Andreas Steffen@ITA

History of the Protocol

- 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
  - <u>Not</u> interoperable with SSL 3.0
    - TLS uses HMAC instead of MAC; can run on any port

TLS history

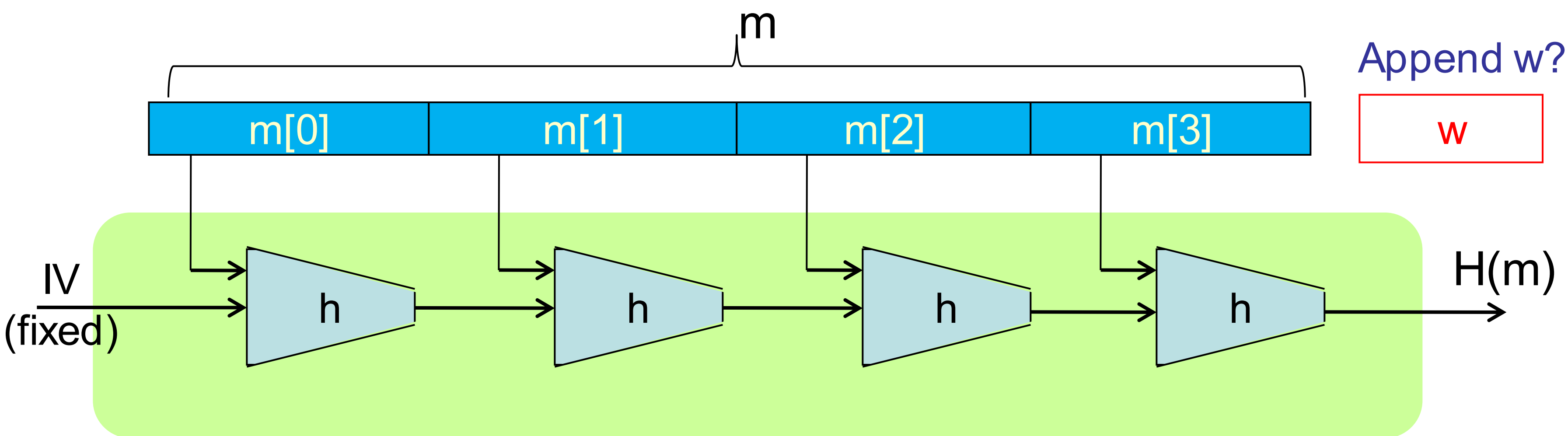
- TLS 1.1, 2006
  - RFC 4346
  - Protection against cipher-block chaining (CBC) attacks
- TLS 1.2, 2008
  - RFC 5246
  - More options in cipher suite
    - Eg. SHA 256, AES-related
- TLS 1.3, 2018
  - Published as RFC 8446
  - Some insecure ciphers removed (RC4, DES,…)
  - streamline RTT handshakes (e.g.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

HMAC: Constructing MAC from Hash Fn.

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

Merkel Damgård construction

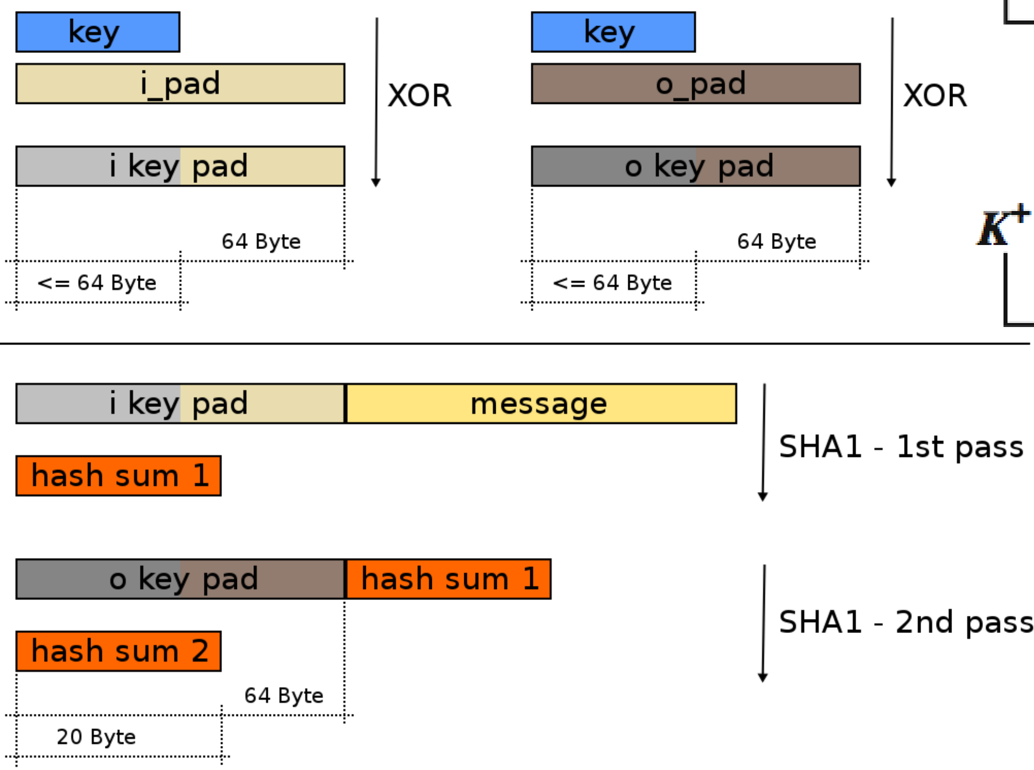
e.g. h is a compression fn. like MD5 – 512 bit block

