THE WORLD OF TLS

Security, Attacks, TLS 1.3

HTTPS:// AND FTPS:// AND….

- Have you done any of the following today?
  - E-shopping: Amazon, Ebay, Audible, …
  - Checked your Email
  - Visited a social networking site: Facebook, Twitter, …
  - Used a secure FTP
  - Used Voice over IP
  - Used Google
  - Used any URL strting with https:// and a green lock

Congratulations, you used TLS/SSL!

PART 1
ABOUT TLS/SSL

WHAT TLS DOES

- Main goal:
  - Confidentiality and Authenticity of communications
  - Privacy of data and services exchanged
    - Your searches on Google, or even the fact that you used Google Search rather than Google mail
  - Guarantees still work if keys are compromised (PFS)
  - Mostly Client (you) ↔ Server (Service Provider)
- How TLS does this:
  - Key Exchange: yields keys for SEnc and MAC
  - Record layer: use authenticated encryption with keys to secure communication
  - Authentication: usually only server side (eases PKI)

THE CLIENT-SERVER SCENARIO

- Online shopping:
  - You go to amazon.fr
  - You choose what you want to buy
  - Put it in your virtual shopping cart
  - Log in with your user name and password
  - Pay
  - Wait for your delivery
- What actually happens:
  - You type amazon.fr in your browser
  - Your browser negotiates a TLS connection with Amazon
  - You get to the website on https:// for secure browsing
  - You authenticate to amazon on a secure link

A BIT OF HISTORY

- Started out as Secure Socket Layer (SSL)
  - Developed by Netscape around 1995
  - Main goal: secure communication over the Internet
- Changed to Transport Layer Security (TLS) in 1999
  - Secure communication over the Internet: https
  - … but also: secure file sharing (ftp), secure emailing etc.
  - Heavily standardised
- Some implementations:
  - OpenSSL
  - BoringSSL, mbedTLS
  - s2n: TLS by Amazon

BIT OF A BLACK SHEEP

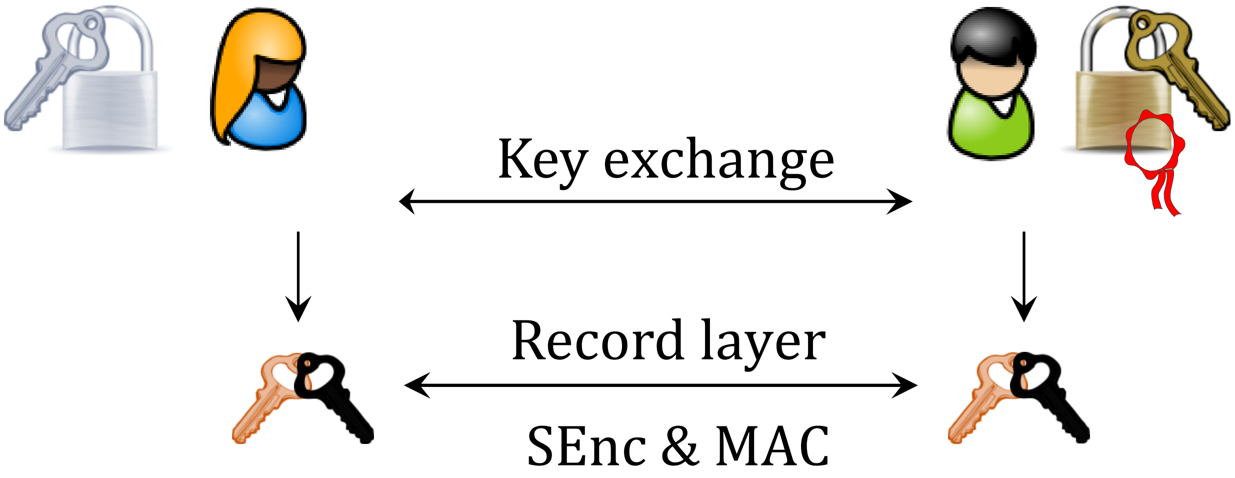
- SSL 1.0: never released (too insecure for release)
- SSL 2.0: released in Feb. 1995

  "contained a number of security flaws"
- SSL 3.0: released in 1996, complete re-design from 2.0

- TLS 1.0: "no dramatic changes", but "more secure"

  backward compatible: can relax to SSL 3.0
- TLS 1.1: some protection against CBC-mode attacks: explicit IV, better padding
- TLS 1.2: problems with MD5, more recently RC4 renegotiation, export ciphersuites, implem. faults

BACKGROUND: TLS/SSL



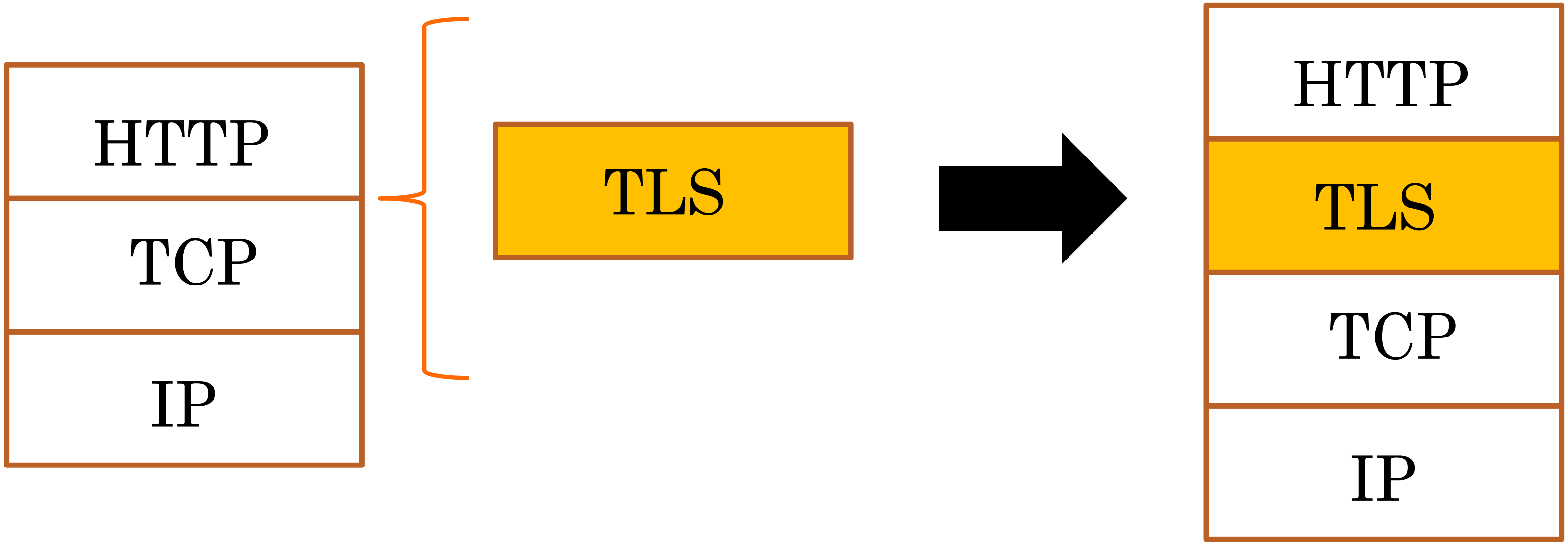
Key exchange

Record layer

SEnc & MAC

- Intuition:
  - If keys are “good”, they should secure Record layer
  - Q1: What is a “good” key?
  - Q2: How do we encrypt and authenticate?

TLS AS A COMMUNICATION LAYER



HTTP

TCP

IP

TLS

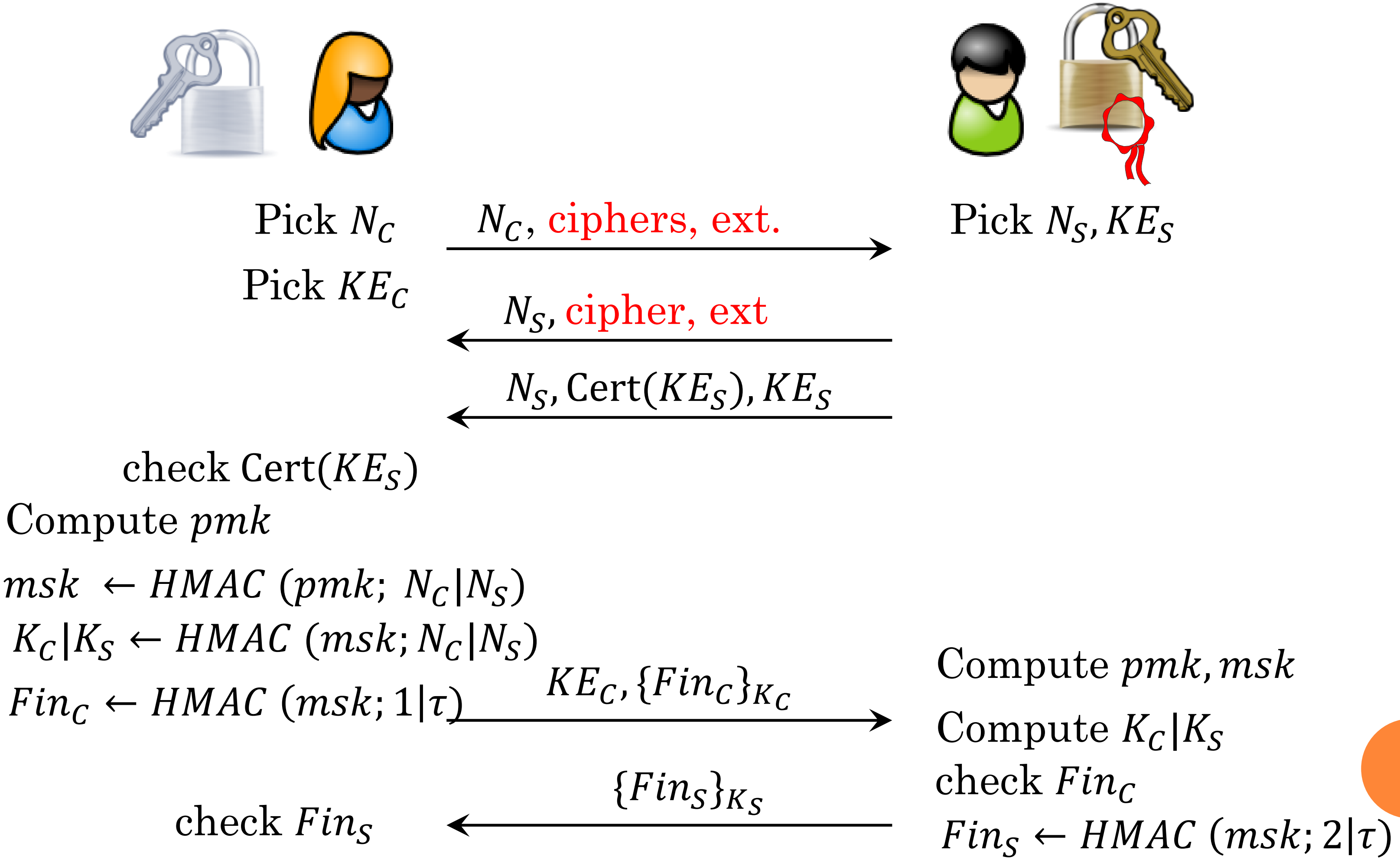
HTTP

TLS

TCP

IP

THE TLS (1.2) HANDSHAKE (AKE)



Pick N_C

Pick KE_C

N_C, ciphers, ext. $\rightarrow$

Pick N_S, KE_S

$\leftarrow$ N_S, cipher, ext

$\leftarrow$ $N_S, \text{Cert}(KE_S), KE_S$

check $\text{Cert}(KE_S)$

Compute pmk

$msk \leftarrow HMAC\ (pmk;\ N_C|N_S)$

$K_C|K_S \leftarrow HMAC\ (msk; N_C|N_S)$

$Fin_C \leftarrow HMAC\ (msk; 1|\tau)$

$KE_C, \{Fin_C\}_{K_C}$ $\rightarrow$

Compute pmk, msk

Compute $K_C|K_S$

check Fin_C

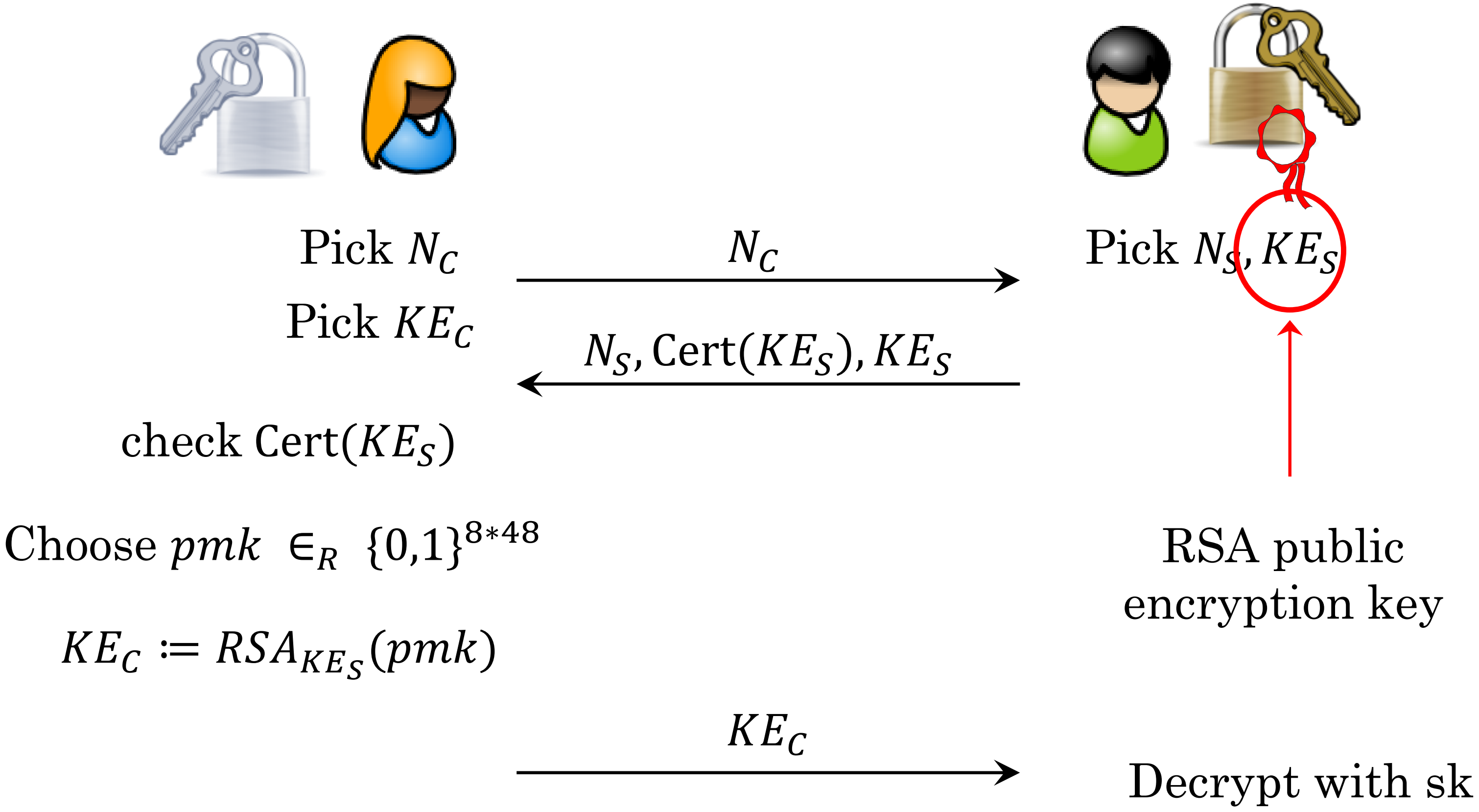
$Fin_S \leftarrow HMAC\ (msk; 2|\tau)$

$\leftarrow$ $\{Fin_S\}_{K_S}$

check Fin_S

THE THREE MODES

- TLS-RSA (most used):



Pick N_C $\xrightarrow{N_C}$ Pick N_S, KE_S

Pick KE_C

$\xleftarrow{N_S, \text{Cert}(KE_S), KE_S}$

check $\text{Cert}(KE_S)$

Choose $pmk \in_R \{0,1\}^{8*48}$

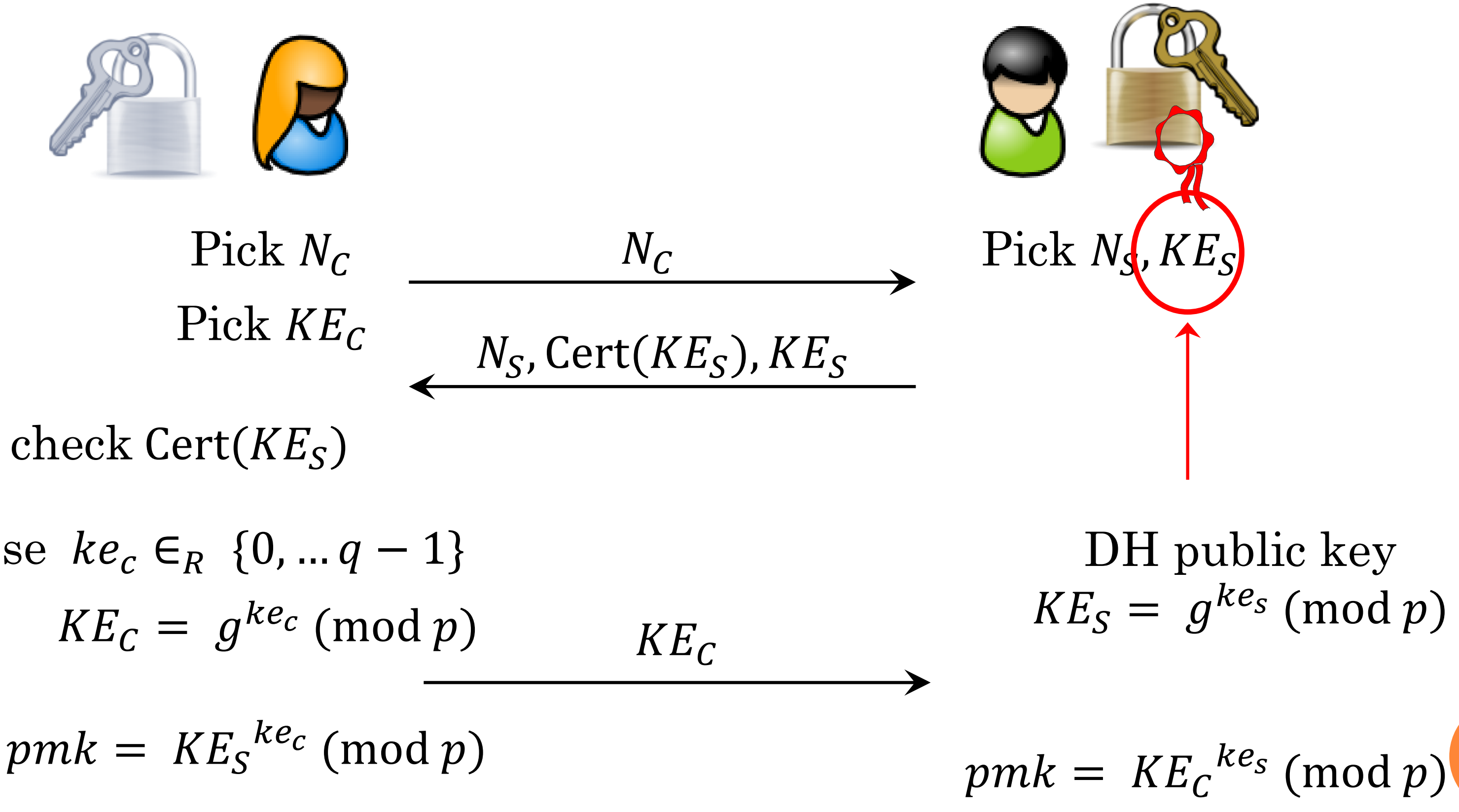
$KE_C := RSA_{KE_S}(pmk)$

RSA public
encryption key

$\xrightarrow{KE_C}$ Decrypt with sk

THE THREE MODES

- TLS-DH (second best):