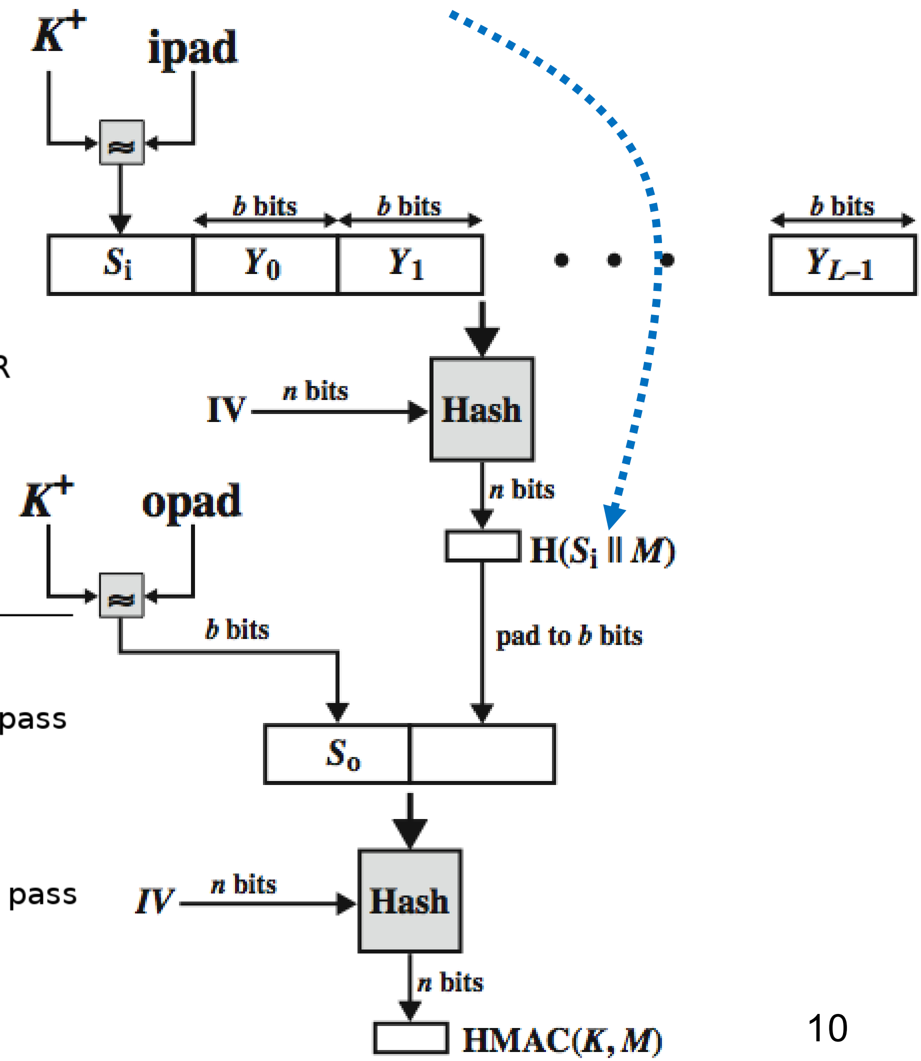


- Assume key is already prepended into m
  - Secret||original_msg = m
- Attacker doesn't know secret or original_msg
- Yet she wishes to append w after m
- What if string w is appended after m?
- h(H(m),w) vs. H(m||w)

Hash-based MAC (HMAC)

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



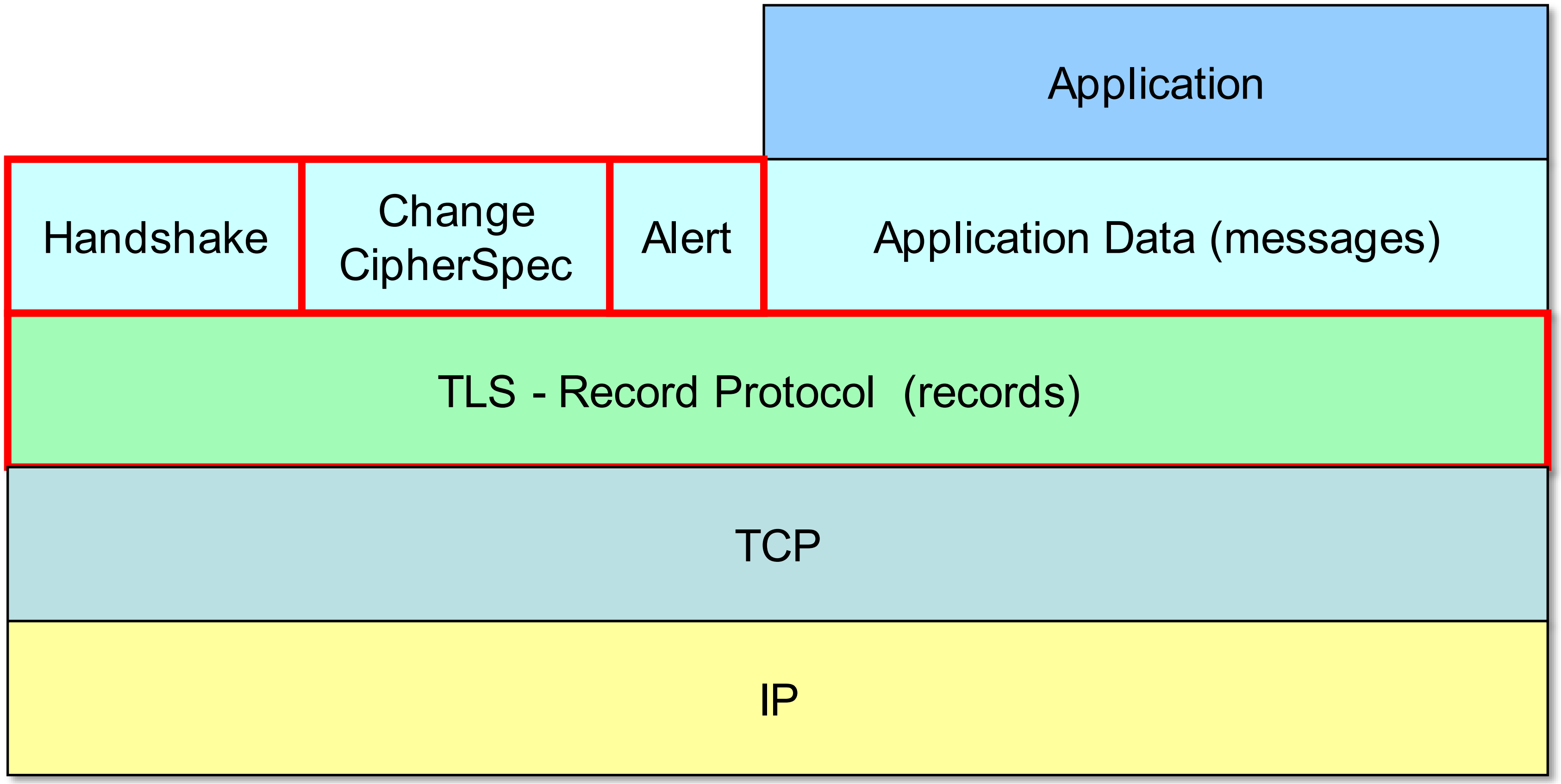


Source: wikipedia

TLS Basics

- TLS consists of two main protocols
  - Familiar pattern for key exchange 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 (optional)
  - Use digital certificates to learn each other's public keys and verify each other's identity
- Use public keys to establish a shared secret
- Symmetric key is generated from the secret
- The following is based on TLS 1.1 & 1.2