Pick N_C

N_C

Pick N_S, KE_S

Pick KE_C

$N_S, \mathrm{Cert}(KE_S), KE_S$

check $\mathrm{Cert}(KE_S)$

Choose $ke_c \in_R \{0, \ldots q-1\}$

$KE_C = g^{ke_c} \pmod p$

KE_C

DH public key

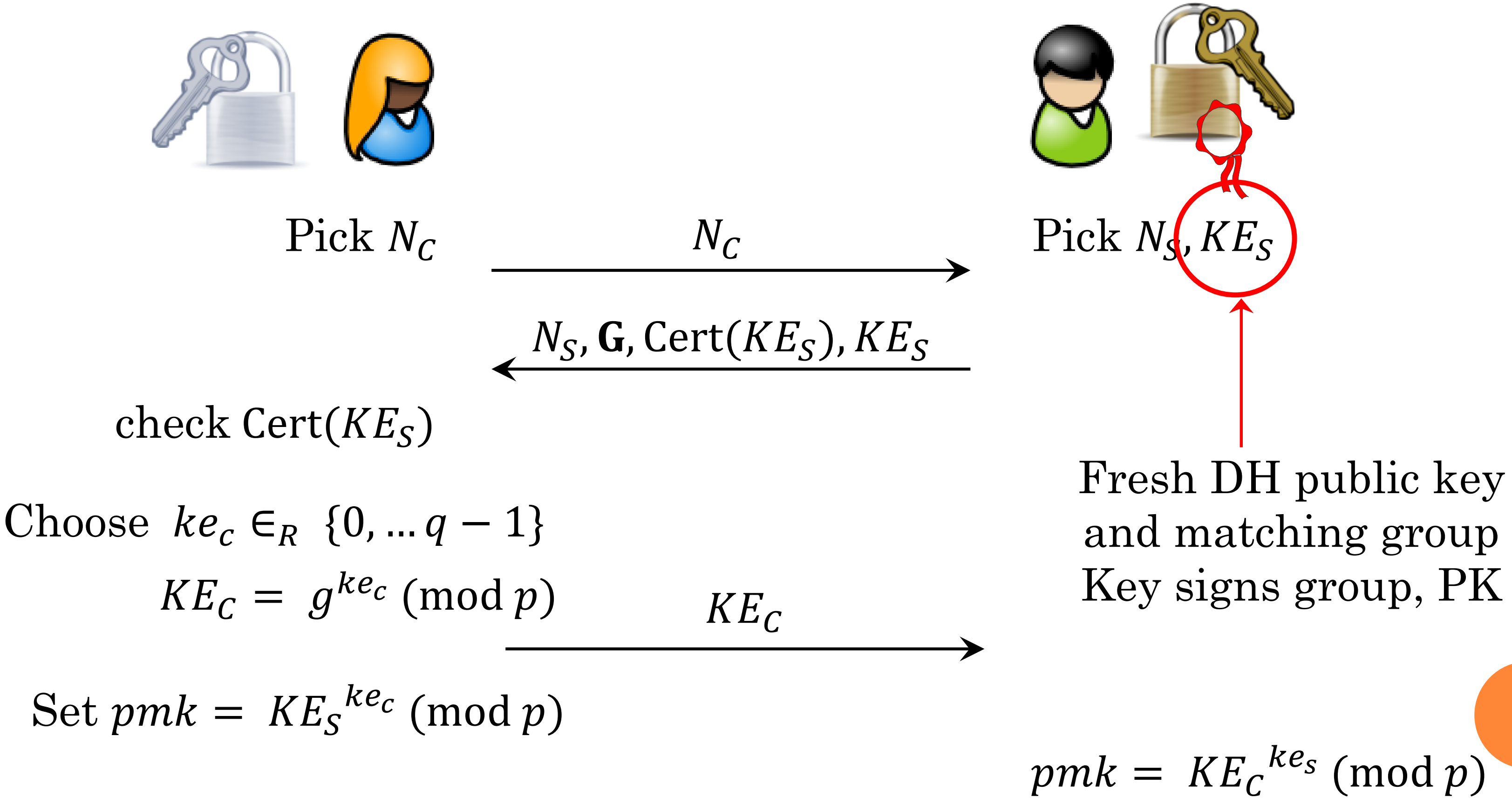
$KE_S = g^{ke_s} \pmod p$

Set $pmk = {KE_S}^{ke_c} \pmod p$

$pmk = {KE_C}^{ke_s} \pmod p$

THE THREE MODES

- TLS-DHE (ephemeral DH):



Pick N_C

N_C

Pick N_S, KE_S

$N_S, \mathbf{G}, \mathrm{Cert}(KE_S), KE_S$

check $\mathrm{Cert}(KE_S)$

Choose $ke_c \in_R \{0, \dots q-1\}$

$KE_C = g^{ke_c} \pmod{p}$

KE_C

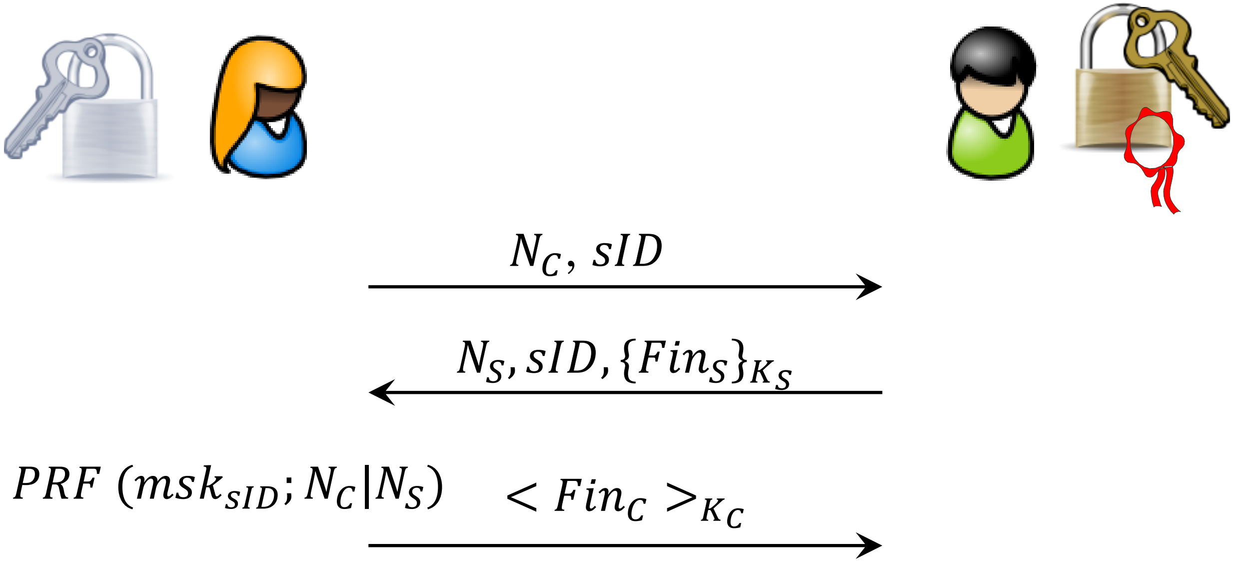
Fresh DH public key
and matching group
Key signs group, PK

Set $pmk = {KE_S}^{ke_c} \pmod{p}$

$pmk = {KE_C}^{ke_s} \pmod{p}$

KEY DERIVATION AND RENEGOTIATION

- Runs of TLS are “sessions” and have session IDs
  - If client has seen server before, reuse key material (msk)
  - Use sID instead of N_C and N_S



$N_C,\ sID$

$N_S, sID, \{Fin_S\}_{K_S}$

$K_C | K_S \leftarrow PRF\ (msk_{sID}; N_C | N_S)$

$< Fin_C >_{K_C}$

$Fin_C \leftarrow PRF\ (msk_{sID}; 1 | \tau)$

TLS HANDSHAKE SUMMARY

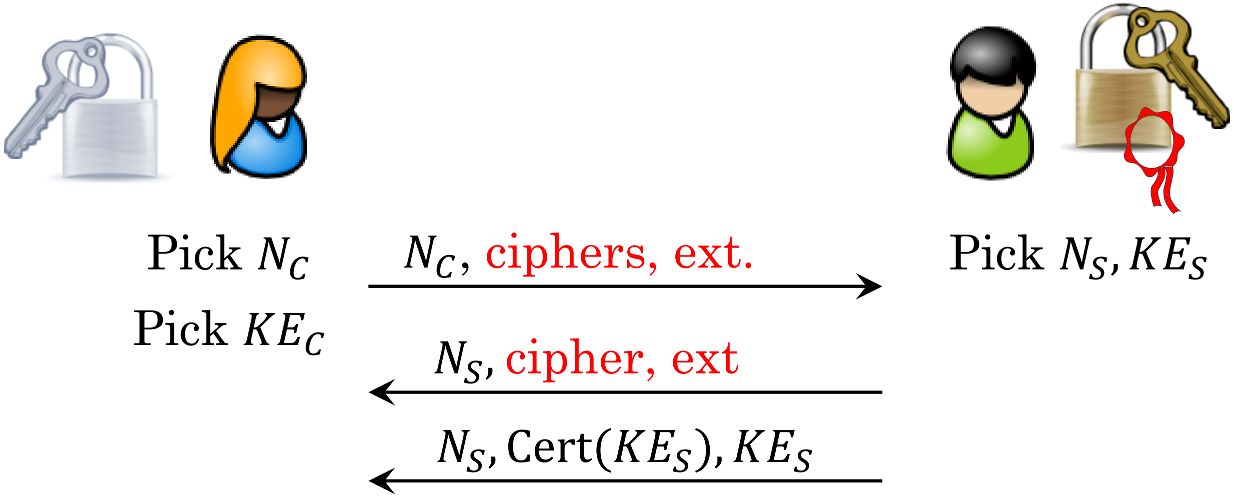
- Session freshness
  - Nonces N_C, N_S involved in key derivation

$$msk \leftarrow PRF\ (pmk;\ N_C | N_S)$$

  - Prevent replay attacks (that enforce same keys)
- Server authentication
  - Certificate ensures only server shares key with client
  - Unilateral: anyone can exchange keys with server
- Key confirmation
  - Finished messages: authenticated encryption with session keys, of a fixed message
  - Both parties are sure they computed the same keys
- Forward secrecy : only in DHE mode

SOME PROBLEMS

- Configuration parameters not part of key



Pick N_C

Pick KE_C

N_C, ciphers, ext. →

Pick N_S, KE_S

← N_S, cipher, ext

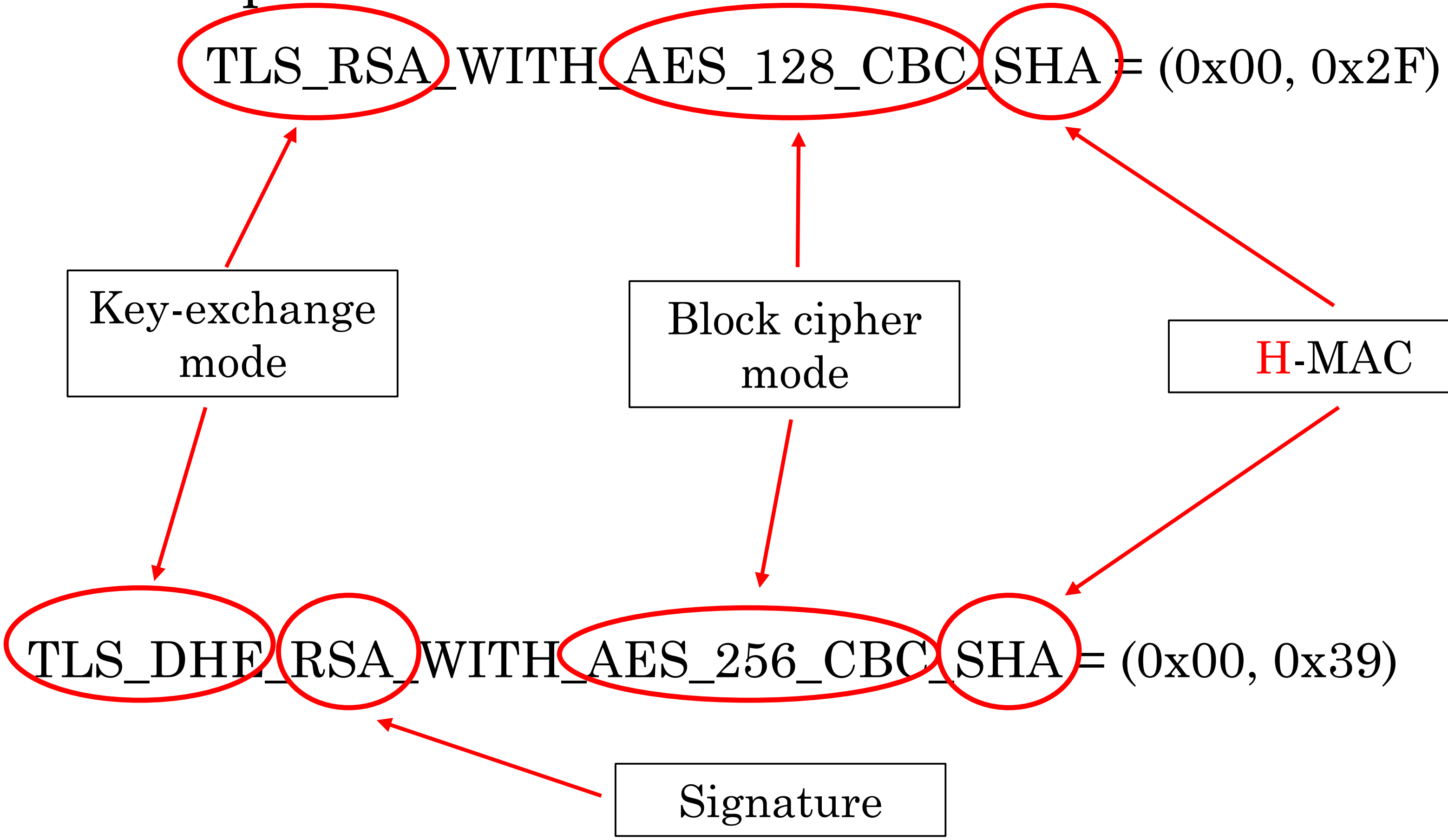
← $N_S, \mathrm{Cert}(KE_S), KE_S$

$$K_C|K_S \leftarrow PRF\ (msk; N_C|N_S)$$

- Compatibility of ciphers and size not verified (enabling the use of export cipher suites)

CIPHER SUITES FOR TLS 1.2

- Example:



TLS_RSA_WITH_AES_128_CBC_SHA = (0x00, 0x2F)

Key-exchange mode

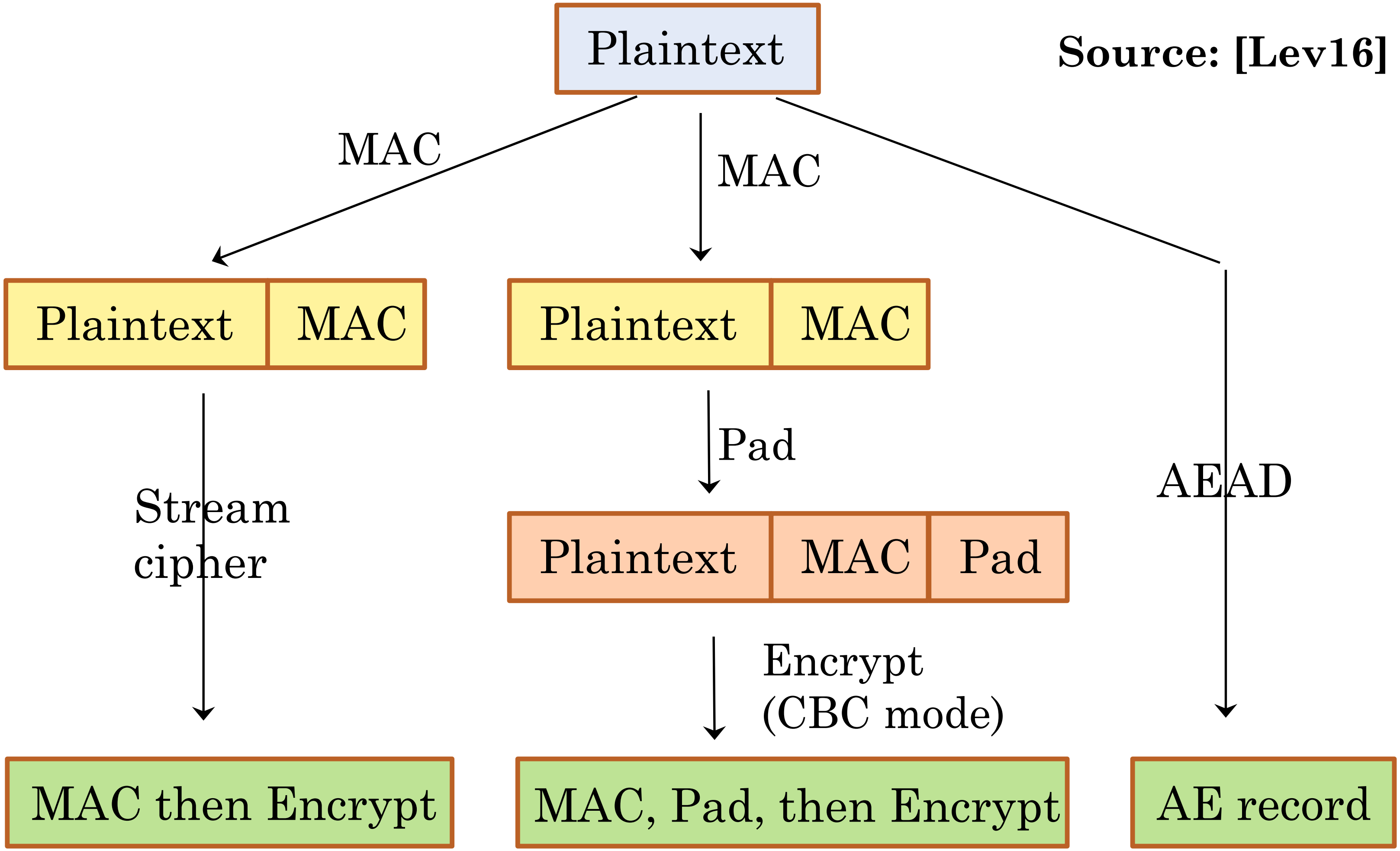
Block cipher mode

H-MAC

TLS_DHE_RSA_WITH_AES_256_CBC_SHA = (0x00, 0x39)

Signature

RECORD LAYER TREATMENT


Source: [Lev16]

Plaintext

MAC

MAC

Plaintext | MAC

Plaintext | MAC

Pad

Plaintext | MAC | Pad

Stream cipher

Encrypt (CBC mode)

AEAD

MAC then Encrypt

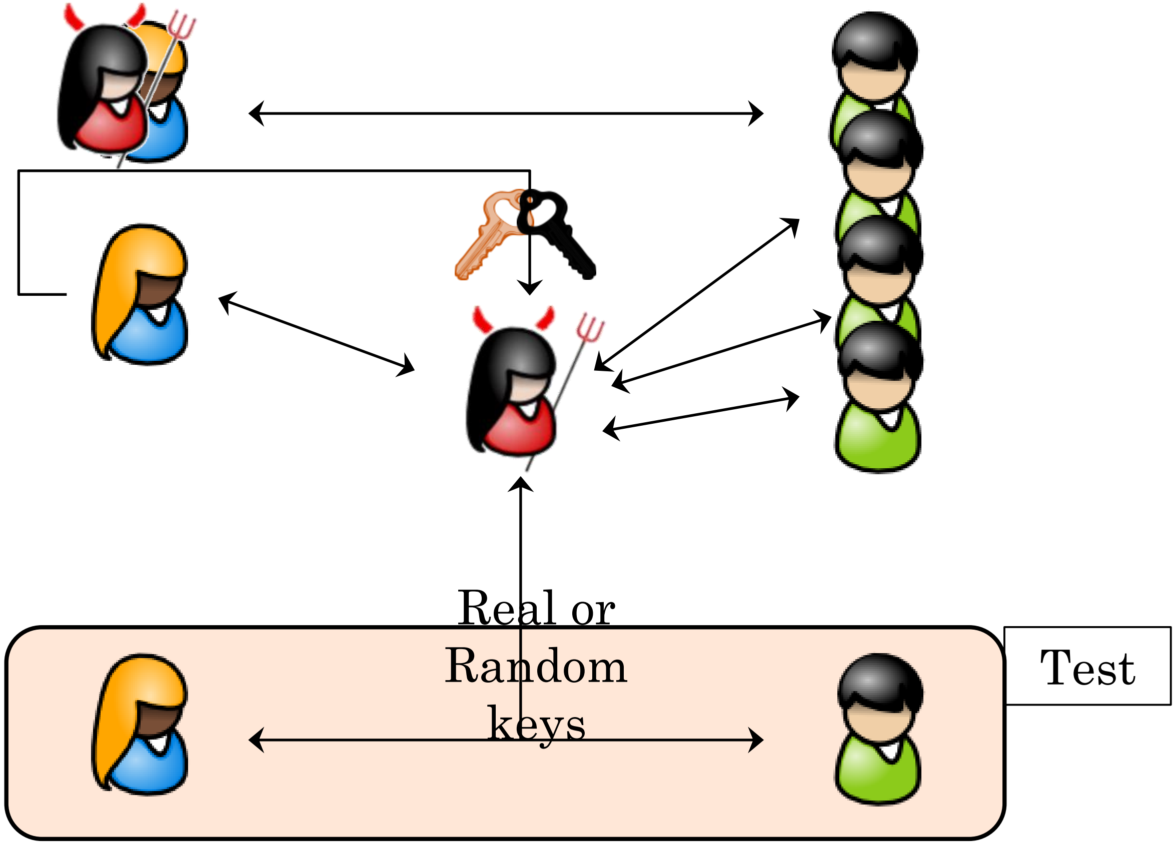
MAC, Pad, then Encrypt

AE record

PART 2
PROVABLE SECURITY AND ATTACKS

WHAT IS A GOOD KEY?

- Bellare-Rogaway security for key exchange [BR93]:



Real or Random keys

Test

BR ATTACKERS

- Active Man-in-the-Middle:
  - Can observe communication
  - Can instantiate communication with any party, in separate session
  - Can reveal session keys
  - Can corrupt parties to learn long-term keys

  - And yet, Adv. cannot distinguish specific session key from random without revealing/ corrupting

- Forward secrecy:
  - Even past keys of corrupted parties look random

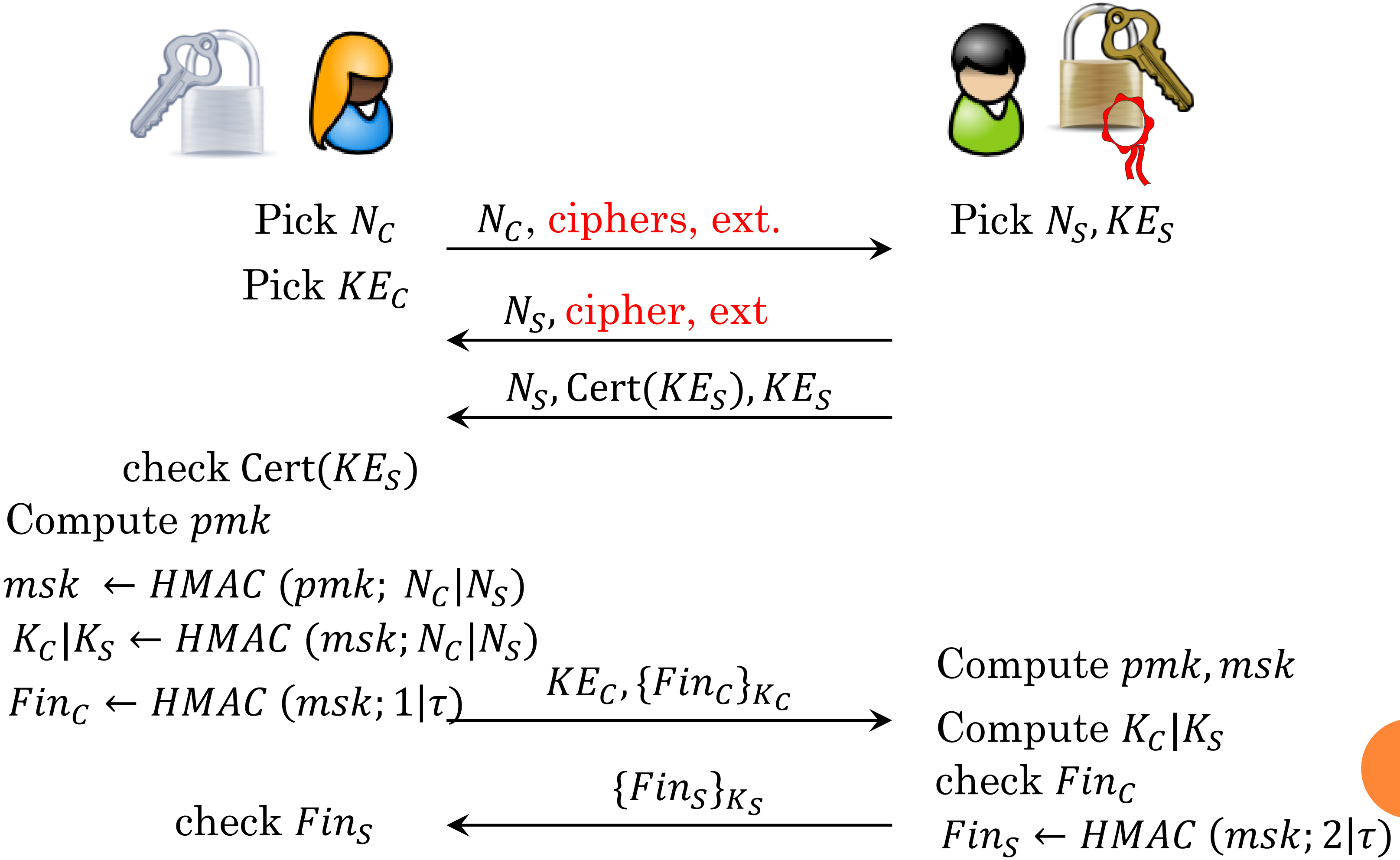
WHY AKE SECURITY

- AKE security:
  - Say session key is indistinguishable from random
  - Whatever you use that key for will be as secure as it is if a random key is used

- Secure symmetric encryption:
  - The key is picked at random from a key space, by the Key Generation algorithm
  - The adversary is never given this key
  - IND-CPA security: the adversary cannot learn even one bit of the encrypted plaintext

- However, the guarantee holds only if key looks random

IS TLS BR-SECURE?



Pick N_C

Pick KE_C

N_C, ciphers, ext. →

Pick N_S, KE_S

← N_S, cipher, ext

← $N_S, \mathrm{Cert}(KE_S), KE_S$

check $\mathrm{Cert}(KE_S)$

Compute pmk

$msk \leftarrow HMAC\ (pmk;\ N_C|N_S)$

$K_C|K_S \leftarrow HMAC\ (msk; N_C|N_S)$

$Fin_C \leftarrow HMAC\ (msk; 1|\tau)$

$KE_C, \{Fin_C\}_{K_C}$ →

Compute pmk, msk

Compute $K_C|K_S$

check Fin_C

$Fin_S \leftarrow HMAC\ (msk; 2|\tau)$

← $\{Fin_S\}_{K_S}$

check Fin_S

TLS AND BR SECURITY

- TLS combines handshake with auth. encryption

- TLS is not secure because of Finished messages
  - Check Real/Random by simulating Finished messages
  - If the key is confirmed, it's real; else, it's random

- ACCE security:
  - Introduced by Jager et al. [JKSS, Crypto 2012]
  - 2 guarantees:
    - unilateral or mutual authentication
    - Channel security (the computed key is safe to use with AE)
    - No guarantees for other uses (e.g. for authentication)

(S)ACCE SECURITY OF TLS

- Breakthrough in TLS Security
  - Krawczyk, Paterson, Wee (2013): TLS 1.2 is secure
  - Bhargavan et al. (2014): TLS 1.2 is secure even with session resumption and changing ciphersuites
  - Kohlweiss et al. (2014): TLS 1.2 is secure even in composition with other protocols

- Guarantee requires:
  - MSK expansion from $\mathrm{KE}_C, \mathrm{KE}_S$ is truly random
  - Key expansion function is PRF
  - Gap Diffie-Hellman problem is hard
  - Record-layer primitives are secure

TLS & FORWARD SECRECY

- Forward secrecy:
  - Adversary watches some sessions, records transcripts
  - Adversary corrupts server to get key
- TLS-RSA mode:
  - Corruption yields long-term RSA secret-key
  - Adversary can decrypt all past *pmk* encryptions
- TLS-DH mode:
  - Corruption yields discrete log of static DH share
  - Adversary can calculate past *pmk* values
- TLS-DHE mode:
  - Corruption yields long-term signature secret key
  - Adversary can sign, but cannot retrieve past DLogs

RECORD-LAYER SECURITY

- Cipher Suites:
  - Chosen by client when sending nonce
  - Define: key-exchange, sym. encryption, MAC, PRF
  - Choice of block or stream ciphers, hash functions, etc.

- Provable security:
  - If you have good keys, IND-CPA-secure authenticated encryption, then this creates a secure channel
  - Problem 1: we don't really know which cipher suites are IND-CPA secure
  - Problem 2: security models feature single-block msgs; real world msgs are multi-block and padded

PROBLEMS WITH CBC-MODE

- Why we like CBC mode:
  - Efficient in practice: can decrypt a lot in constant memory and linear time
  - Just as good as ECB for efficiency, better security

- Some limits:
  - Problems with choice of IV
  - CBC-MAC has problems with unforgeability

- More serious: attack by Vaudenay

VAUDENAY'S ATTACK

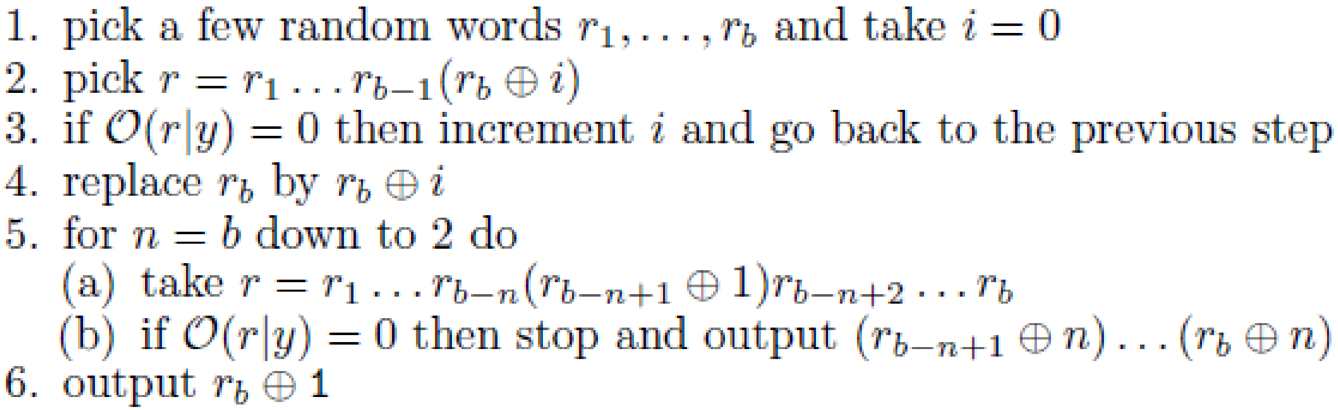
- Works for specific kind of padding:
  - Consider block length b in bytes
  - Message m that has length (in bytes) not a multiple of b
  - Pad with n bytes, each equal to n : 1, 22, 333, etc.
  - Padded message: $[x_1, \ldots, x_N]$, each x_i a full block
  - Encrypt:

$$y_1 = C(IV\ \ XOR\ \ x_i);\ \text{and } y_i = C(y_{i-1}\ XOR\ x_i)$$

- Uses error messages as oracles:
  - If padding is incorrect, receiving party usually complains
  - Change ciphertext y and watch if padding still ok

BASIC ATTACK

- First step: find last word of y

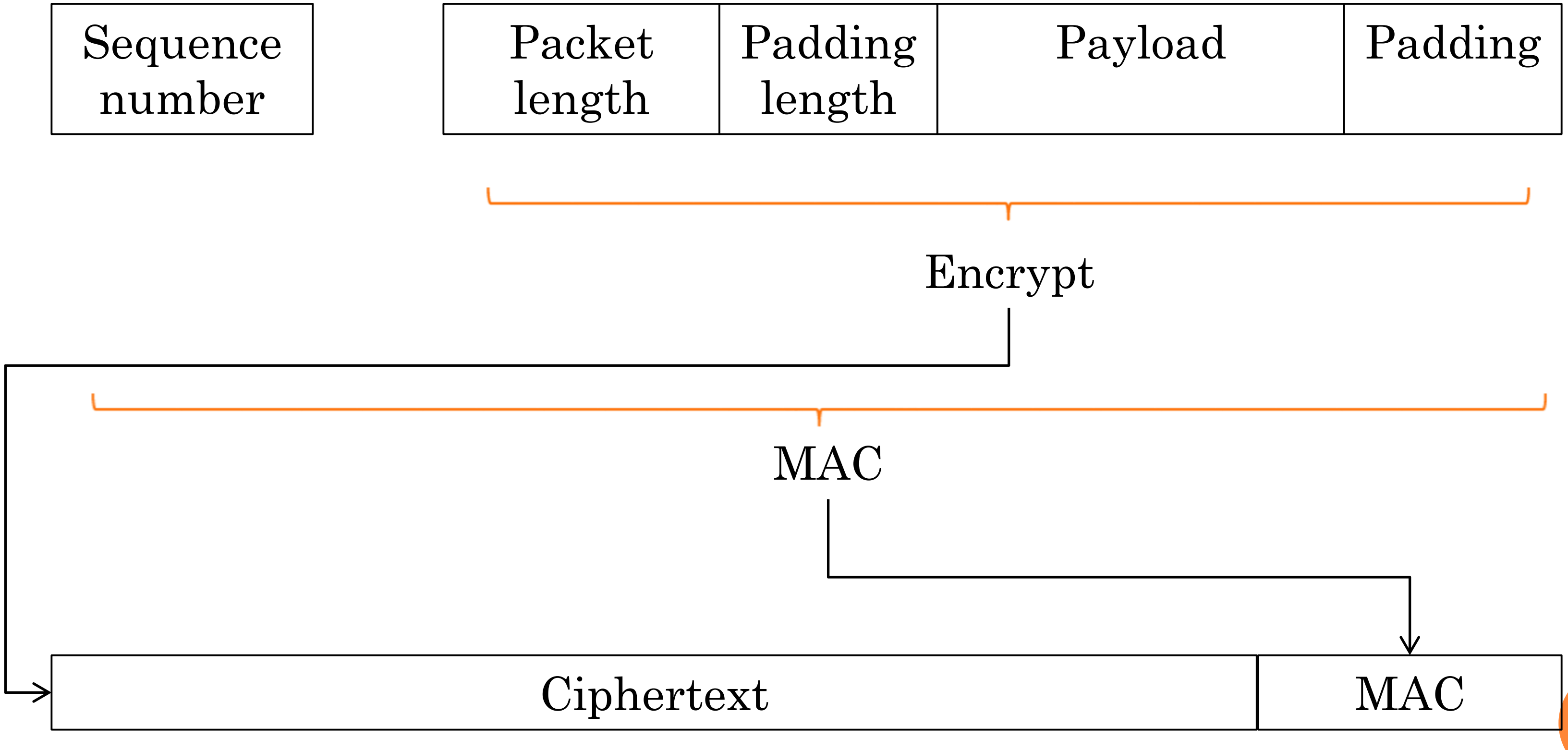


```
1. pick a few random words $r_1, \ldots, r_b$ and take $i = 0$
2. pick $r = r_1 \ldots r_{b-1}(r_b \oplus i)$
3. if $\mathcal{O}(r|y) = 0$ then increment $i$ and go back to the previous step
4. replace $r_b$ by $r_b \oplus i$
5. for $n = b$ down to 2 do
   (a) take $r = r_1 \ldots r_{b-n}(r_{b-n+1} \oplus 1)r_{b-n+2} \ldots r_b$
   (b) if $\mathcal{O}(r|y) = 0$ then stop and output $(r_{b-n+1} \oplus n) \ldots (r_b \oplus n)$
6. output $r_b \oplus 1$
```

- Why this works:
  - If $O(r|y) = 1$ then padding checks for decrypted ciphertext
  - Which means, padding is correct for $C^{-1}(y)\ XOR\ r$
- Repeat to get last block of y , then to get y

ERRORS THAT KILL (OPENSSH)

- Encrypt-then-MAC is bad: Albrecht et al.



PLAINTEXT RECOVERY

- Idea:
  - Forget about the length being a length field
  - Imagine you wanted to decrypt a ciphertext
  - Start with one block of this ciphertext (which you want to decrypt), and send it as the first part of a new ciphertext
  - Wait and see
  - If no termination, then the packet passed the length check
  - We learn 14 bits of plaintext
  - Repeat this to get 32 bits, then more

HISTORY OF TLS ATTACKS

- Renegotiation attack vs >SSL 3.0: plaintext injection

  Ideal Patch: kill renegotiation/generate more entropy

  Real Patch: include previous session history

- Version rollback attacks: use older, weaker version/cipher

  Ideal Patch: kill backward compatibility/weak ciphers

  Real Patch: ??? (not an important/realistic attack)

- BEAST: browser exploits of CBC vulnerabilities

  Ideal Patch: kill CBC mode/ kill < TLS 1.2

  Real Patch: fixed in TLS 1.1, but even if client has TLS >1.1, weak servers can force it to TLS 1.0.

  Extra Patch: discouraged CBC mode

  encouraged RC4…

MORE ATTACKS ON TLS

- CRIME/BREACH: exploit compression characteristics
  - **Ideal** Patch: kill data compression
  - **Real** Patch: can kill some compression in TLS/SPDY headers; cannot kill HTTP compression (against BREACH)
- Timing attacks/Lucky 13: exploit padding problems
  - **Ideal** Patch: kill CBC mode
  - **Real** Patch: encourage RC4 instead of CBC mode
    TLS 1.2 does offer one good ciphersuite: AES-GCM
- POODLE: downgrade to SSL 3.0 and exploit away
  - **Ideal** Patch: kill backward compatibility
  - **Real** Patch: close our eyes and hope it goes away?

AND EVEN MORE ATTACKS

- RC4 attacks: RC4 output biased – NOT pseudorandom

Attack specifics: 2014 – use many encryptions (2^{34}) and lots of generated traffic to do something à la BREACH/CRIME (on cookies)

2015 – use less encryptions (2^{26}) on passwords with100 tries before lockout. Password recovery rate: 50% for pw-length 6 for BasicAuth (Chrome)

Ideal Patch: kill RC4

Real Patch: RFC 7465 prohibits RC4 cipher suites.

Real Deployment: 30% of SSL/TLS traffic still uses RC4[1]
74.5% of sites allow RC4 negotiation[2]
few sites deploy TLS 1.2, which means alternatives are just as bad…

---

[1] ICSI Certificate Notary project; [2] SSL Pulse

DOES IT EVER STOP?

- Heartbleed: does not affect SSL/TLS, rather OpenSSL

  Attack strategy: read memory of users with problematic versions of OpenSSL, essentially learning their long-term data

  Patch: do not use OpenSSL 1.0.1. to 1.0.1f.

- 3Shake: S_1^* forces same MSK in S_1^* - A and S_1^* - S_2

  Attack strategy: use same PMK material in two sessions, then use session resumption (no certificates!)

  Ideal Patch: kill renegotiation and finite fields; use ECDHE

  Real Patch: not really all that much…

- FREAK: force connection on weak parameters

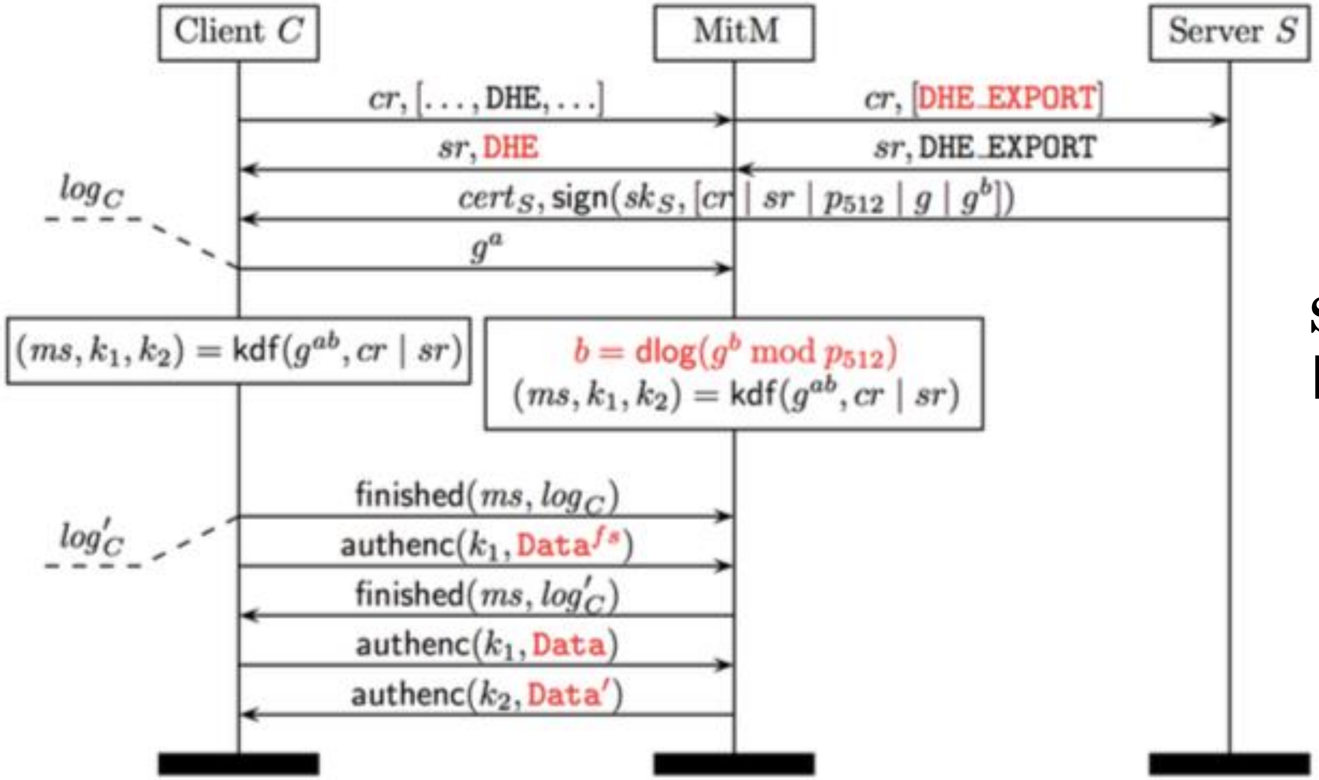
  Ideal Patch: kill backward compatibility

  Real Patch: fix OpenSSL, preserve backward compatibility

A RECENT BUG: LOGJAM

- Export cipher suites: date back to 90s, have small primes

  Can break DLog on those groups easily, thus forge connection



Source: [ABD+15]

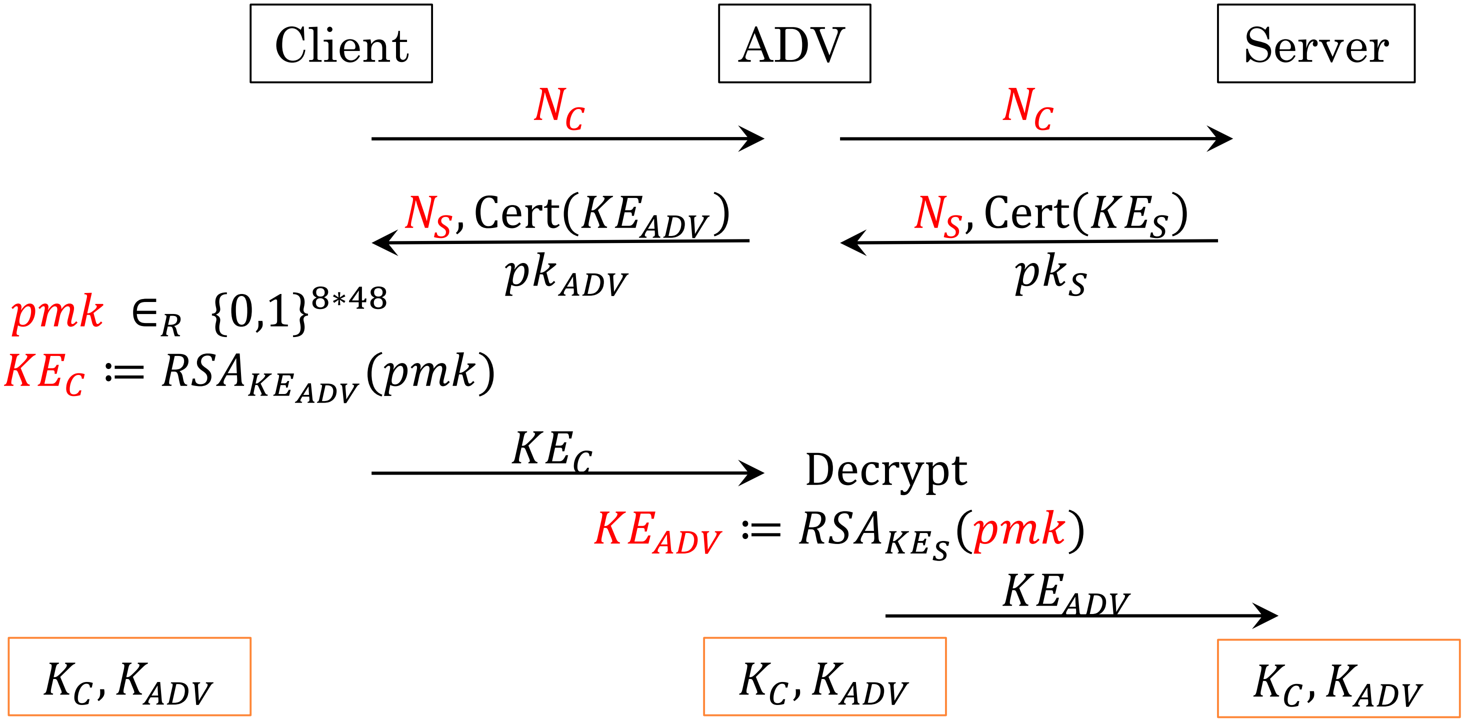
WHY LOGJAM WORKS

- Export ciphers still exist
  - Originally for exporting cipher suites outside the US
  - No longer really needed, but dormant in implementation
  - They look innocuous, like regular DH parameters