Handshake + ChangeCipherSpec

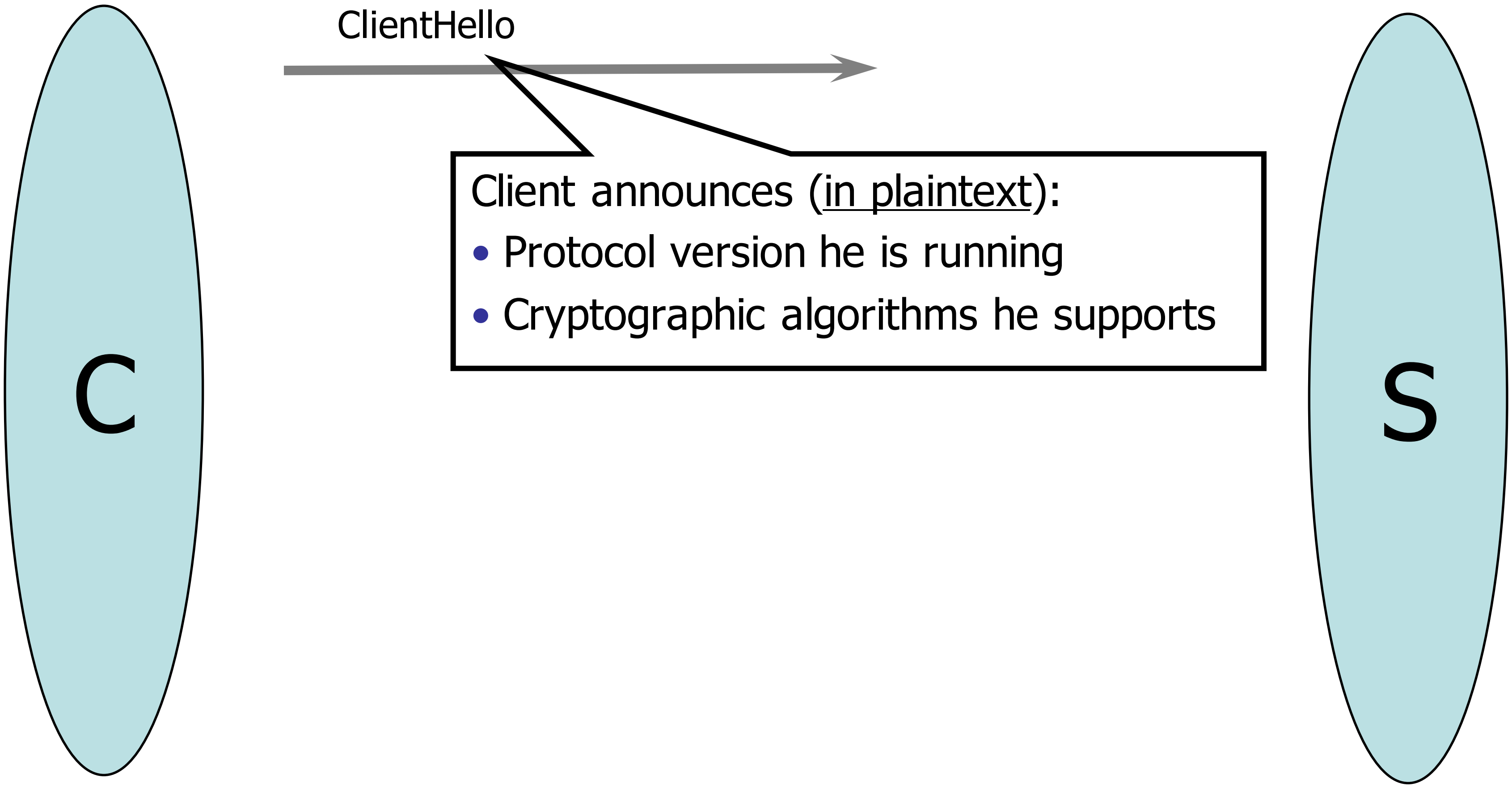
```
Client                                               Server

ClientHello                  -------->
                                                ServerHello
                                               Certificate*
                                         ServerKeyExchange*
                                        CertificateRequest*
                             <--------      ServerHelloDone
Certificate*
ClientKeyExchange
CertificateVerify*
[ChangeCipherSpec]
Finished                     -------->
                                         [ChangeCipherSpec]
                             <--------             Finished
Application Data             <------->     Application Data

         Figure 1.  Message flow for a full handshake
```

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

ClientHello



The following message flows are common from SSL 3.0 to TLS 1.2

ClientHello

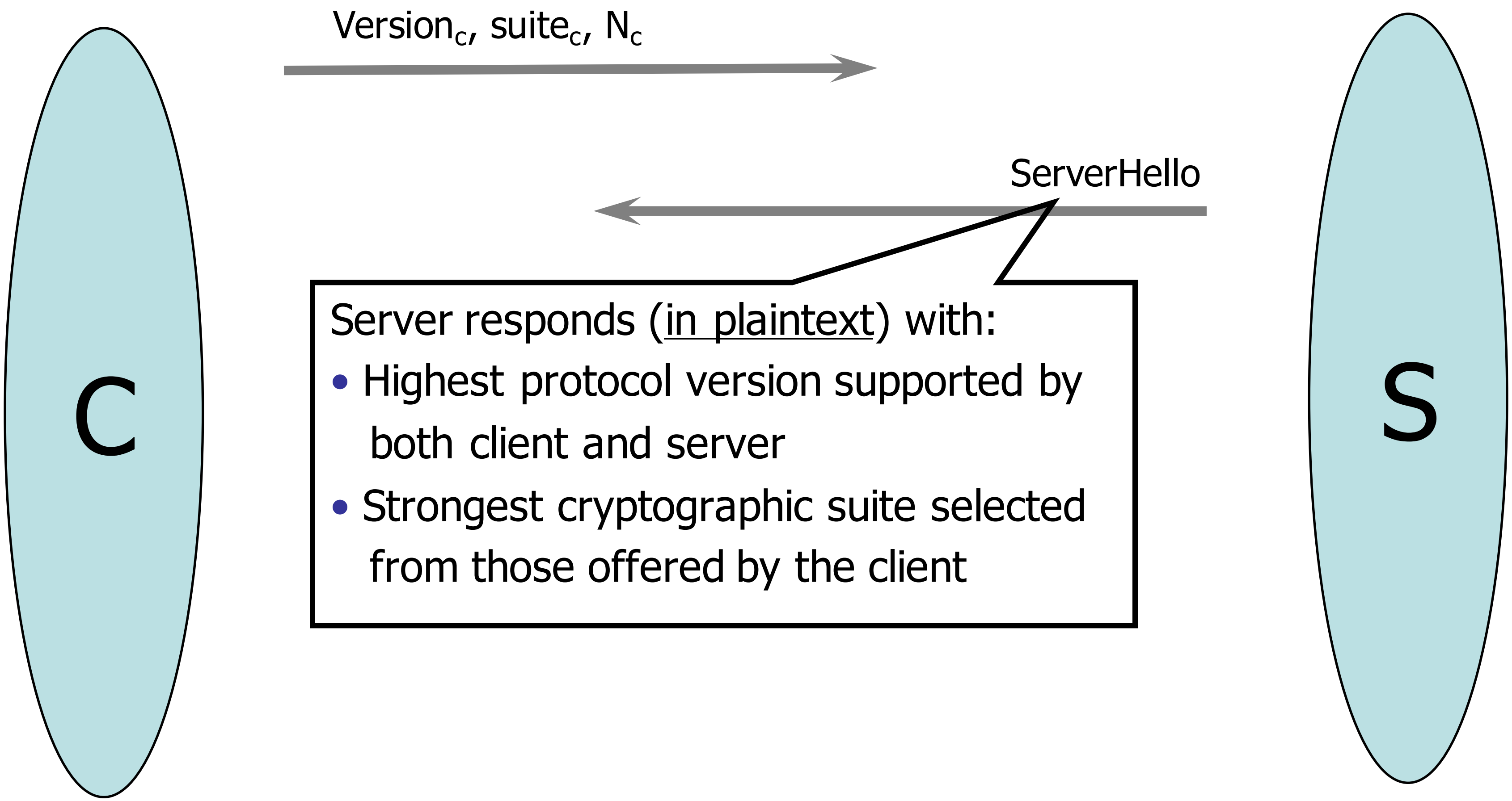
```
struct {
    ProtocolVersion client_version;
    Random random;
    SessionID session_id;
    CipherSuite cipher_suites;
    CompressionMethod
     compression_methods;
} ClientHello
```

Highest version of the protocol supported by the client

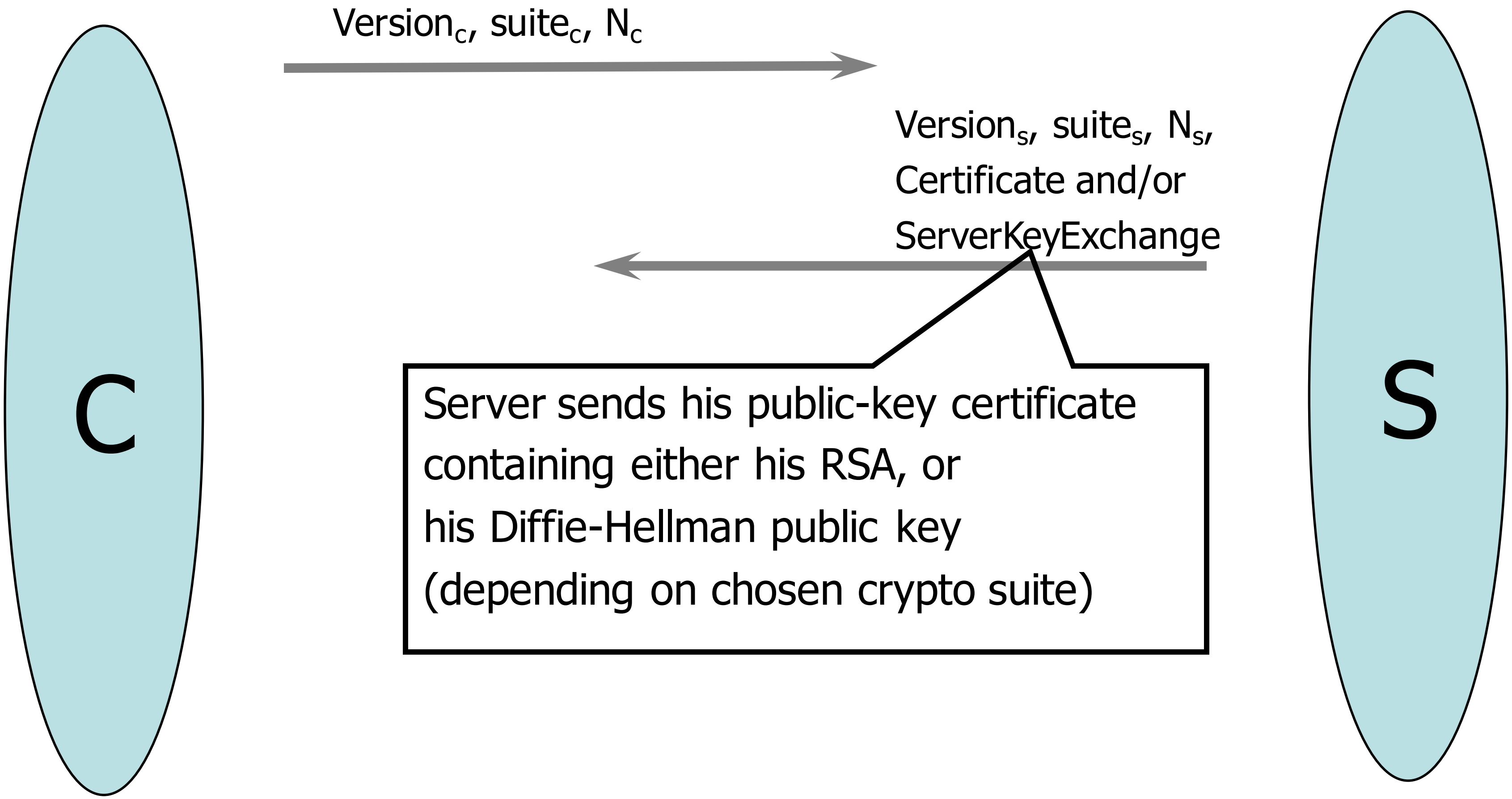
Session id (if the client wants to resume an old session)