- Solving DLog on 512-bit fields
  - Usually servers use the same primes over and over again: break it once, you will know it next time
  - Generally takes longer than usual timeout of sessions…
  - … but we can feed the server nonsense messages to make it wait longer
  - Bhargavan et al.: 70 seconds to break DLog

Another bug: 3Shake [BDF+14]

- What if the attacker is a legitimate server?
  - This server has a legitimate certificate
  - Its goal is to see information meant for other servers
- Strategy: first synch. keys, then relay



Client ADV Server

$\xrightarrow{N_C}$ $\xrightarrow{N_C}$

$\xleftarrow[pk_{ADV}]{N_S, \text{Cert}(KE_{ADV})}$ $\xleftarrow[pk_S]{N_S, \text{Cert}(KE_S)}$

$pmk \in_R \{0,1\}^{8*48}$

$KE_C := RSA_{KE_{ADV}}(pmk)$

$\xrightarrow{KE_C}$ Decrypt

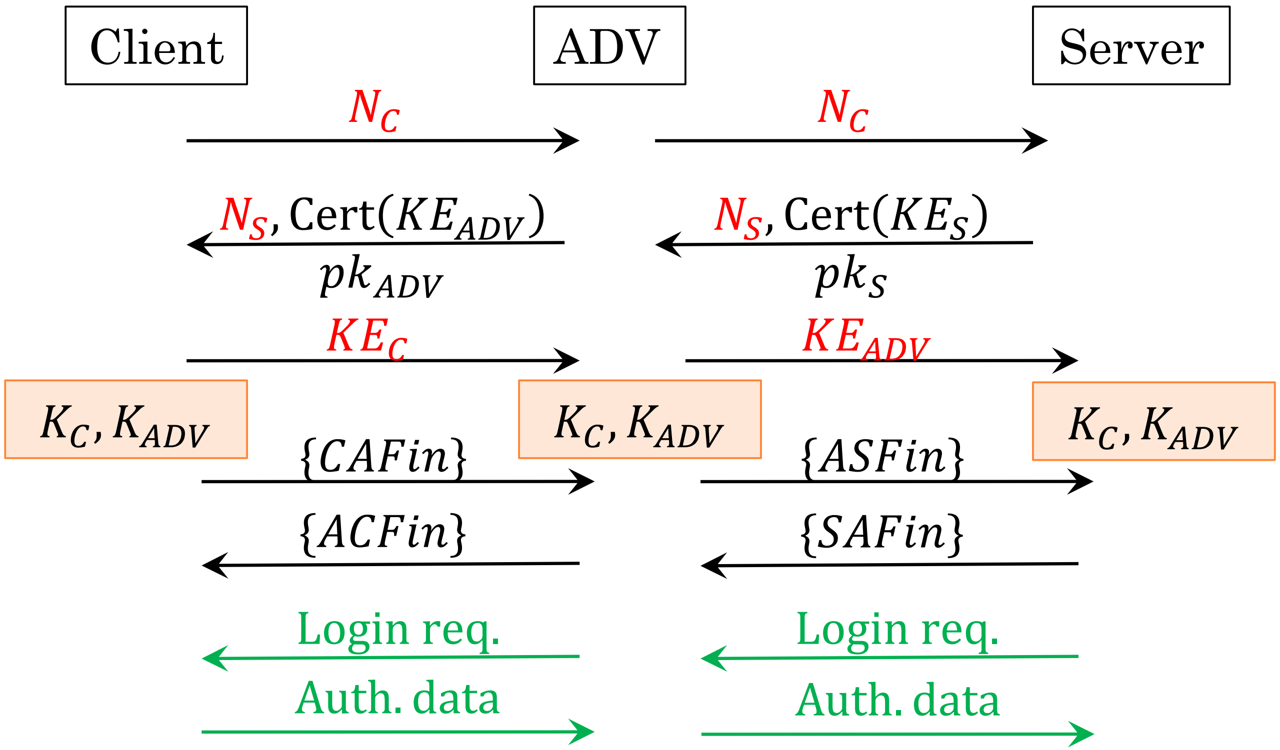
$KE_{ADV} := RSA_{KE_S}(pmk)$

$\xrightarrow{KE_{ADV}}$

$\boxed{K_C, K_{ADV}}$ $\boxed{K_C, K_{ADV}}$ $\boxed{K_C, K_{ADV}}$

ANOTHER BUG: 3SHAKE [BDF+14]

- Now suppose the three parties share keys
  - Adv now wants to access C’s Amazon’s account
  - Amazon requires user-name + password



Client

ADV

Server

N_C

N_C

$N_S, \text{Cert}(KE_{ADV})$

pk_{ADV}

$N_S, \text{Cert}(KE_S)$

pk_S

KE_C

KE_{ADV}

K_C, K_{ADV}

K_C, K_{ADV}

K_C, K_{ADV}

$\{CAFin\}$

$\{ASFin\}$

$\{ACFin\}$

$\{SAFin\}$

Login req.

Login req.

Auth. data

Auth. data

BUT… WASN'T TLS PROVABLY SECURE?

- ➢ Security statement equivalent to:
  - ➢ In the ROM (or with weird assumptions), given:
    - A secure certification scheme (PKI)
    - A collision-resistant hash function
    - A PRF that is indistinguishable from random
    - A Strongly-unforgeable HMAC
    - Either CBC-mode block cipher that is a super PRP; or a stream cipher with PR output

---

- ➢ Then: TLS-RSA, TLS-DH, TLS-DHE secure

How does that fit in with attacks?

GAP MODEL/REALITY

- De-facto security model:
  - 1 server, perfect protocol implementation:

    Rules out 3Shake, Heartbleed, Padding attacks

    Rules out cookie problems: BREACH/CRIME…
  - Does not capture changing ciphersuites/renegotiation

    Rules out FREAK, renegotiation, version rollback…
- Reductions
  - Assuming CBC-mode block cipher that is a super PRP…

    … which is not true for TLS…
  - Assuming stream cipher with PR output…

    … DEFINITELY not true for RC 4…

Close the gap or change the protocol

PART 3
TLS 1.3

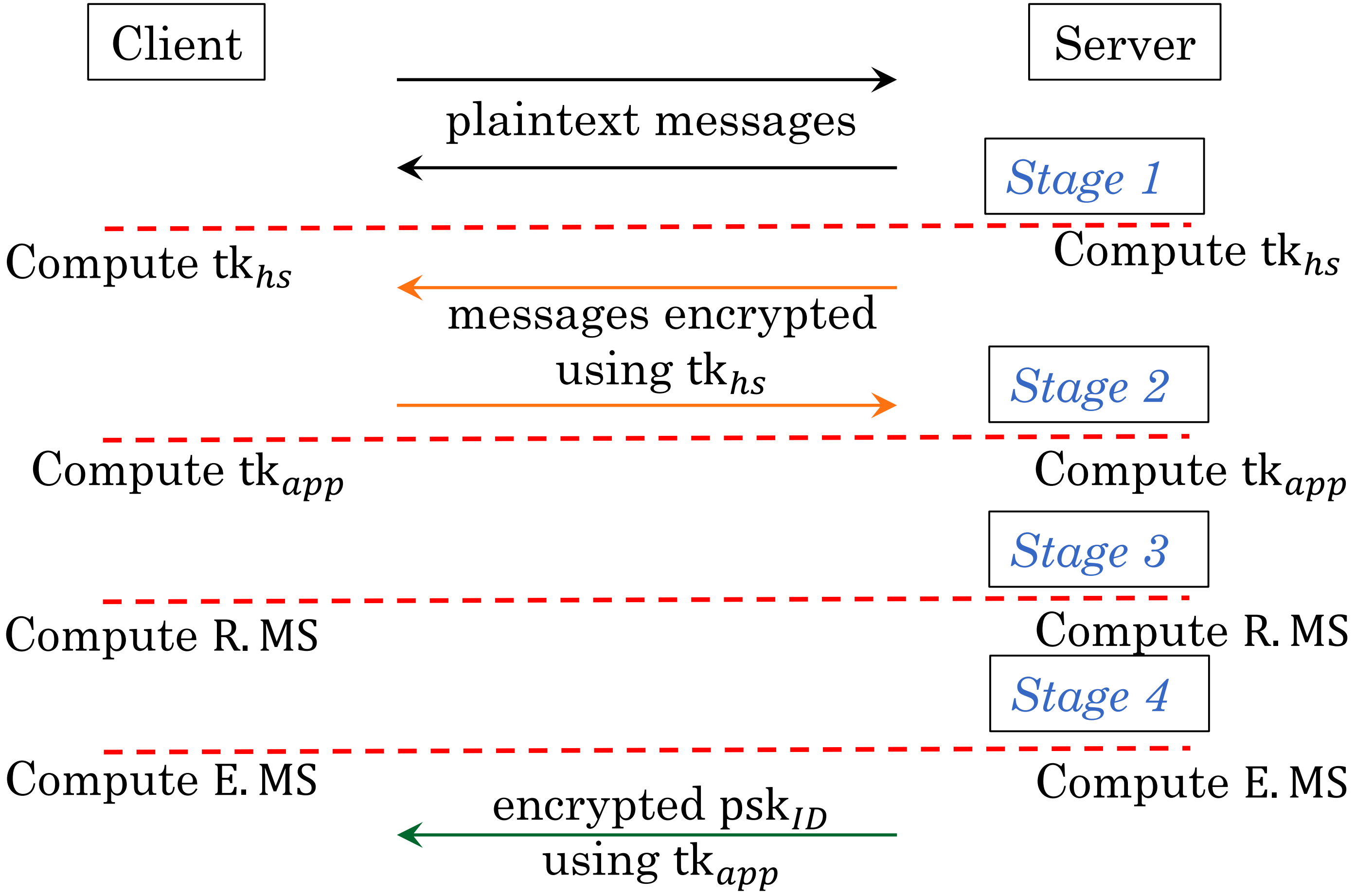
BASICS OF TLS 1.3

- TLS 1.3 philosophy:
  - Modular protocol design
  - Preserves features such as key-confirmation
  - … but guarantees AKE security (is composable)
  - Few, good ciphersuites
  - As much privacy as Tor (privacy vs. passive attacks)

- Several modes of operation:
  - Full handshake in DHE mode
  - Pre-Shared Key
  - PSK + DHE
  - 0-RTT

FULL HANDSHAKE STRUCTURE [V13]

- Several *stages*, one stage per key:



Client

Server

plaintext messages

Stage 1

Compute tk_{hs}

Compute tk_{hs}

messages encrypted using tk_{hs}

Stage 2

Compute tk_{app}

Compute tk_{app}

Stage 3

Compute R. MS

Compute R. MS

Stage 4

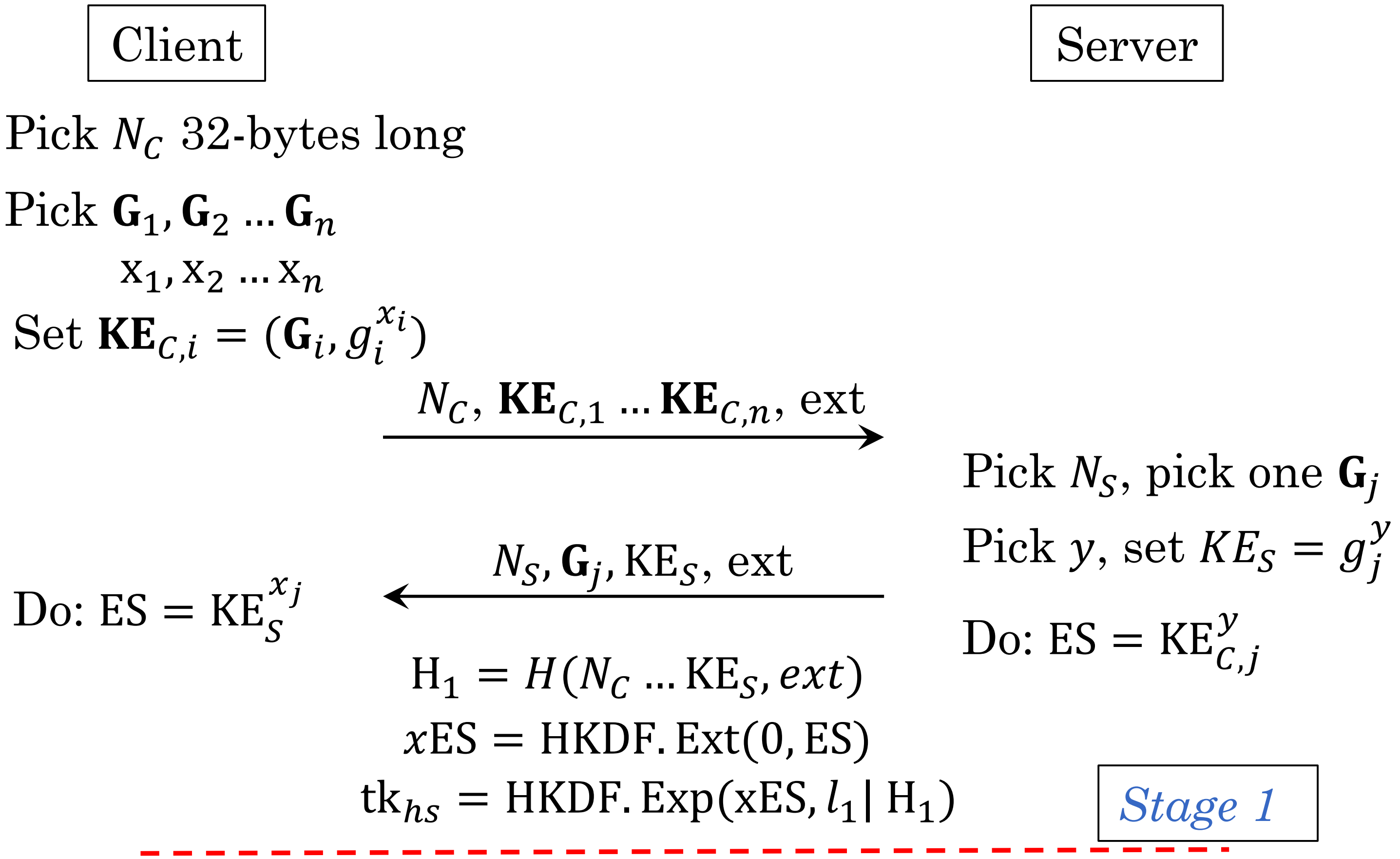
Compute E. MS

Compute E. MS

encrypted psk_{ID} using tk_{app}

STAGE 1 OF FULL HANDSHAKE

- Stage 1: handshake keys



Client

Server

Pick N_C 32-bytes long

Pick $\mathbf{G}_1, \mathbf{G}_2 \dots \mathbf{G}_n$

$\mathrm{x}_1, \mathrm{x}_2 \dots \mathrm{x}_n$

Set $\mathbf{KE}_{C,i} = (\mathbf{G}_i, g_i^{x_i})$

$\xrightarrow{N_C,\ \mathbf{KE}_{C,1} \dots \mathbf{KE}_{C,n},\ \text{ext}}$

Pick N_S, pick one $\mathbf{G}_j$

Pick y, set $KE_S = g_j^y$

$\xleftarrow{N_S, \mathbf{G}_j, \mathrm{KE}_S,\ \text{ext}}$

Do: $\mathrm{ES} = \mathrm{KE}_S^{x_j}$

Do: $\mathrm{ES} = \mathrm{KE}_{C,j}^{y}$

$$\mathrm{H}_1 = H(N_C \dots \mathrm{KE}_S, ext)$$

$$x\mathrm{ES} = \mathrm{HKDF.Ext}(0, \mathrm{ES})$$

$$\mathrm{tk}_{hs} = \mathrm{HKDF.Exp}(\mathrm{xES}, l_1 | \mathrm{H}_1)$$

Stage 1

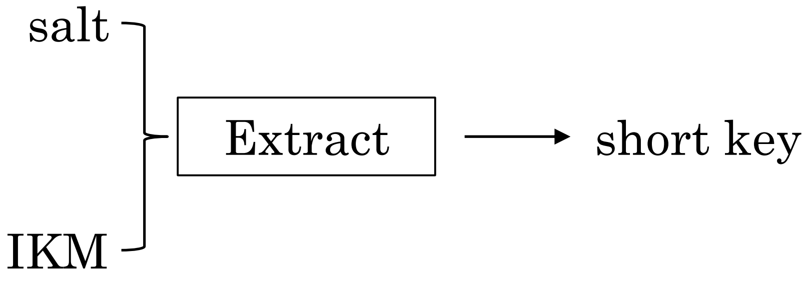
CLIENT & SERVER HELLO

- TLS 1.2
  - Client Hello message:
    - Version, random, sID, ciphersuites, compression, extensions
  - Server Hello message:
    - Version, random, sID, ciphersuite, compression, extensions
  - In TLS-DHE, server chooses (EC)DHE group

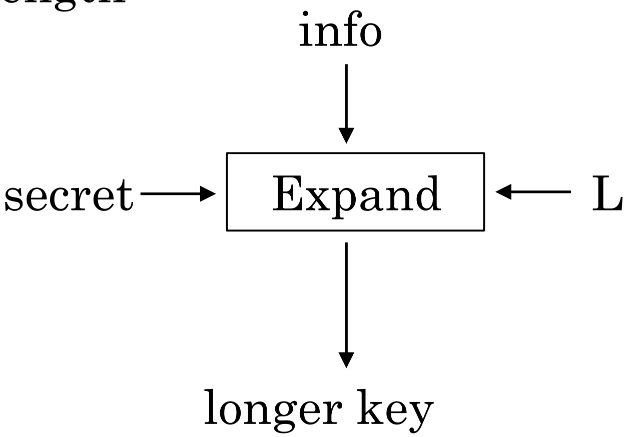
- TLS 1.3
  - Client Hello:
    - Includes list of groups and key-shares for all those groups
  - Server Hello:
    - Chooses one group, generates key share

THE HKDF FUNCTIONS [RFC 5869]

- 2 functions:
  - Extract takes a “salt” and an “input key material”
    - Its goal is to extract entropy
  - Expand takes a “secret”, a context, and a length
    - Its goal is to return PR keys of that length

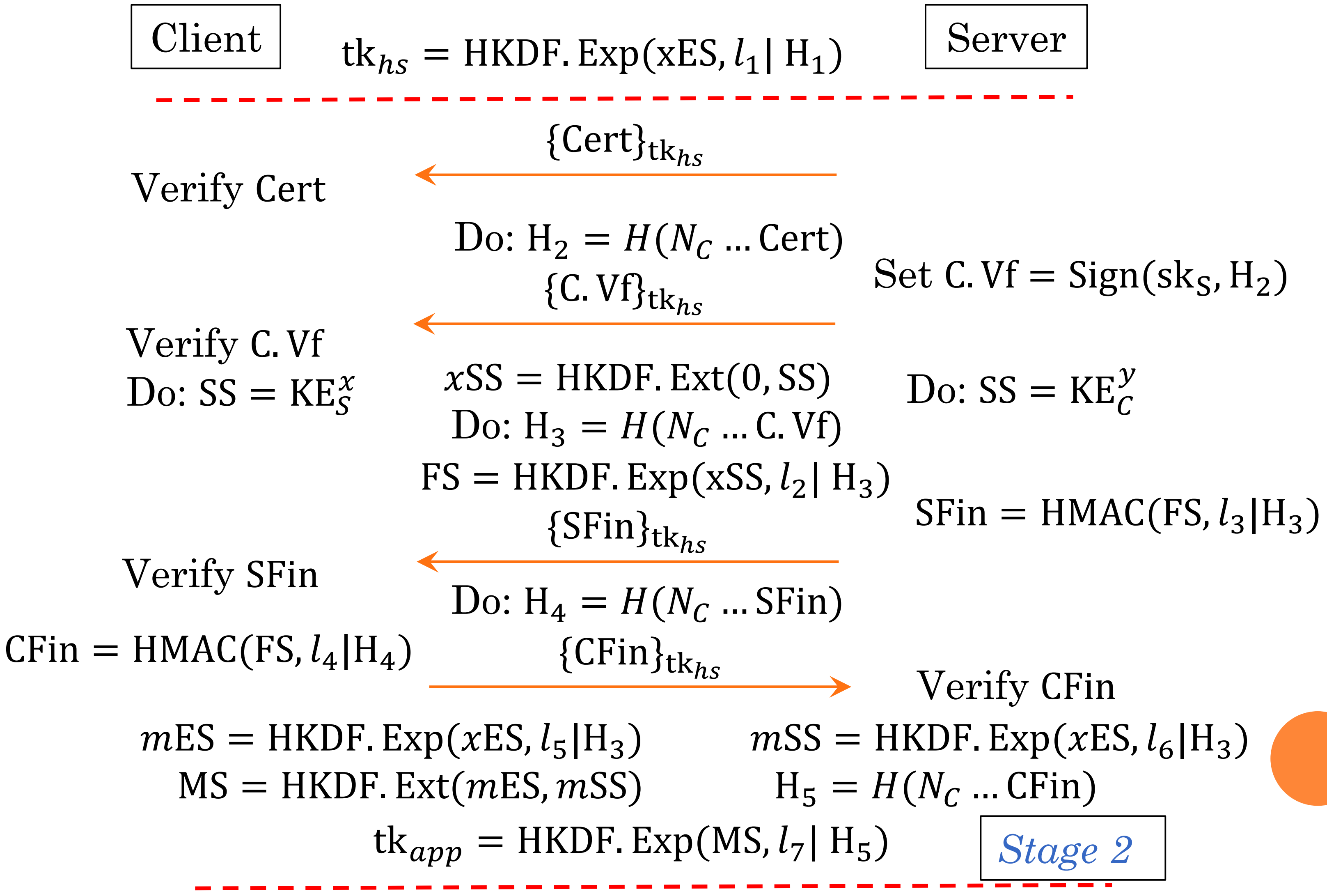


$$xES = \text{HKDF.Ext}(\underset{\text{salt}}{0}, \overset{\text{IKM}}{\text{ES}})$$



$$\text{tk}_{hs} = \text{HKDF.Exp}(\underset{\text{secret}}{\text{xES}}, \underbrace{l_1 | \text{H}_1}_{\text{info}})$$

STAGE 2 OF FULL HANDSHAKE



Client

$\mathrm{tk}_{hs} = \mathrm{HKDF.Exp}(\mathrm{xES}, l_1 |\, \mathrm{H}_1)$

Server

$\xleftarrow{\{\mathrm{Cert}\}_{\mathrm{tk}_{hs}}}$

Verify Cert

Do: $\mathrm{H}_2 = H(N_C \ldots \mathrm{Cert})$

Set $\mathrm{C.Vf} = \mathrm{Sign}(\mathrm{sk}_S, \mathrm{H}_2)$

$\xleftarrow{\{\mathrm{C.Vf}\}_{\mathrm{tk}_{hs}}}$

Verify C.Vf

Do: $\mathrm{SS} = \mathrm{KE}_S^x$

$x\mathrm{SS} = \mathrm{HKDF.Ext}(0, \mathrm{SS})$

Do: $\mathrm{SS} = \mathrm{KE}_C^y$

Do: $\mathrm{H}_3 = H(N_C \ldots \mathrm{C.Vf})$

$\mathrm{FS} = \mathrm{HKDF.Exp}(\mathrm{xSS}, l_2 |\, \mathrm{H}_3)$

$\mathrm{SFin} = \mathrm{HMAC}(\mathrm{FS}, l_3 | \mathrm{H}_3)$

$\xleftarrow{\{\mathrm{SFin}\}_{\mathrm{tk}_{hs}}}$

Verify SFin

Do: $\mathrm{H}_4 = H(N_C \ldots \mathrm{SFin})$

$\mathrm{CFin} = \mathrm{HMAC}(\mathrm{FS}, l_4 | \mathrm{H}_4)$

$\xrightarrow{\{\mathrm{CFin}\}_{\mathrm{tk}_{hs}}}$

Verify CFin

$m\mathrm{ES} = \mathrm{HKDF.Exp}(x\mathrm{ES}, l_5 | \mathrm{H}_3)$

$m\mathrm{SS} = \mathrm{HKDF.Exp}(x\mathrm{ES}, l_6 | \mathrm{H}_3)$

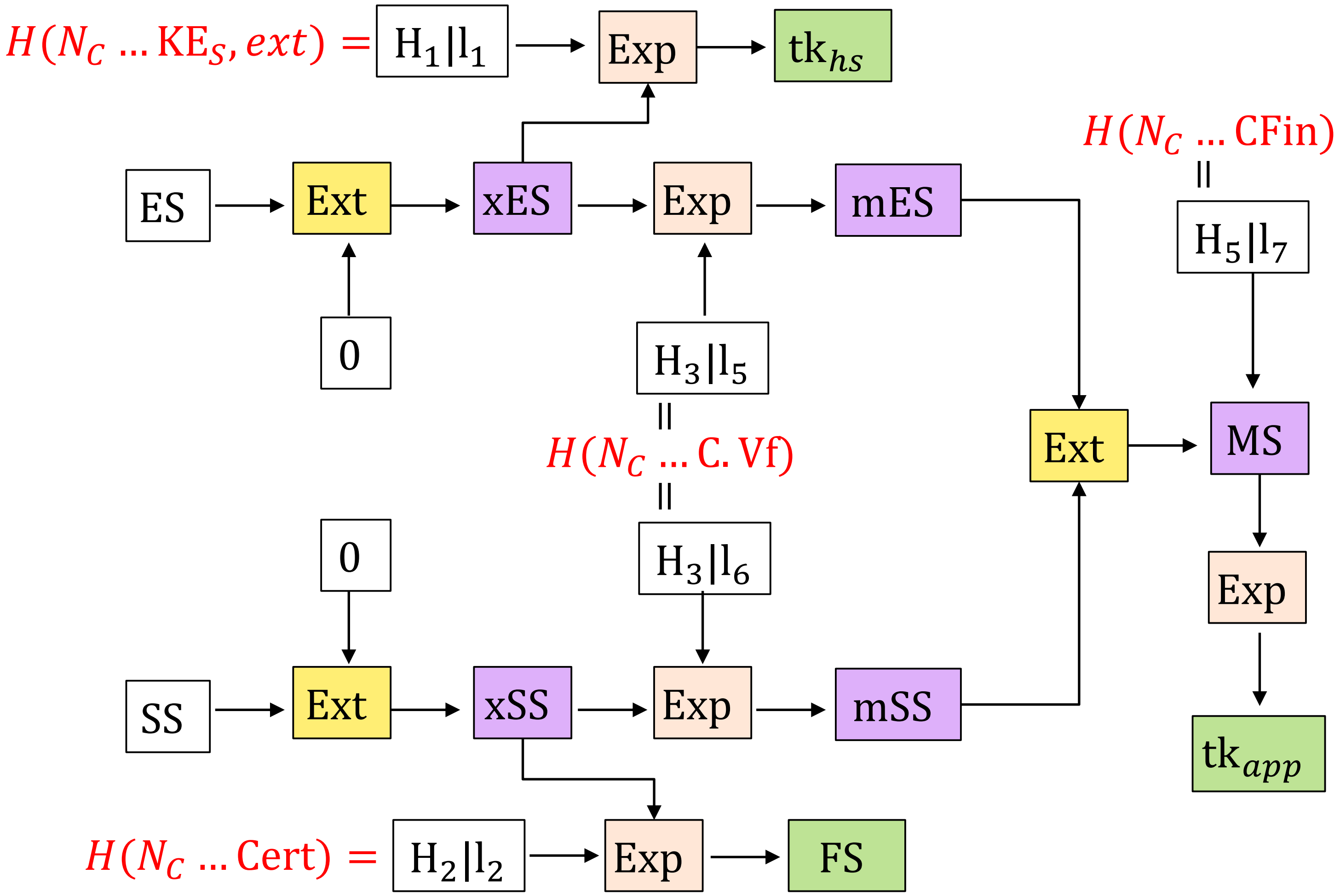
$\mathrm{MS} = \mathrm{HKDF.Ext}(m\mathrm{ES}, m\mathrm{SS})$

$\mathrm{H}_5 = H(N_C \ldots \mathrm{CFin})$

$\mathrm{tk}_{app} = \mathrm{HKDF.Exp}(\mathrm{MS}, l_7 |\, \mathrm{H}_5)$

Stage 2

KEY SCHEDULE UP TO STAGE 2



$H(N_C \ldots \mathrm{KE}_S, ext) =$ $\mathrm{H}_1 || \mathrm{l}_1$ → Exp → tk_{hs}

ES → Ext → xES → Exp → mES

0

$H(N_C \ldots \mathrm{CFin})$

||

$\mathrm{H}_5 || \mathrm{l}_7$

$\mathrm{H}_3 || \mathrm{l}_5$

||

$H(N_C \ldots \mathrm{C.Vf})$

||

$\mathrm{H}_3 || \mathrm{l}_6$

Ext → MS → Exp → tk_{app}

0

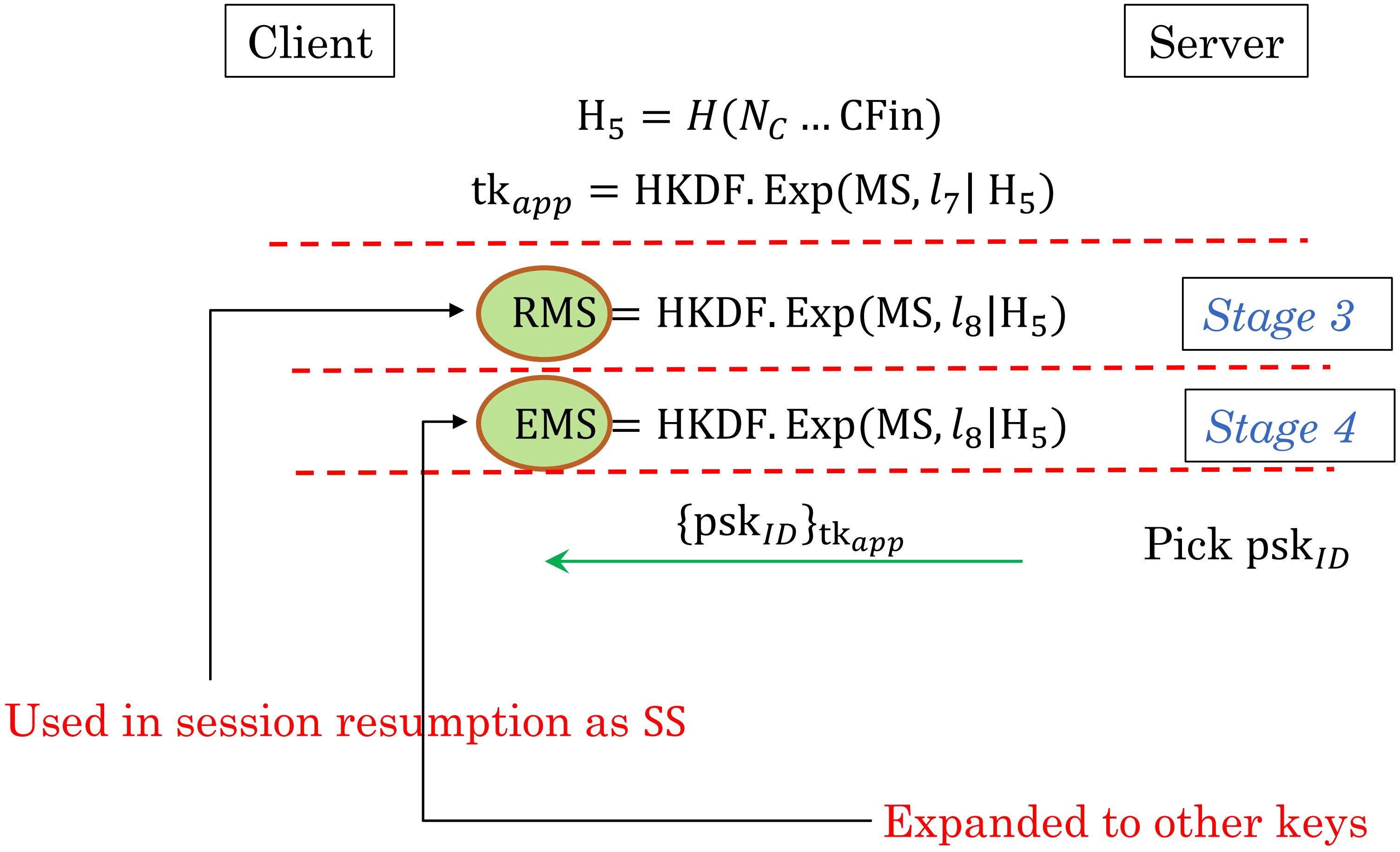
SS → Ext → xSS → Exp → mSS

$H(N_C \ldots \mathrm{Cert}) =$ $\mathrm{H}_2 || \mathrm{l}_2$ → Exp → FS

ABOUT OUR KEYS

- Stage 1: tk_{hs} computed from ES $= g^{x \cdot y}$ and hello hash
  - but not authenticated by end of Stage 1
- Stage 2: tk_{app} computed from ES $= g^{x \cdot y}$ and SS $=$ ES
  - Step I: authenticate tk_{hs} -- indirect authentication of ES
  - Step II: obtain FS from SS via *Extract* + *Expand*
    - With a different Hash + label than tk_{hs} from ES
    - FS, tk_{hs} independent, but confirming same secret
  - Step III: obtain $m\text{ES}, m\text{SS}$ from $x\text{ES} = x\text{SS}$
    - Hash used in both cases is identical
    - But label is different, making $m\text{ES}, m\text{SS}$ independent
  - Step IV: get master secret MS from $m\text{ES}$, $m\text{SS}$
  - Step V: get tk_{app} from MS with yet another hash & label

STAGES 3 AND 4



Client

Server

$H_5 = H(N_C \ldots \mathrm{CFin})$

$\mathrm{tk}_{app} = \mathrm{HKDF.Exp}(\mathrm{MS}, l_7 | \mathrm{H}_5)$

RMS $= \mathrm{HKDF.Exp}(\mathrm{MS}, l_8 | \mathrm{H}_5)$

Stage 3

EMS $= \mathrm{HKDF.Exp}(\mathrm{MS}, l_8 | \mathrm{H}_5)$

Stage 4

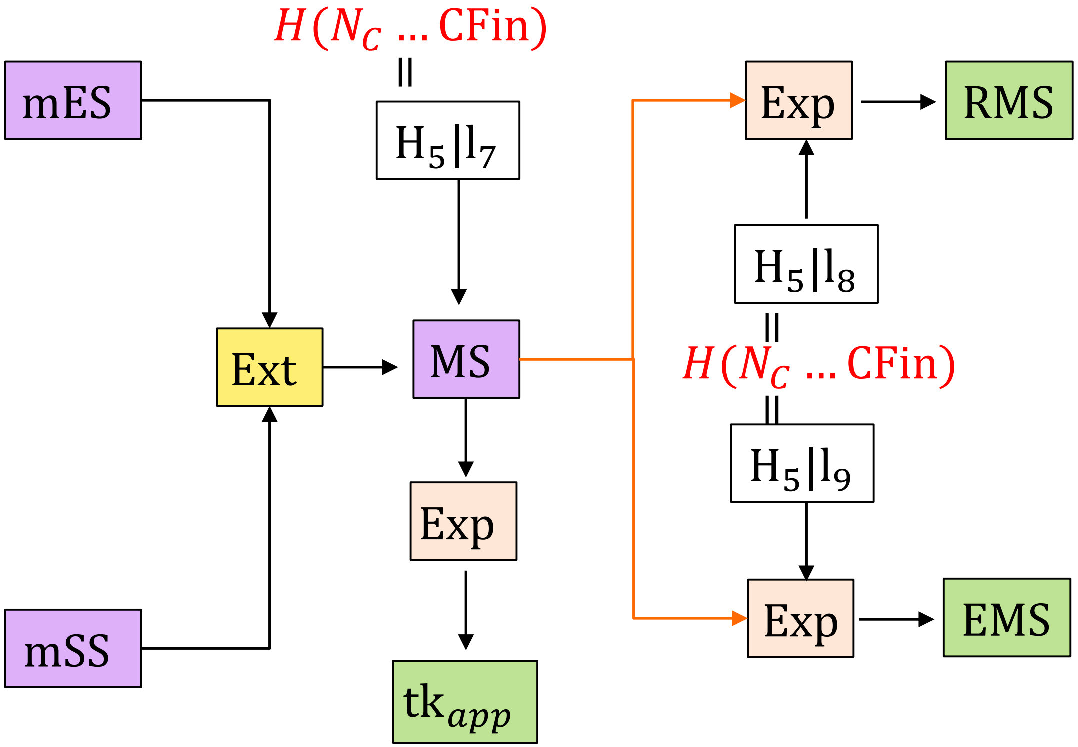
$\{\mathrm{psk}_{ID}\}\mathrm{tk}_{app}$

Pick psk_{ID}

Used in session resumption as SS

Expanded to other keys

MORE KEY SCHEDULING



RESUMPTION AND EXPORT SECRETS

- The resumption secret RMS
  - Result of expanding MS with new label, session hash
  - The RMS is maintained, associated with psk$_{id}$
  - If prompted with psk$_{id}$, parties will use RMS as SS
  - We will see resumption later

- The export secret EMS
  - Will be used to yield further (independent) keys
  - Export keys: used for other applications, like:
    - Personal authentication
    - Encryption in different applications

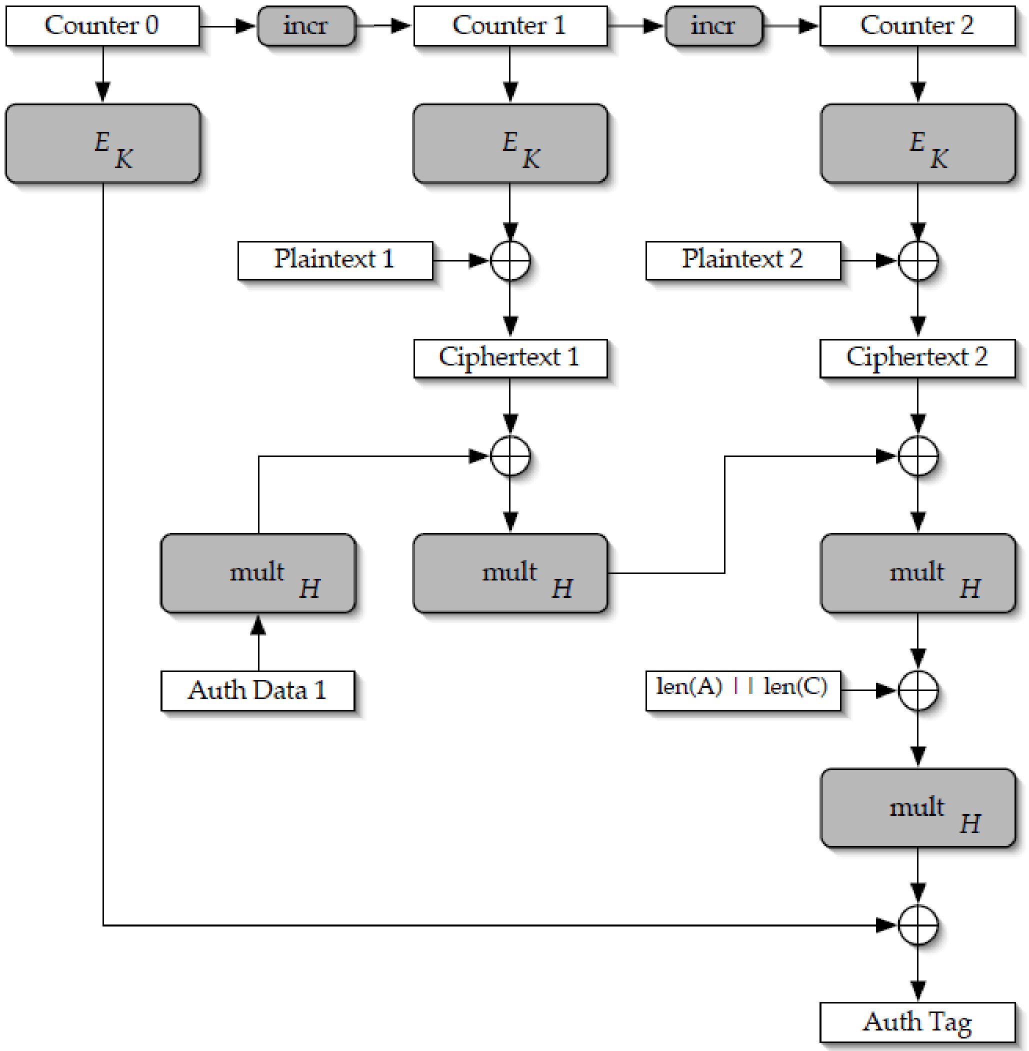
RECORD LAYER PRIMITIVES

- One block cipher, one stream cipher only

- AES – GCM (McGrew, Viega)
  - Allows not only encrypt + MAC, but also includes EA
  - Couter-mode encryption

- ChaCha20-Poly1305
  - ChaCha20: stream cipher based on Salsa20 [Bernstein]
  - Poly1305: AES-based MAC (Nir, Langley, RFC 7539)

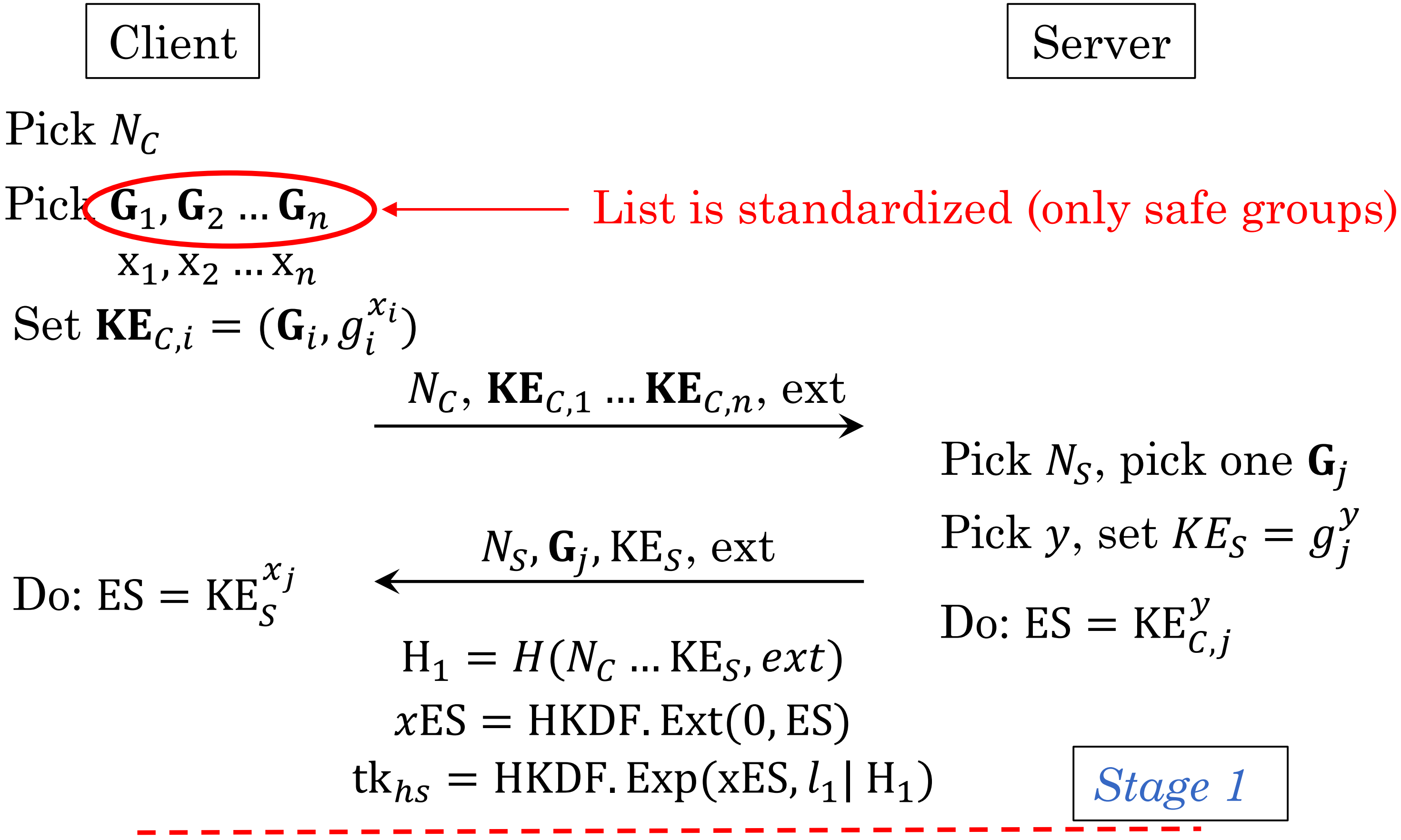
AES-GCM



Source: [AES.GCM]