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

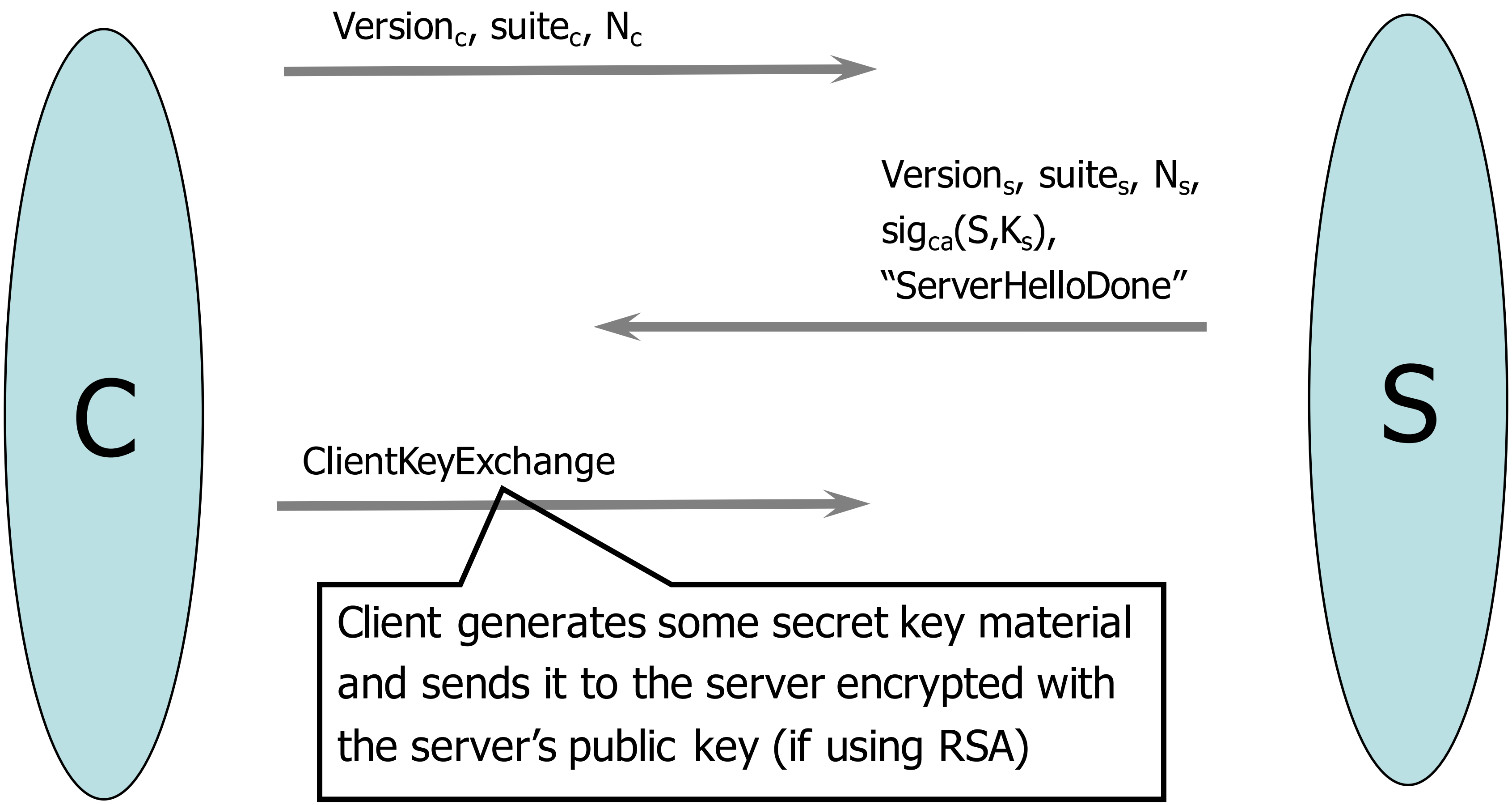
ServerHello



Certificate and/or ServerKeyExchange