PART 4
THE SECURITY OF TLS 1.3

THE SECURITY OF TLS 1.3 (FULL)



Client

Server

Pick N_C

Pick $\mathbf{G}_1, \mathbf{G}_2 \dots \mathbf{G}_n$

List is standardized (only safe groups)

$x_1, x_2 \dots x_n$

Set $\mathbf{KE}_{C,i} = (\mathbf{G}_i, g_i^{x_i})$

$$\xrightarrow{N_C,\ \mathbf{KE}_{C,1} \dots \mathbf{KE}_{C,n},\ \mathrm{ext}}$$

Pick N_S, pick one $\mathbf{G}_j$

Pick y, set $KE_S = g_j^y$

$$\xleftarrow{N_S,\ \mathbf{G}_j,\ \mathrm{KE}_S,\ \mathrm{ext}}$$

Do: $\mathrm{ES} = \mathrm{KE}_S^{x_j}$

Do: $\mathrm{ES} = \mathrm{KE}_{C,j}^{y}$

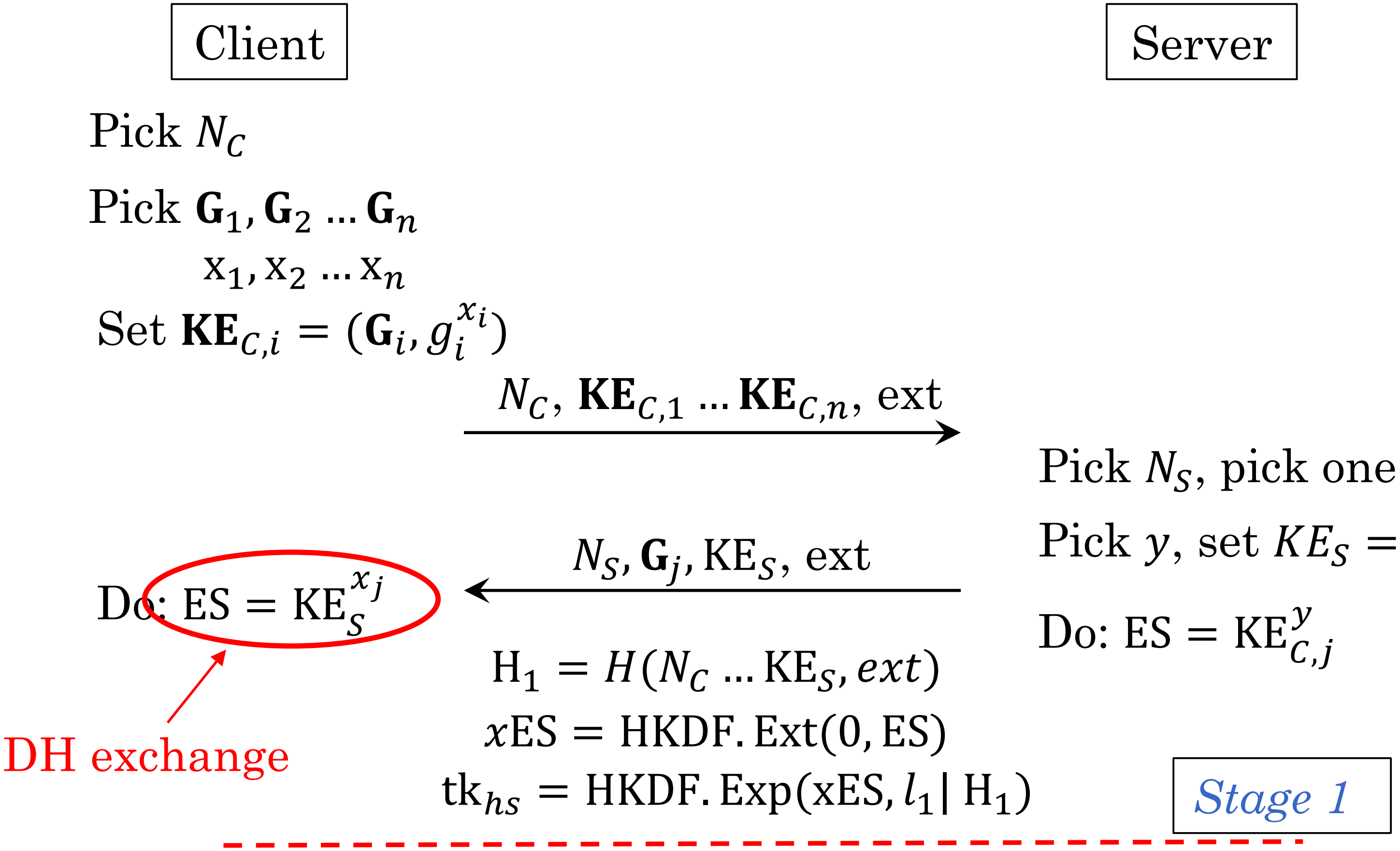
$$\mathrm{H}_1 = H(N_C \dots \mathrm{KE}_S, ext)$$

$$x\mathrm{ES} = \mathrm{HKDF.Ext}(0, \mathrm{ES})$$

$$\mathrm{tk}_{hs} = \mathrm{HKDF.Exp}(\mathrm{xES}, l_1 | \mathrm{H}_1)$$

Stage 1

THE SECURITY OF TLS 1.3 (FULL)



Client

Pick N_C

Pick $\mathbf{G}_1, \mathbf{G}_2 \dots \mathbf{G}_n$

$\mathrm{x}_1, \mathrm{x}_2 \dots \mathrm{x}_n$

Set $\mathbf{KE}_{C,i} = (\mathbf{G}_i, g_i^{x_i})$

$$\xrightarrow{N_C,\ \mathbf{KE}_{C,1} \dots \mathbf{KE}_{C,n},\ \mathrm{ext}}$$

Server

Pick N_S, pick one $\mathbf{G}_j$

Pick y, set $KE_S = g_j^y$

$$\xleftarrow{N_S, \mathbf{G}_j, \mathrm{KE}_S,\ \mathrm{ext}}$$

Do: $\mathrm{ES} = \mathrm{KE}_S^{x_j}$

Do: $\mathrm{ES} = \mathrm{KE}_{C,j}^{y}$

DH exchange

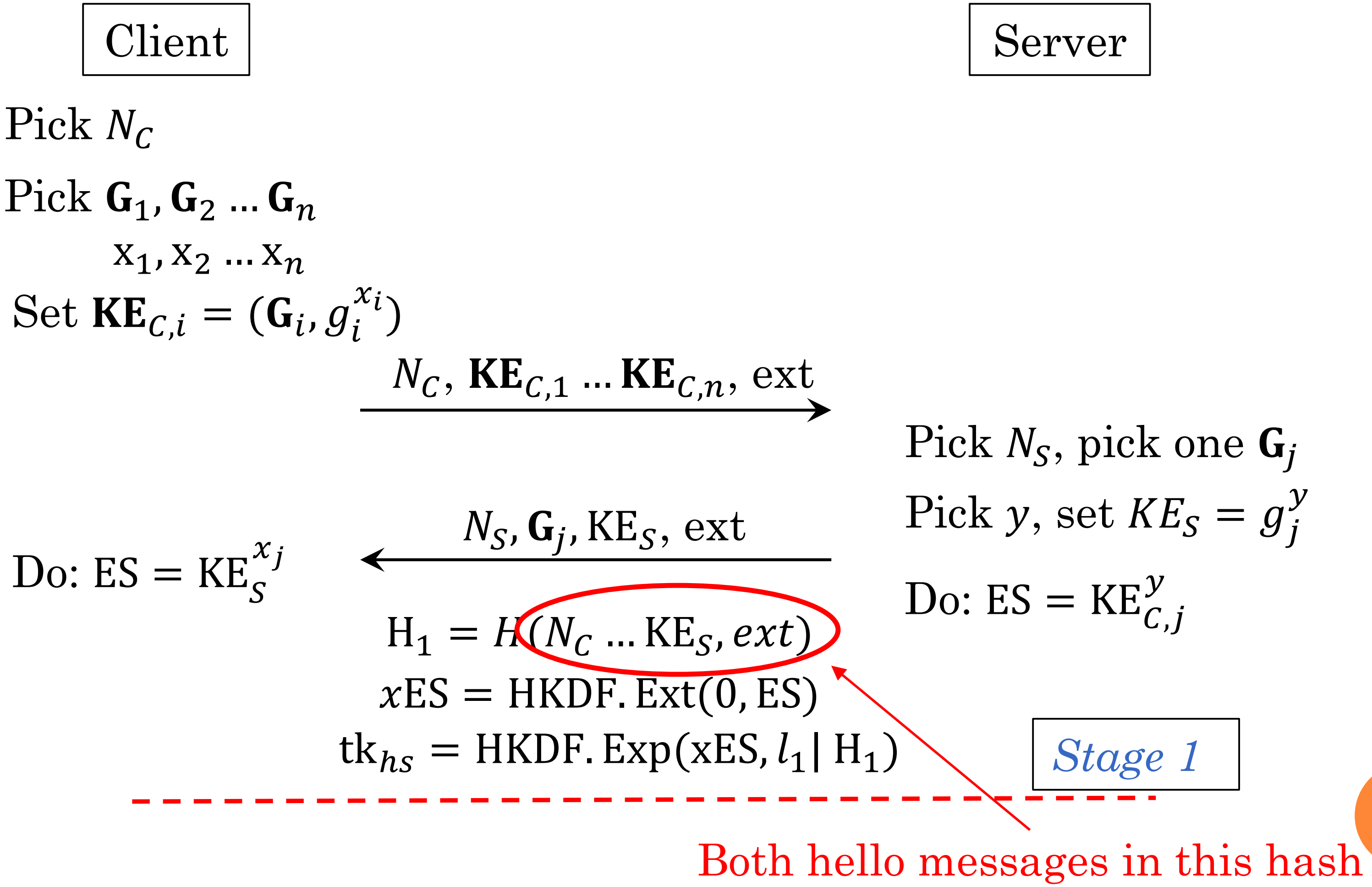
$$\mathrm{H}_1 = H(N_C \dots \mathrm{KE}_S, ext)$$

$$x\mathrm{ES} = \mathrm{HKDF.Ext}(0, \mathrm{ES})$$

$$\mathrm{tk}_{hs} = \mathrm{HKDF.Exp}(\mathrm{xES}, l_1 | \mathrm{H}_1)$$

Stage 1

THE SECURITY OF TLS 1.3 (FULL)



Client

Pick N_C

Pick $\mathbf{G}_1, \mathbf{G}_2 \ldots \mathbf{G}_n$

$\mathrm{x}_1, \mathrm{x}_2 \ldots \mathrm{x}_n$

Set $\mathbf{KE}_{C,i} = (\mathbf{G}_i, g_i^{x_i})$

$$\xrightarrow{N_C,\ \mathbf{KE}_{C,1} \ldots \mathbf{KE}_{C,n},\ \text{ext}}$$

Server

Pick N_S, pick one $\mathbf{G}_j$

Pick y, set $KE_S = g_j^y$

$$\xleftarrow{N_S,\ \mathbf{G}_j,\ \mathrm{KE}_S,\ \text{ext}}$$

Do: $\mathrm{ES} = \mathrm{KE}_S^{x_j}$

Do: $\mathrm{ES} = \mathrm{KE}_{C,j}^{y}$

$$\mathrm{H}_1 = H(N_C \ldots \mathrm{KE}_S, ext)$$

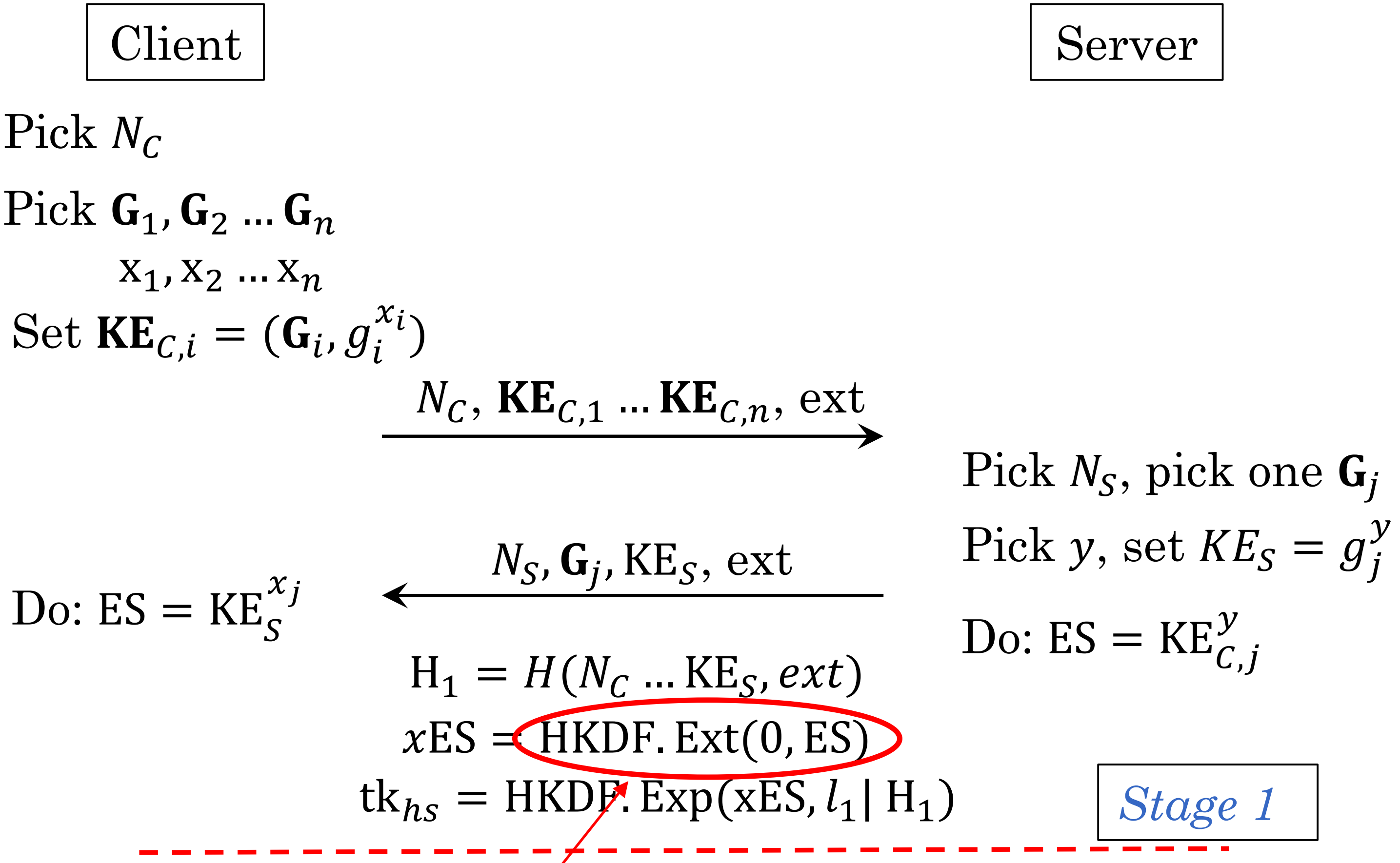
$$x\mathrm{ES} = \mathrm{HKDF.Ext}(0, \mathrm{ES})$$

$$\mathrm{tk}_{hs} = \mathrm{HKDF.Exp}(\mathrm{xES}, l_1 | \mathrm{H}_1)$$

Stage 1

Both hello messages in this hash

THE SECURITY OF TLS 1.3 (FULL)



Client

Server

Pick N_C

Pick $\mathbf{G}_1, \mathbf{G}_2 \ldots \mathbf{G}_n$

$x_1, x_2 \ldots x_n$

Set $\mathbf{KE}_{C,i} = (\mathbf{G}_i, g_i^{x_i})$

$$\xrightarrow{N_C,\ \mathbf{KE}_{C,1} \ldots \mathbf{KE}_{C,n},\ \text{ext}}$$

Pick N_S, pick one $\mathbf{G}_j$

Pick y, set $KE_S = g_j^y$

$$\xleftarrow{N_S,\ \mathbf{G}_j,\ \text{KE}_S,\ \text{ext}}$$

Do: $\text{ES} = \text{KE}_S^{x_j}$

Do: $\text{ES} = \text{KE}_{C,j}^{y}$

$$\text{H}_1 = H(N_C \ldots \text{KE}_S, ext)$$

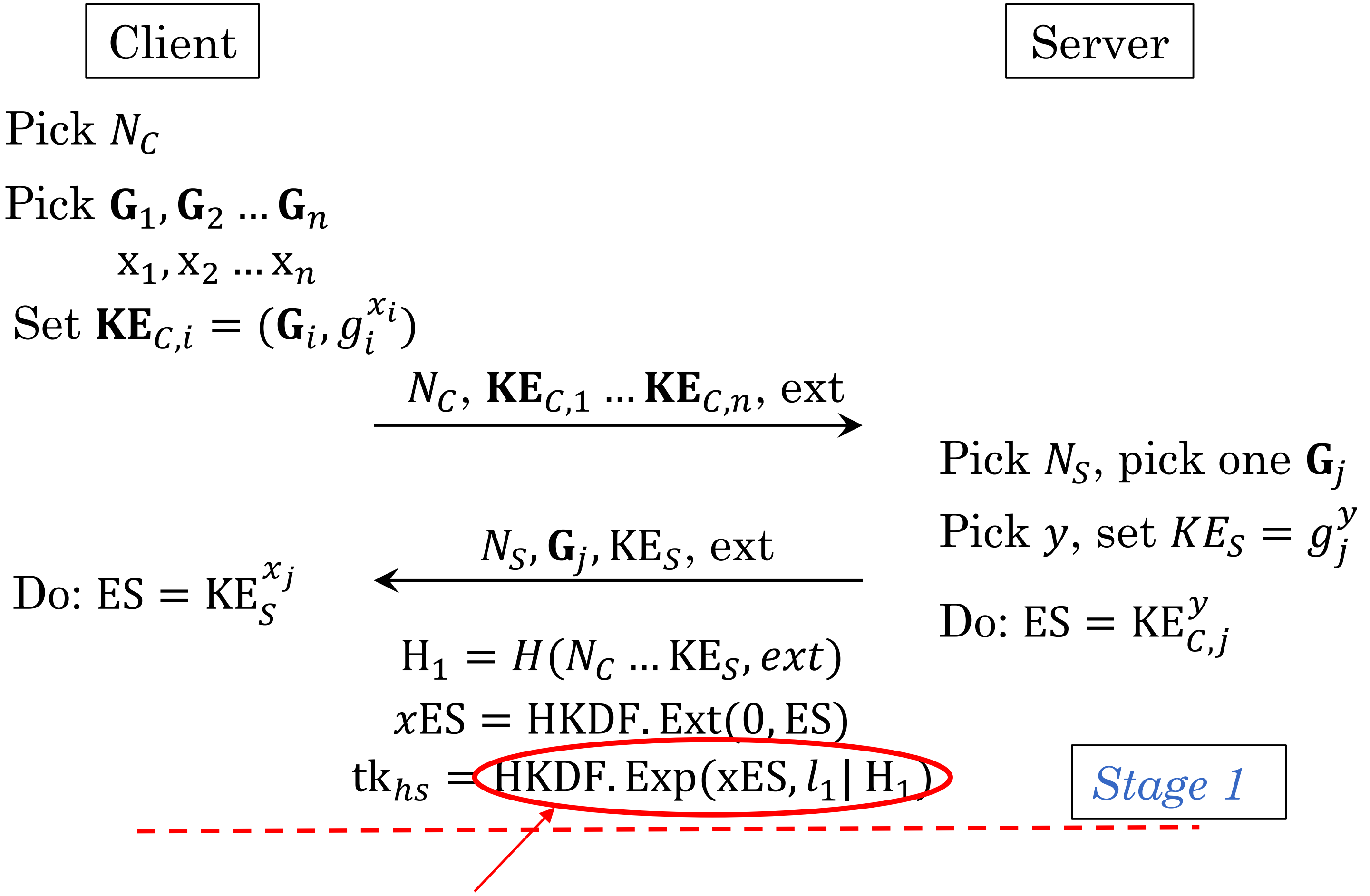
$$x\text{ES} = \text{HKDF.Ext}(0, \text{ES})$$

$$\text{tk}_{hs} = \text{HKDF.Exp}(\text{xES}, l_1 | \text{H}_1)$$

Stage 1

Extract with 0 salt: secure only in the ROM

THE SECURITY OF TLS 1.3 (FULL)



Client

Server

Pick N_C

Pick $\mathbf{G}_1, \mathbf{G}_2 \dots \mathbf{G}_n$

$\mathrm{x}_1, \mathrm{x}_2 \dots \mathrm{x}_n$

Set $\mathbf{KE}_{C,i} = (\mathbf{G}_i, g_i^{x_i})$

$$\xrightarrow{N_C,\ \mathbf{KE}_{C,1} \dots \mathbf{KE}_{C,n},\ \text{ext}}$$

Pick N_S, pick one $\mathbf{G}_j$

Pick y, set $KE_S = g_j^y$

$$\xleftarrow{N_S, \mathbf{G}_j, \mathrm{KE}_S,\ \text{ext}}$$

Do: $\mathrm{ES} = \mathrm{KE}_S^{x_j}$

Do: $\mathrm{ES} = \mathrm{KE}_{C,j}^{y}$

$\mathrm{H}_1 = H(N_C \dots \mathrm{KE}_S, ext)$

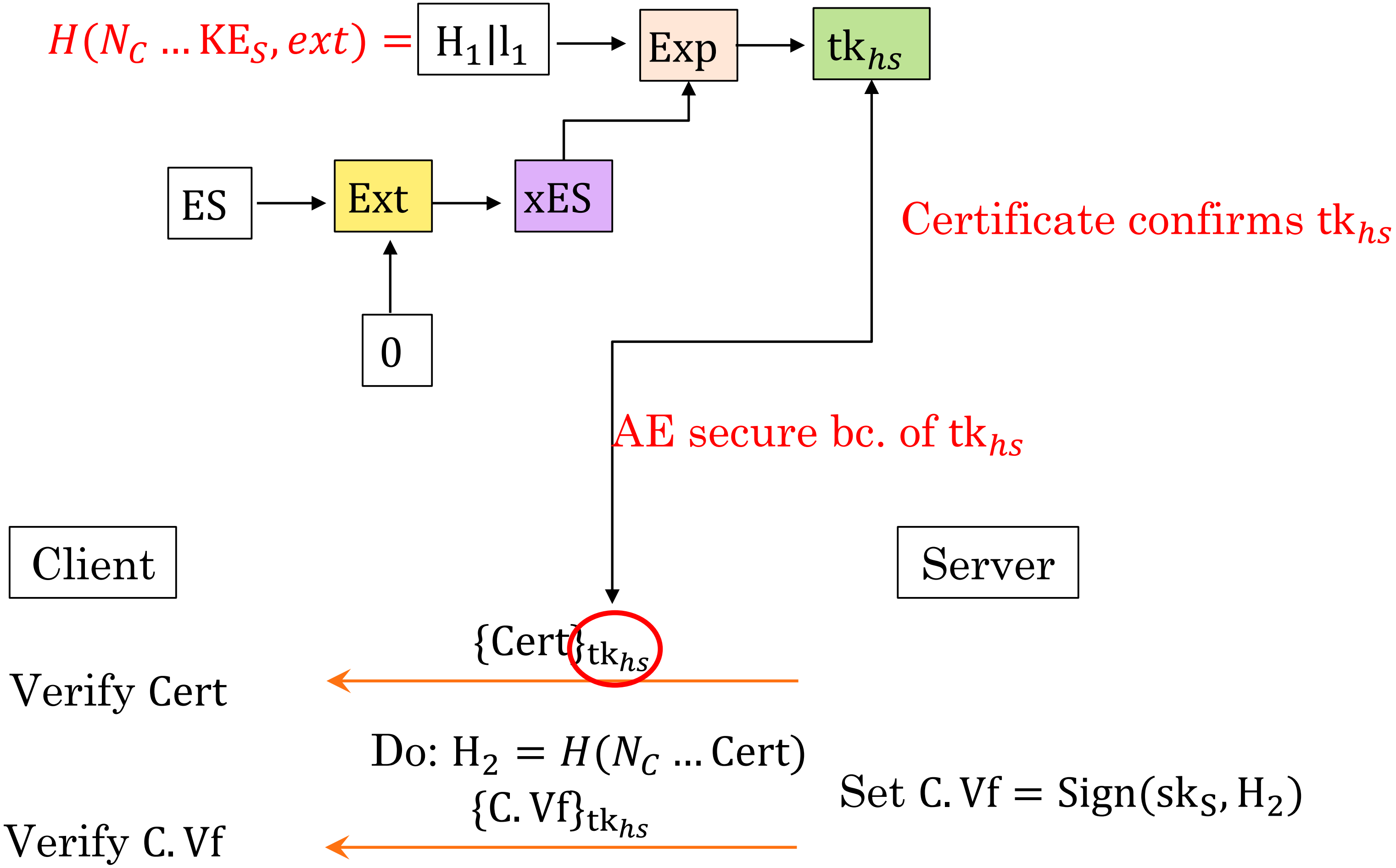
$x\mathrm{ES} = \mathrm{HKDF.Ext}(0, \mathrm{ES})$

$\mathrm{tk}_{hs} = \mathrm{HKDF.Exp}(\mathrm{xES}, l_1 | \mathrm{H}_1)$

Stage 1

Handshake keys based on entire handshake so far

KEY CONFIRMATION



$H(N_C \ldots \mathrm{KE}_S, ext) =$ $\mathrm{H}_1||\mathrm{l}_1$

Exp

tk_{hs}

ES

Ext

xES

0

Certificate confirms tk_{hs}

AE secure bc. of tk_{hs}

Client

Server

$\{\mathrm{Cert}\}_{\mathrm{tk}_{hs}}$

Verify Cert

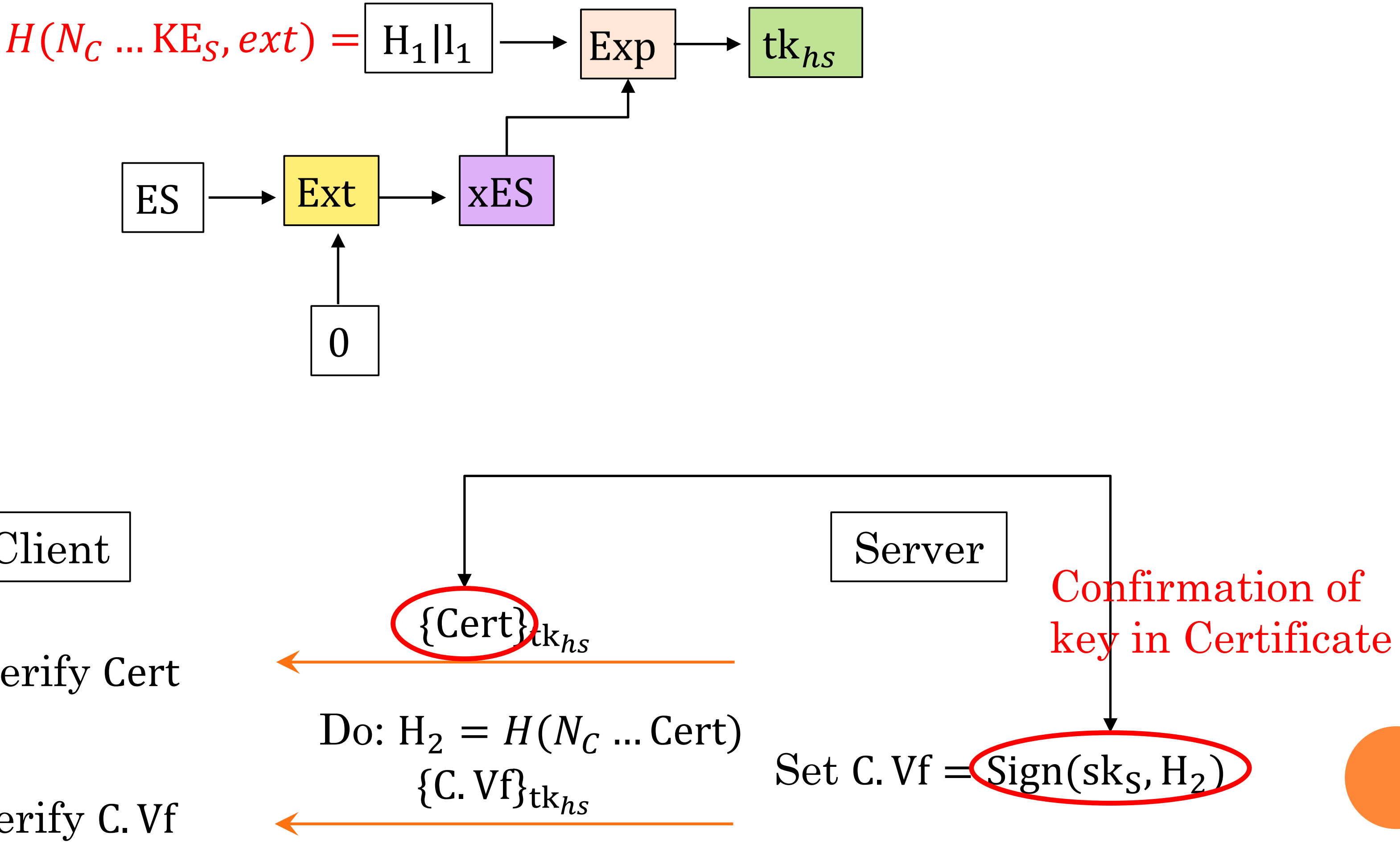
Do: $\mathrm{H}_2 = H(N_C \ldots \mathrm{Cert})$

$\{\mathrm{C.Vf}\}_{\mathrm{tk}_{hs}}$

Set $\mathrm{C.Vf} = \mathrm{Sign}(\mathrm{sk}_S, \mathrm{H}_2)$

Verify $\mathrm{C.Vf}$

KEY CONFIRMATION



$H(N_C \ldots \mathrm{KE}_S, ext) = \mathrm{H}_1||\mathrm{l}_1$ → Exp → tk_{hs}

ES → Ext → xES → Exp

0 → Ext

Client

Server

$\{\mathrm{Cert}\}_{\mathrm{tk}_{hs}}$

Verify Cert

Confirmation of key in Certificate

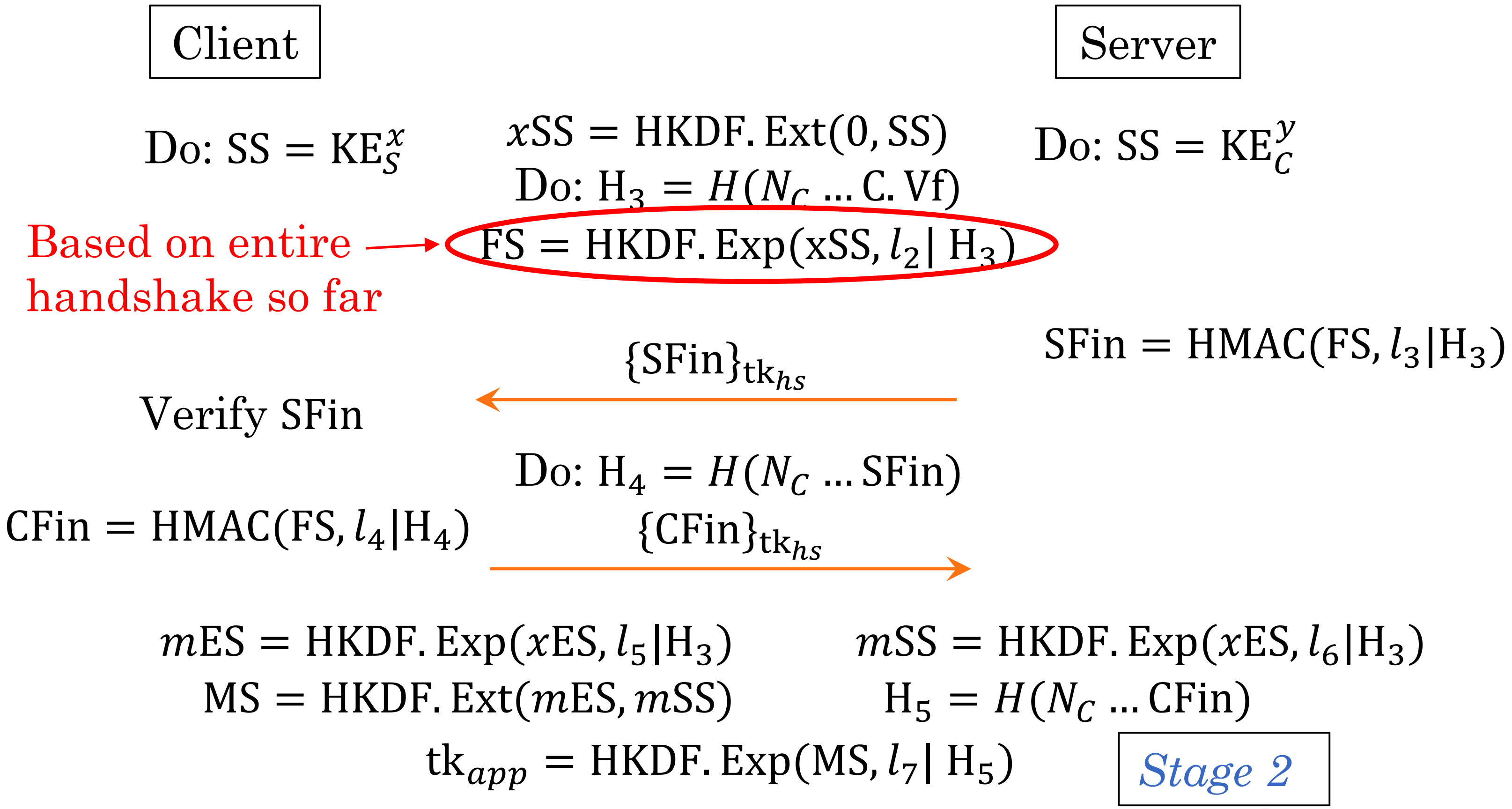
Do: $\mathrm{H}_2 = H(N_C \ldots \mathrm{Cert})$

Set $\mathrm{C.Vf} = \mathrm{Sign}(\mathrm{sk}_S, \mathrm{H}_2)$

$\{\mathrm{C.Vf}\}_{\mathrm{tk}_{hs}}$

Verify $\mathrm{C.Vf}$

KEY CONFIRMATION



Client

Server

Do: $\mathrm{SS} = \mathrm{KE}_S^x$

$x\mathrm{SS} = \mathrm{HKDF.Ext}(0, \mathrm{SS})$

Do: $\mathrm{SS} = \mathrm{KE}_C^y$

Do: $\mathrm{H}_3 = H(N_C \ldots \mathrm{C.Vf})$

Based on entire handshake so far → $\mathrm{FS} = \mathrm{HKDF.Exp}(\mathrm{xSS}, l_2 |\ \mathrm{H}_3)$

$\mathrm{SFin} = \mathrm{HMAC}(\mathrm{FS}, l_3 | \mathrm{H}_3)$

$\xleftarrow{\{\mathrm{SFin}\}_{\mathrm{tk}_{hs}}}$

Verify SFin

Do: $\mathrm{H}_4 = H(N_C \ldots \mathrm{SFin})$

$\mathrm{CFin} = \mathrm{HMAC}(\mathrm{FS}, l_4 | \mathrm{H}_4)$

$\xrightarrow{\{\mathrm{CFin}\}_{\mathrm{tk}_{hs}}}$

$m\mathrm{ES} = \mathrm{HKDF.Exp}(x\mathrm{ES}, l_5 | \mathrm{H}_3)$

$m\mathrm{SS} = \mathrm{HKDF.Exp}(x\mathrm{ES}, l_6 | \mathrm{H}_3)$

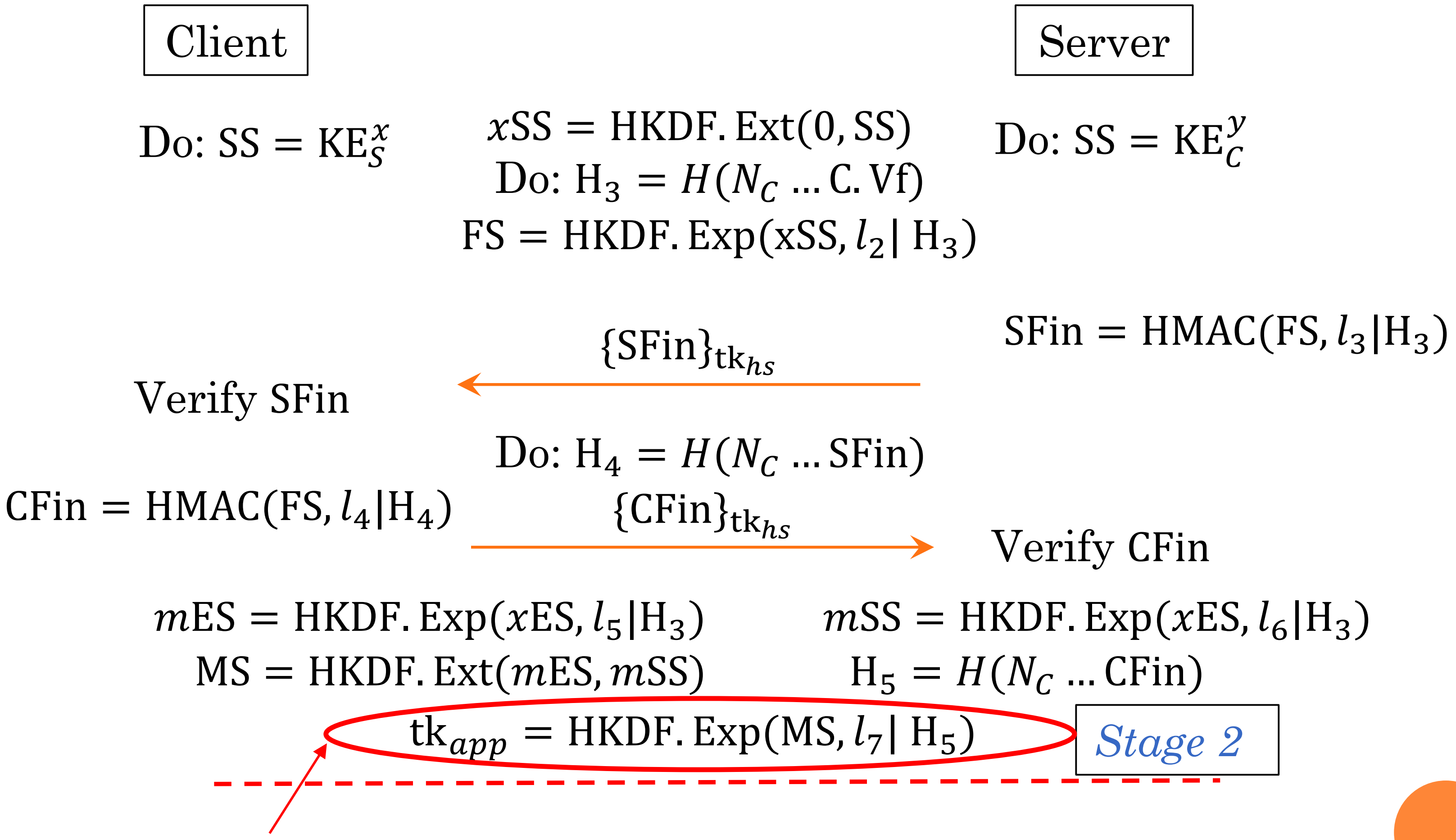
$\mathrm{MS} = \mathrm{HKDF.Ext}(m\mathrm{ES}, m\mathrm{SS})$

$\mathrm{H}_5 = H(N_C \ldots \mathrm{CFin})$

$\mathrm{tk}_{app} = \mathrm{HKDF.Exp}(\mathrm{MS}, l_7 |\ \mathrm{H}_5)$

Stage 2

KEY CONFIRMATION



Client

Server

Do: $\mathrm{SS} = \mathrm{KE}_S^x$

$x\mathrm{SS} = \mathrm{HKDF.Ext}(0, \mathrm{SS})$

Do: $\mathrm{H}_3 = H(N_C \ldots \mathrm{C.Vf})$

$\mathrm{FS} = \mathrm{HKDF.Exp}(\mathrm{xSS}, l_2 | \mathrm{H}_3)$

Do: $\mathrm{SS} = \mathrm{KE}_C^y$

$\mathrm{SFin} = \mathrm{HMAC}(\mathrm{FS}, l_3 | \mathrm{H}_3)$

$\{\mathrm{SFin}\}_{\mathrm{tk}_{hs}}$

Verify SFin

Do: $\mathrm{H}_4 = H(N_C \ldots \mathrm{SFin})$

$\mathrm{CFin} = \mathrm{HMAC}(\mathrm{FS}, l_4 | \mathrm{H}_4)$

$\{\mathrm{CFin}\}_{\mathrm{tk}_{hs}}$

Verify CFin

$m\mathrm{ES} = \mathrm{HKDF.Exp}(x\mathrm{ES}, l_5 | \mathrm{H}_3)$

$m\mathrm{SS} = \mathrm{HKDF.Exp}(x\mathrm{ES}, l_6 | \mathrm{H}_3)$

$\mathrm{MS} = \mathrm{HKDF.Ext}(m\mathrm{ES}, m\mathrm{SS})$

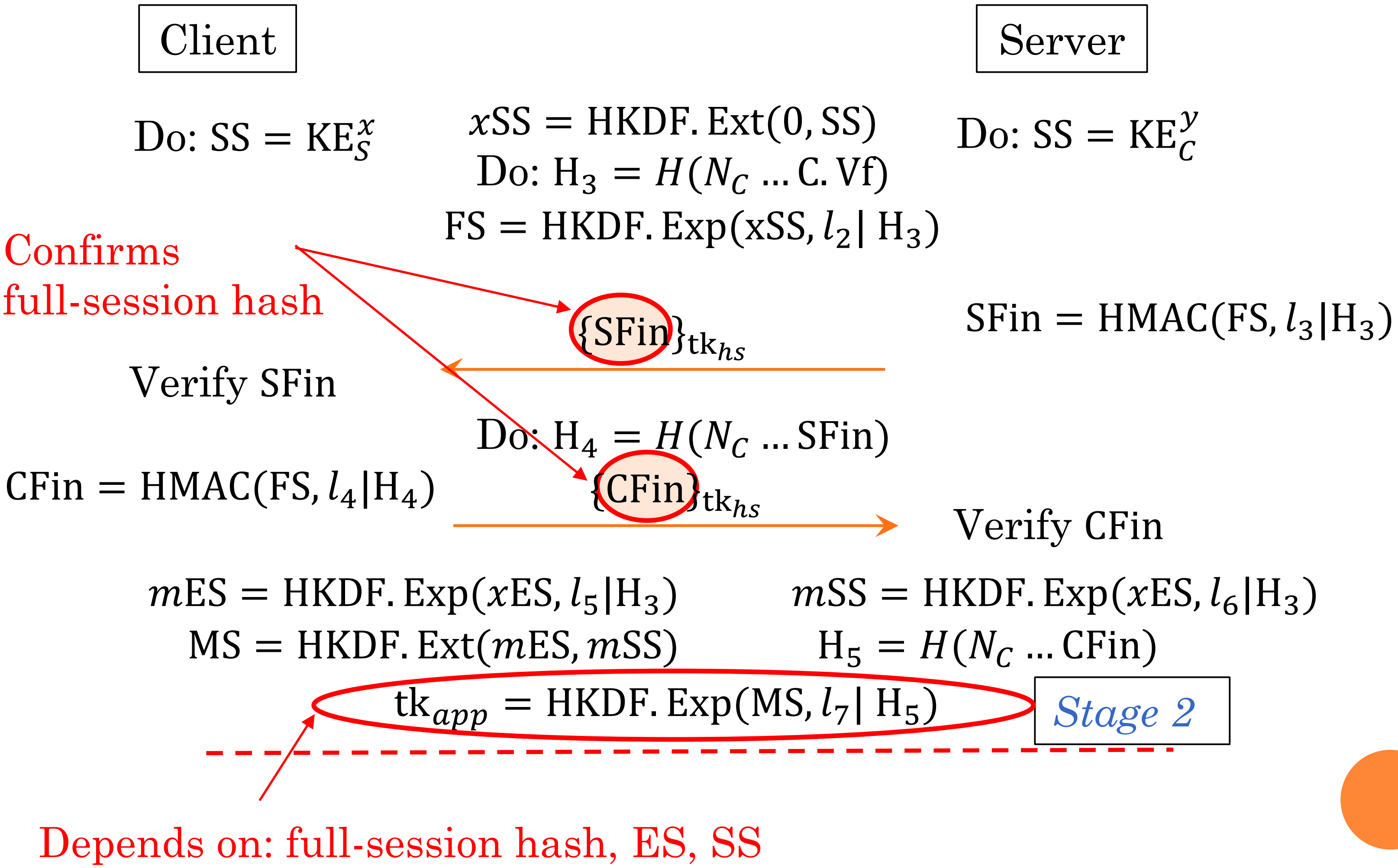
$\mathrm{H}_5 = H(N_C \ldots \mathrm{CFin})$