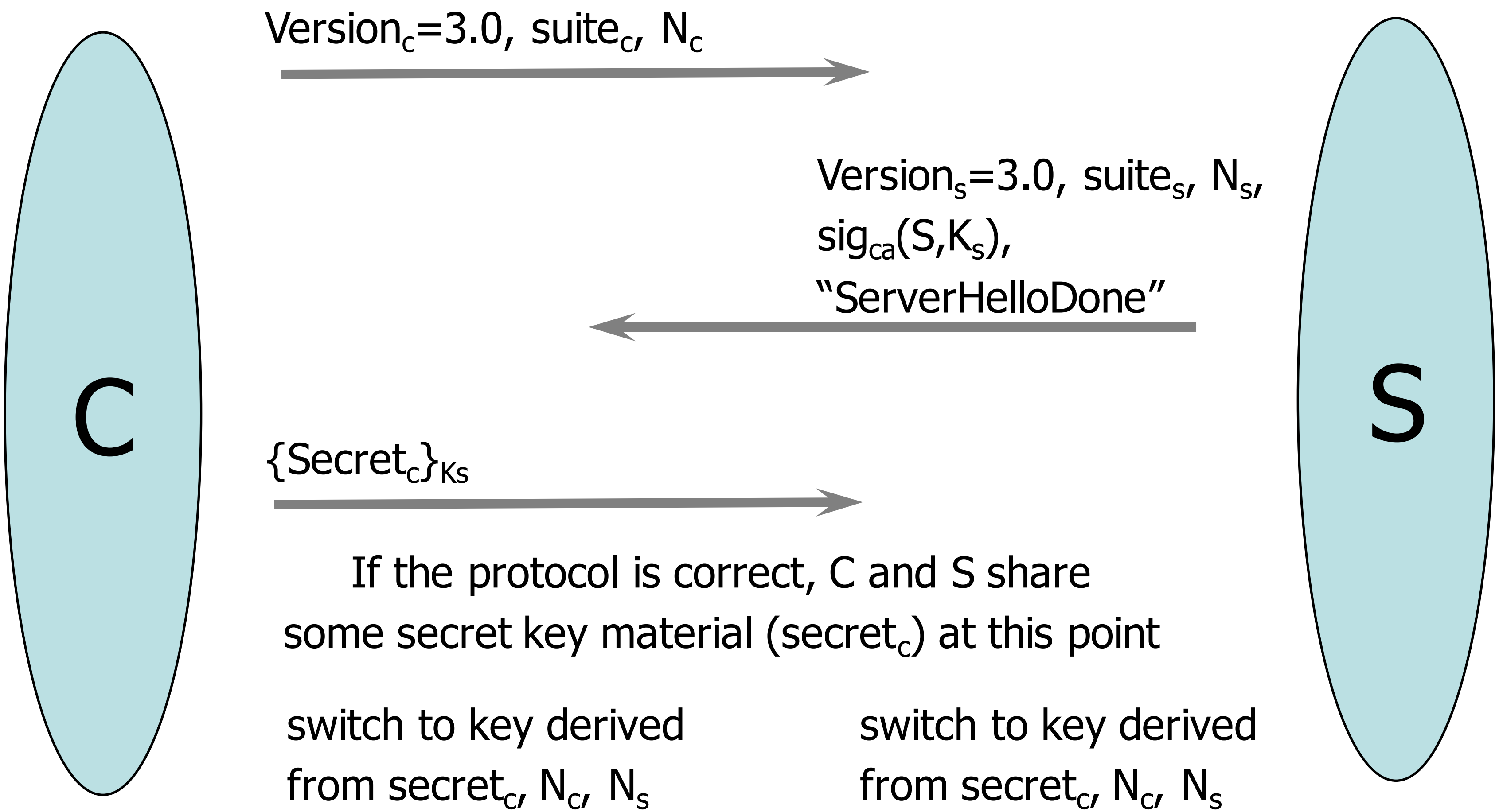


ClientKeyExchange



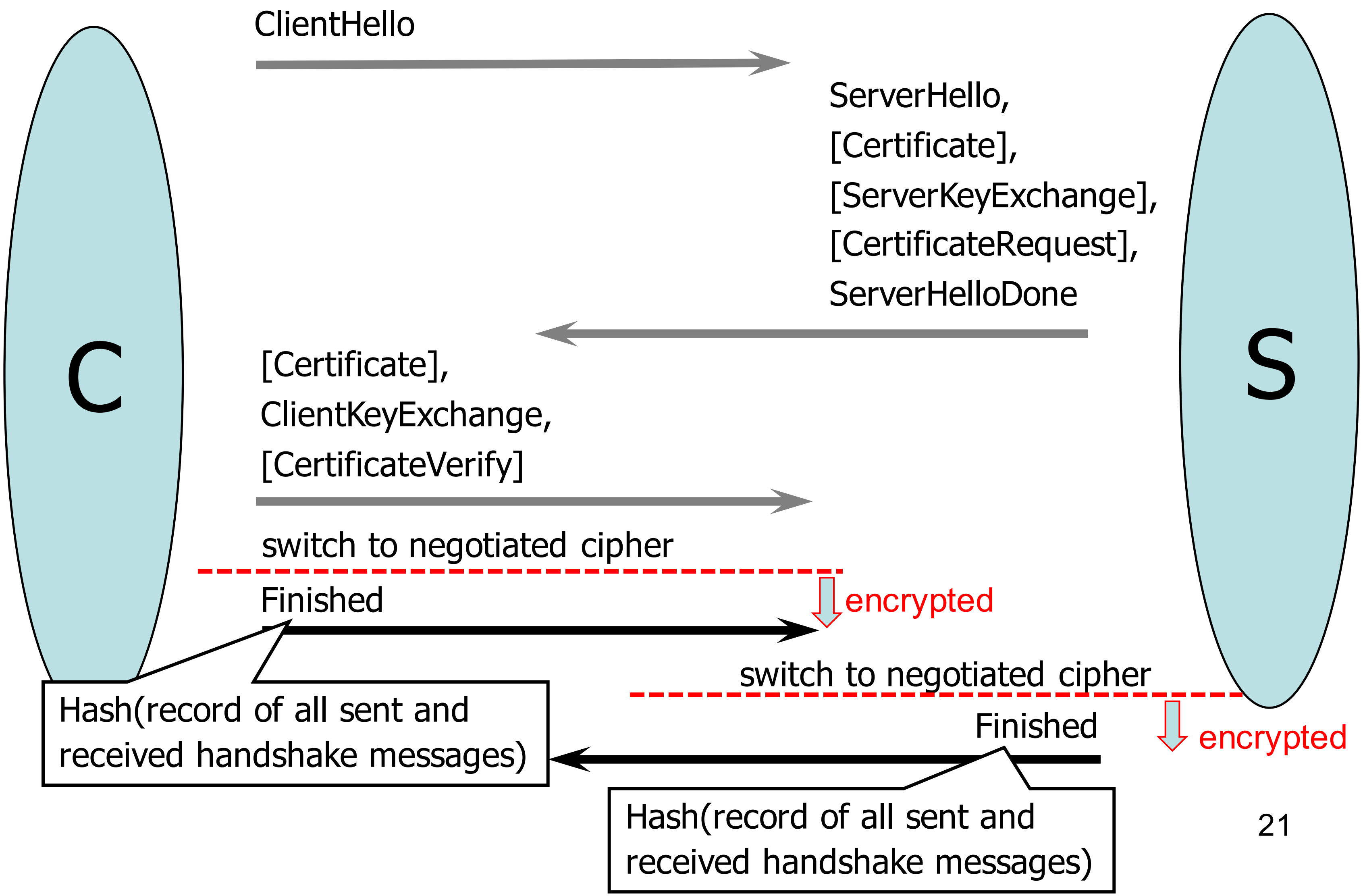
S: server identity, K_s : server's public key, $sig_{ca}(\ )$: CA's signature cert

Handshake: server certificate with RSA



$secret_c$: premaster secret, K_s : server's public key

Handshake Protocol Structure

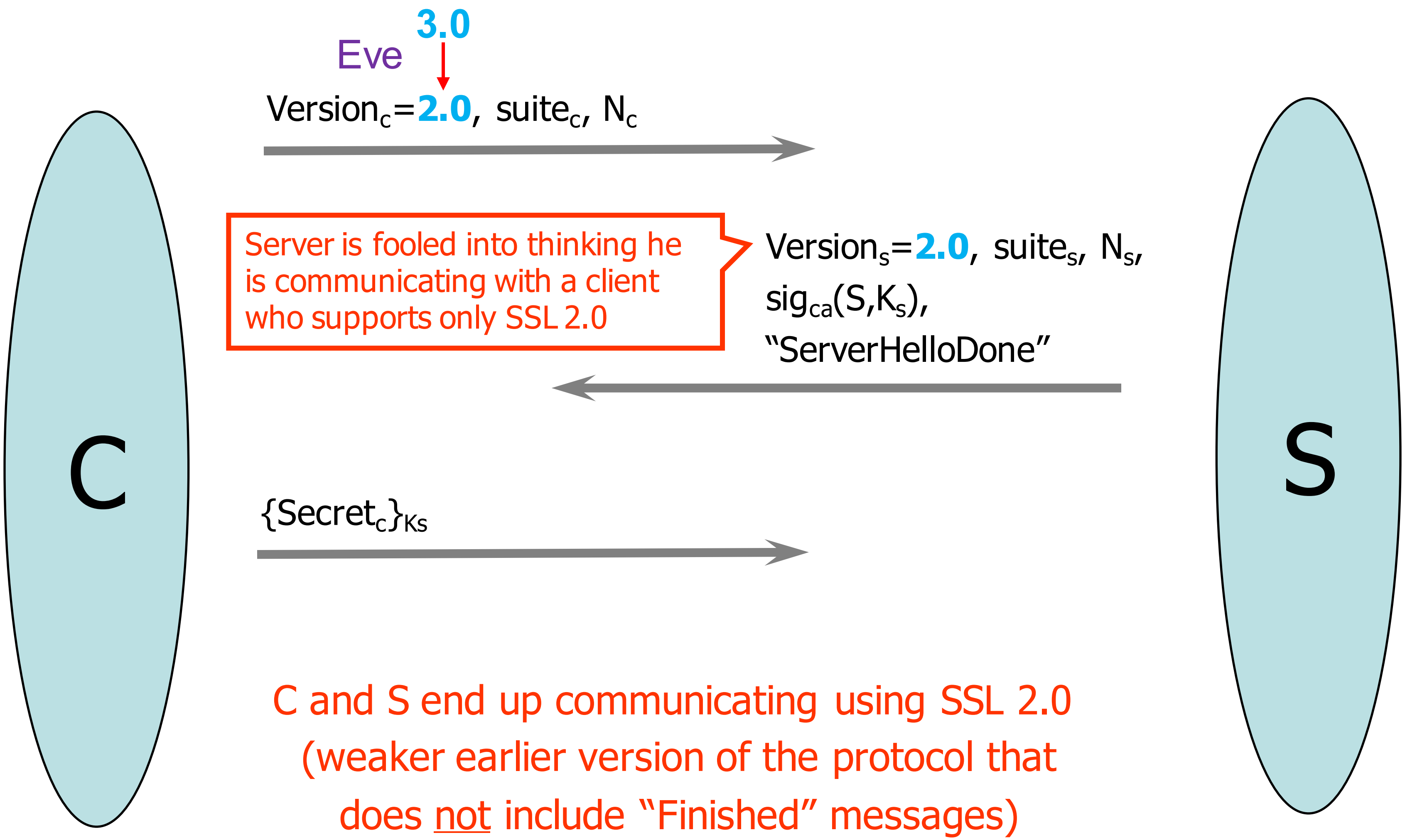


Generating master secret & keys

```
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);
```

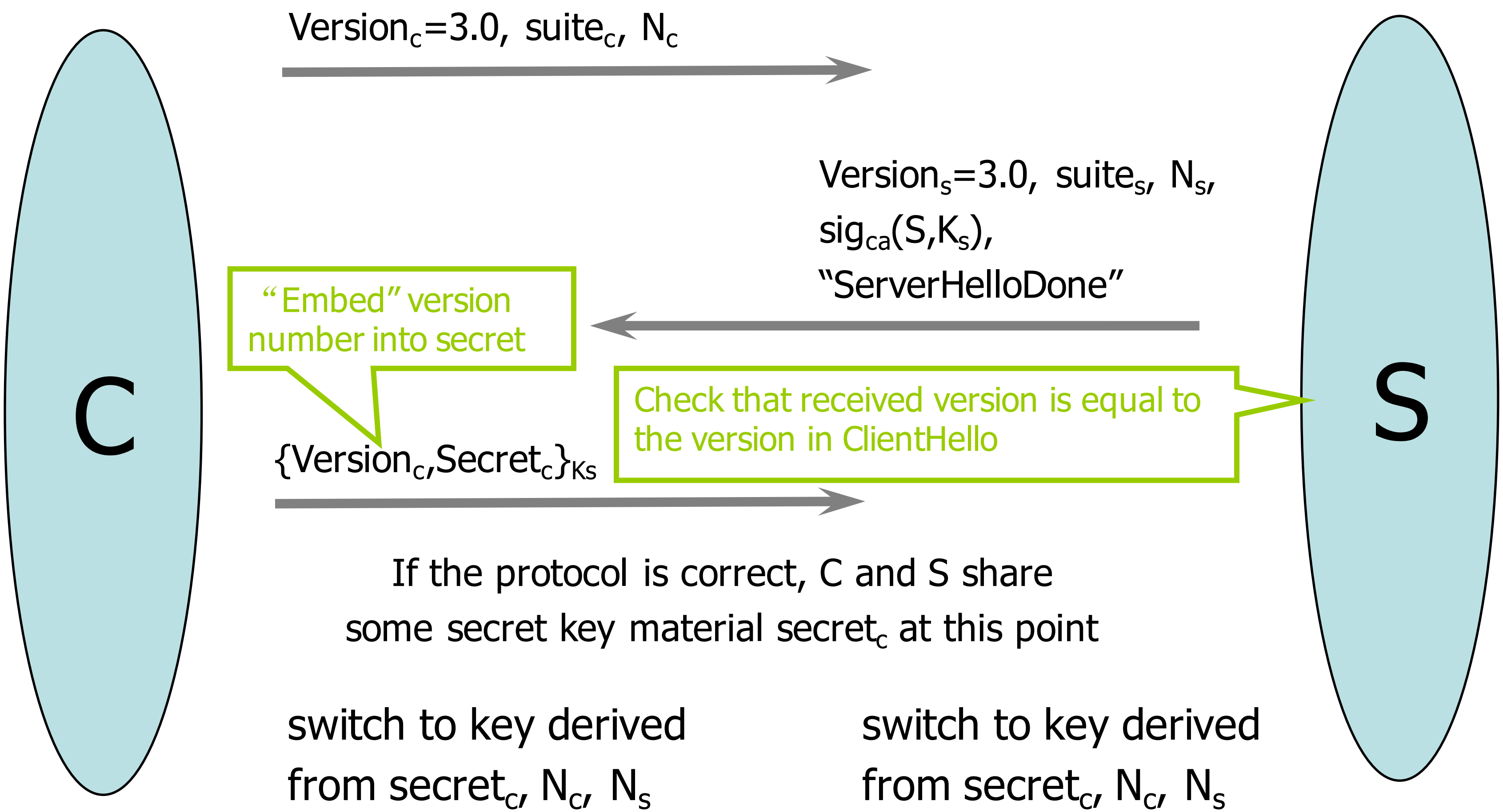
Version Rollback Attack (SSL case)



"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
  - Similar: fool victim into using weak crypto algorithms
- Defense is hard: must authenticate version early
- Many protocols had "version rollback" attacks
  - SSL, SSH, GSM (cell phones)

Version Check in SSL 3.0



In TLS 1.2, Finished message has the hash of all handshake messages

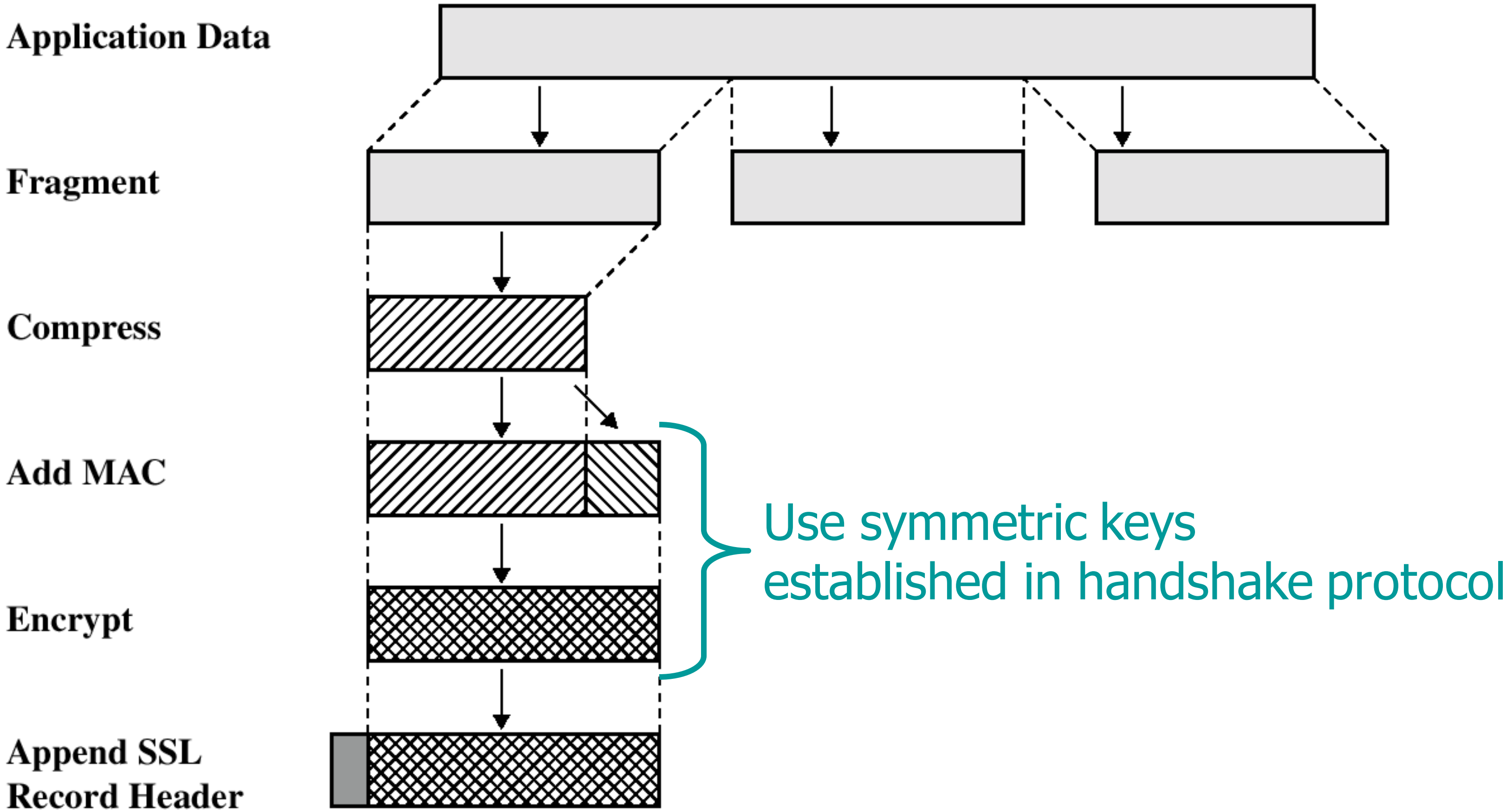
forward secrecy

- Uses a different key for each session
- Prevents an NSA-style attack
  - Store all the traffic to an encrypted site
  - Get the server's private key later with a court order, or a bribe, or by hacking in
  - Decrypt all the stored traffic
- Solution: DHE_* ciphers available in TLS
  - Diffie-Hellman ephemeral (DHE)

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 a DH key pair (not RSA)
  - Client’s sends her DH public key (g^C mod p)
  - CA sign is generated by RSA
- DHE_RSA
  - g^S mod p from server
    - signed by RSA private key: prevent MITM
  - g^C mod p from client
  - Ephemeral keys (g^S mod p & g^{CS} mod p) are discarded after session
  - Forward secrecy!

SSL/TLS Record Protection



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
- improved security
  - some handshake messages are encrypted
  - forward secrecy
    - session ID is obsoleted
  - remove insecure cipher suites
    - even RSA!! (RSA cert is fine)

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
```

```
       Client                                           Server

Key  ^ ClientHello
Exch | + key_share*
     | + signature_algorithms*
     | + psk_key_exchange_modes*
     v + pre_shared_key*       -------->
                                                  ServerHello  ^ Key
                                                 + key_share*  | Exch
                                            + pre_shared_key*  v
encrypted                               {EncryptedExtensions}  ^  Server
                                        {CertificateRequest*}  v  Params
                                               {Certificate*}  ^
                                         {CertificateVerify*}  | Auth
                                                   {Finished}  v
                               <--------  [Application Data*]
     ^ {Certificate*}
Auth | {CertificateVerify*}
     v {Finished}              -------->
       [Application Data]      <------->  [Application Data]
```

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.
```

NewSessionTicket (from server)

```
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]
```