$\mathrm{tk}_{app} = \mathrm{HKDF.Exp}(\mathrm{MS}, l_7 | \mathrm{H}_5)$

Stage 2

Depends on: full-session hash, ES, SS

KEY CONFIRMATION



Client

Server

Do: $SS = KE_S^x$

$xSS = \text{HKDF.Ext}(0, SS)$

Do: $H_3 = H(N_C \ldots \text{C.Vf})$

$FS = \text{HKDF.Exp}(xSS, l_2 | H_3)$

Do: $SS = KE_C^y$

Confirms full-session hash

$\{\text{SFin}\}_{tk_{hs}}$

$\text{SFin} = \text{HMAC}(FS, l_3 | H_3)$

Verify SFin

Do: $H_4 = H(N_C \ldots \text{SFin})$

$\text{CFin} = \text{HMAC}(FS, l_4 | H_4)$

$\{\text{CFin}\}_{tk_{hs}}$

Verify CFin

$mES = \text{HKDF.Exp}(xES, l_5 | H_3)$

$MS = \text{HKDF.Ext}(mES, mSS)$

$mSS = \text{HKDF.Exp}(xES, l_6 | H_3)$

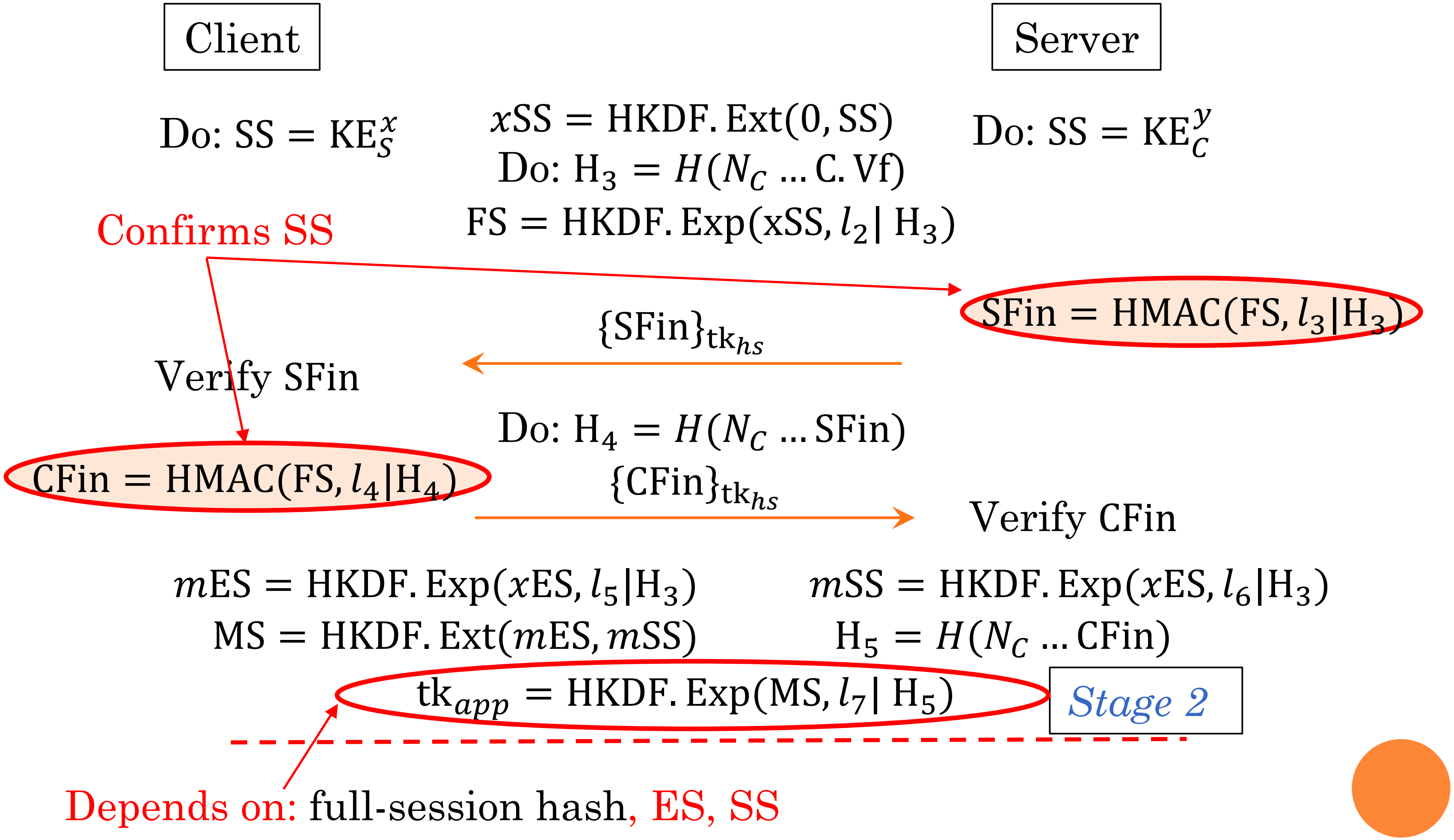
$H_5 = H(N_C \ldots \text{CFin})$

$tk_{app} = \text{HKDF.Exp}(MS, l_7 | H_5)$

Stage 2

Depends on: full-session hash, ES, SS

KEY CONFIRMATION



Client

Server

Do: $\mathrm{SS} = \mathrm{KE}_S^x$

$x\mathrm{SS} = \mathrm{HKDF.Ext}(0, \mathrm{SS})$

Do: $\mathrm{H}_3 = H(N_C \ldots \mathrm{C.Vf})$

$\mathrm{FS} = \mathrm{HKDF.Exp}(\mathrm{xSS}, l_2 | \mathrm{H}_3)$

Do: $\mathrm{SS} = \mathrm{KE}_C^y$

Confirms SS

$\mathrm{SFin} = \mathrm{HMAC}(\mathrm{FS}, l_3 | \mathrm{H}_3)$

$\xleftarrow{\{\mathrm{SFin}\}_{\mathrm{tk}_{hs}}}$

Verify SFin

Do: $\mathrm{H}_4 = H(N_C \ldots \mathrm{SFin})$

$\mathrm{CFin} = \mathrm{HMAC}(\mathrm{FS}, l_4 | \mathrm{H}_4)$

$\xrightarrow{\{\mathrm{CFin}\}_{\mathrm{tk}_{hs}}}$

Verify CFin

$m\mathrm{ES} = \mathrm{HKDF.Exp}(x\mathrm{ES}, l_5 | \mathrm{H}_3)$

$m\mathrm{SS} = \mathrm{HKDF.Exp}(x\mathrm{ES}, l_6 | \mathrm{H}_3)$

$\mathrm{MS} = \mathrm{HKDF.Ext}(m\mathrm{ES}, m\mathrm{SS})$

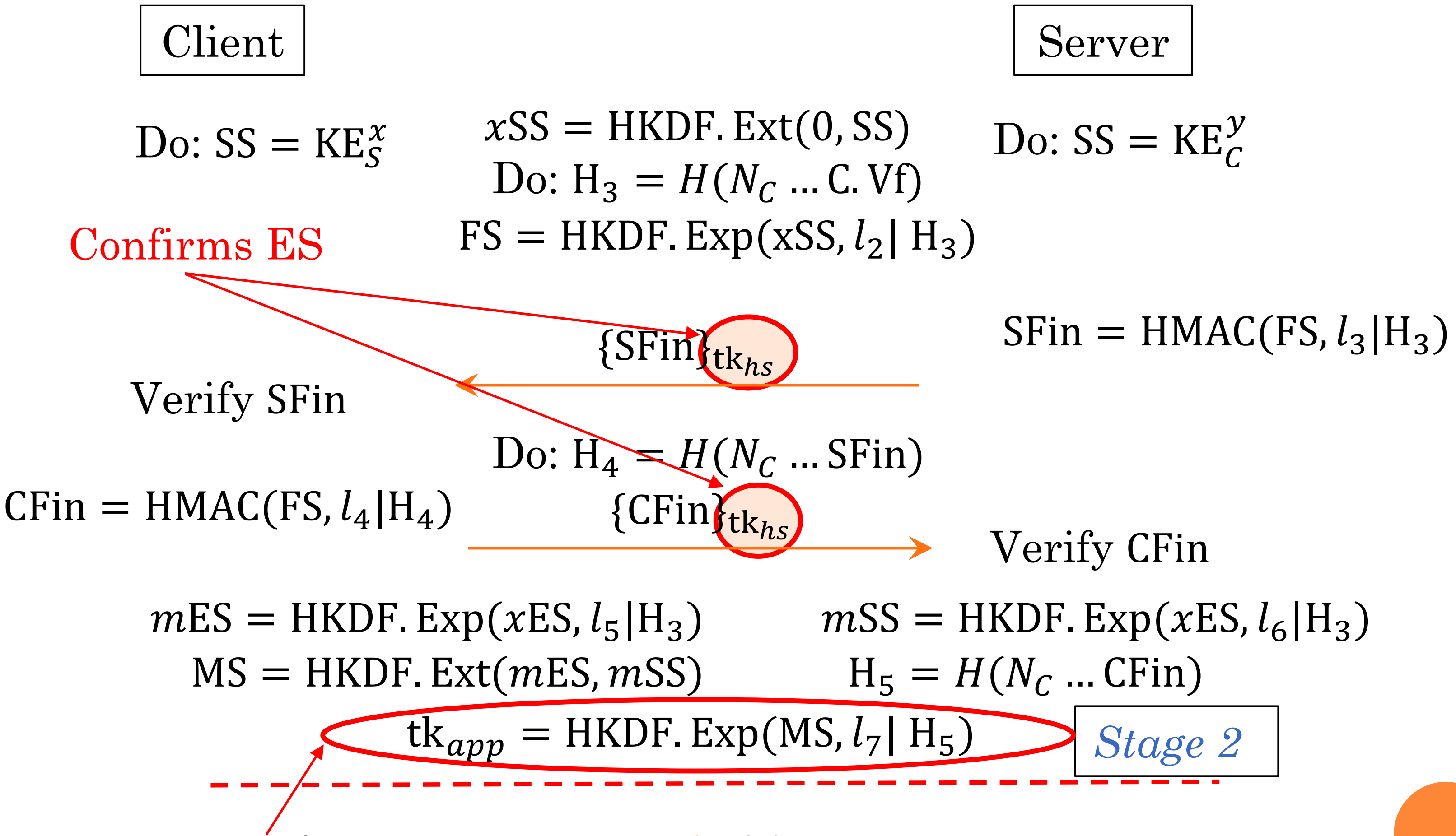
$\mathrm{H}_5 = H(N_C \ldots \mathrm{CFin})$

$\mathrm{tk}_{app} = \mathrm{HKDF.Exp}(\mathrm{MS}, l_7 | \mathrm{H}_5)$

Stage 2

Depends on: full-session hash, ES, SS

KEY CONFIRMATION



Client

Server

Do: $SS = KE_S^x$

$xSS = \mathrm{HKDF.Ext}(0, SS)$

Do: $SS = KE_C^y$

Do: $H_3 = H(N_C \ldots \mathrm{C.Vf})$

$FS = \mathrm{HKDF.Exp}(xSS, l_2 | H_3)$

Confirms ES

$\{SFin\}_{tk_{hs}}$

$SFin = \mathrm{HMAC}(FS, l_3 | H_3)$

Verify SFin

Do: $H_4 = H(N_C \ldots SFin)$

$CFin = \mathrm{HMAC}(FS, l_4 | H_4)$

$\{CFin\}_{tk_{hs}}$

Verify CFin

$mES = \mathrm{HKDF.Exp}(xES, l_5 | H_3)$

$mSS = \mathrm{HKDF.Exp}(xES, l_6 | H_3)$

$MS = \mathrm{HKDF.Ext}(mES, mSS)$

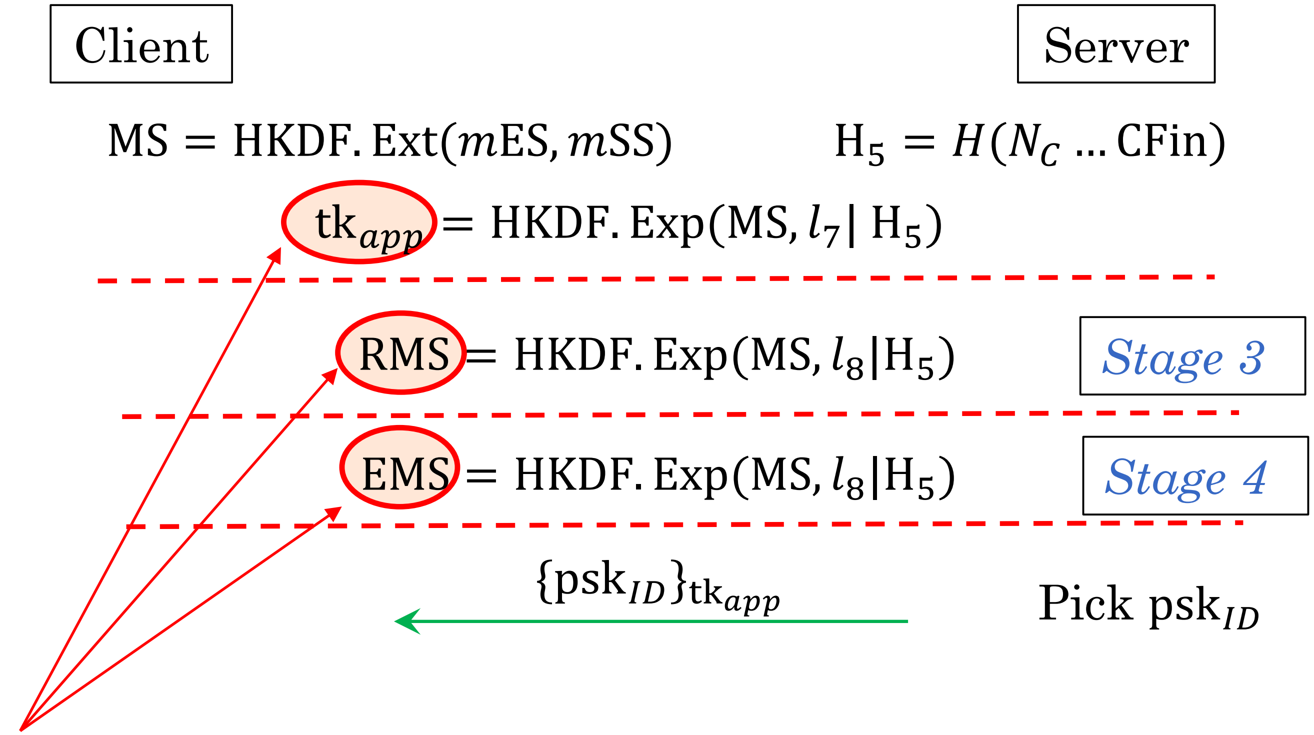
$H_5 = H(N_C \ldots CFin)$

$tk_{app} = \mathrm{HKDF.Exp}(MS, l_7 | H_5)$

Stage 2

Depends on: full-session hash, ES, SS

STAGES 3 AND 4



Client

Server

$\mathrm{MS} = \mathrm{HKDF.Ext}(m\mathrm{ES}, m\mathrm{SS})$

$\mathrm{H}_5 = H(N_C \ldots \mathrm{CFin})$

$\mathrm{tk}_{app} = \mathrm{HKDF.Exp}(\mathrm{MS}, l_7 |\ \mathrm{H}_5)$

$\mathrm{RMS} = \mathrm{HKDF.Exp}(\mathrm{MS}, l_8 | \mathrm{H}_5)$

Stage 3

$\mathrm{EMS} = \mathrm{HKDF.Exp}(\mathrm{MS}, l_8 | \mathrm{H}_5)$

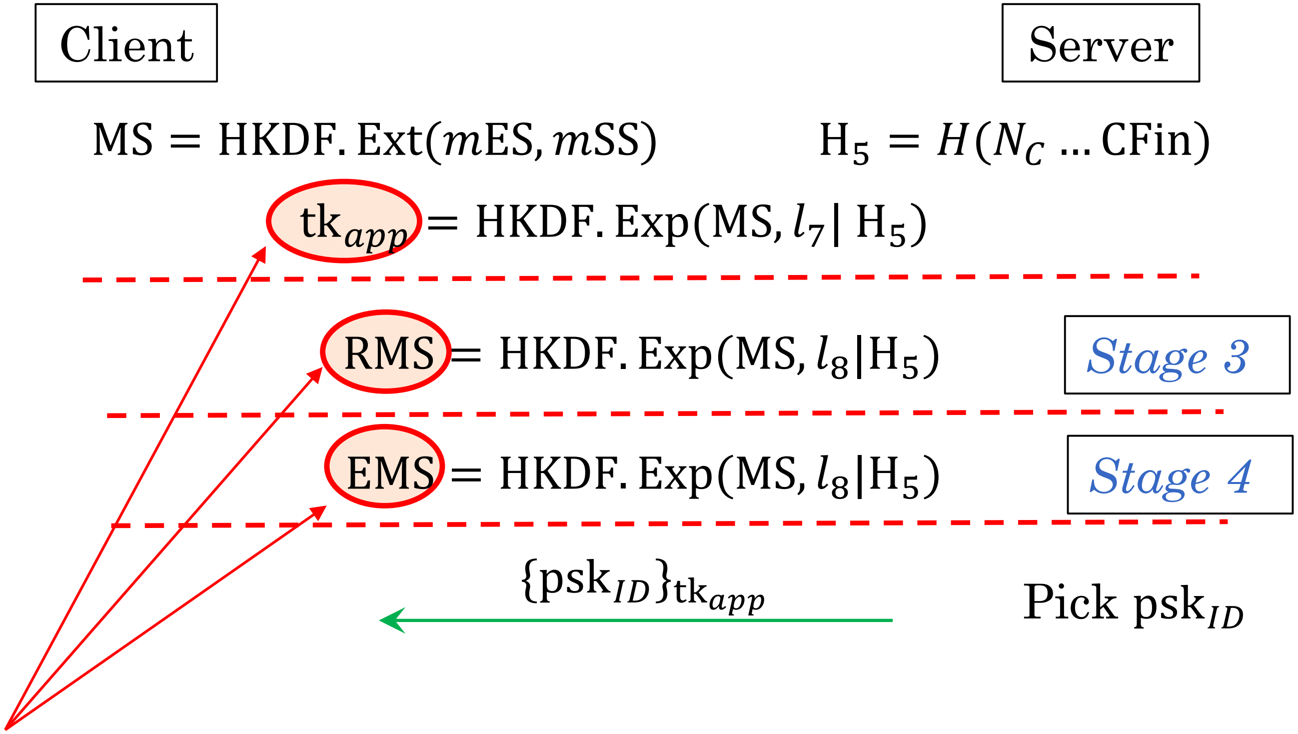
Stage 4

$\{\mathrm{psk}_{ID}\}_{\mathrm{tk}_{app}}$

Pick psk_{ID}

Computed from same MS value
Independent labels => independent keys
Hard to retrieve MS from any of these keys

Stages 3 and 4



Client

Server

$\mathrm{MS} = \mathrm{HKDF.Ext}(m\mathrm{ES}, m\mathrm{SS})$

$\mathrm{H}_5 = H(N_C \dots \mathrm{CFin})$

$\mathrm{tk}_{app} = \mathrm{HKDF.Exp}(\mathrm{MS}, l_7 |\ \mathrm{H}_5)$

$\mathrm{RMS} = \mathrm{HKDF.Exp}(\mathrm{MS}, l_8 | \mathrm{H}_5)$

Stage 3

$\mathrm{EMS} = \mathrm{HKDF.Exp}(\mathrm{MS}, l_8 | \mathrm{H}_5)$

Stage 4

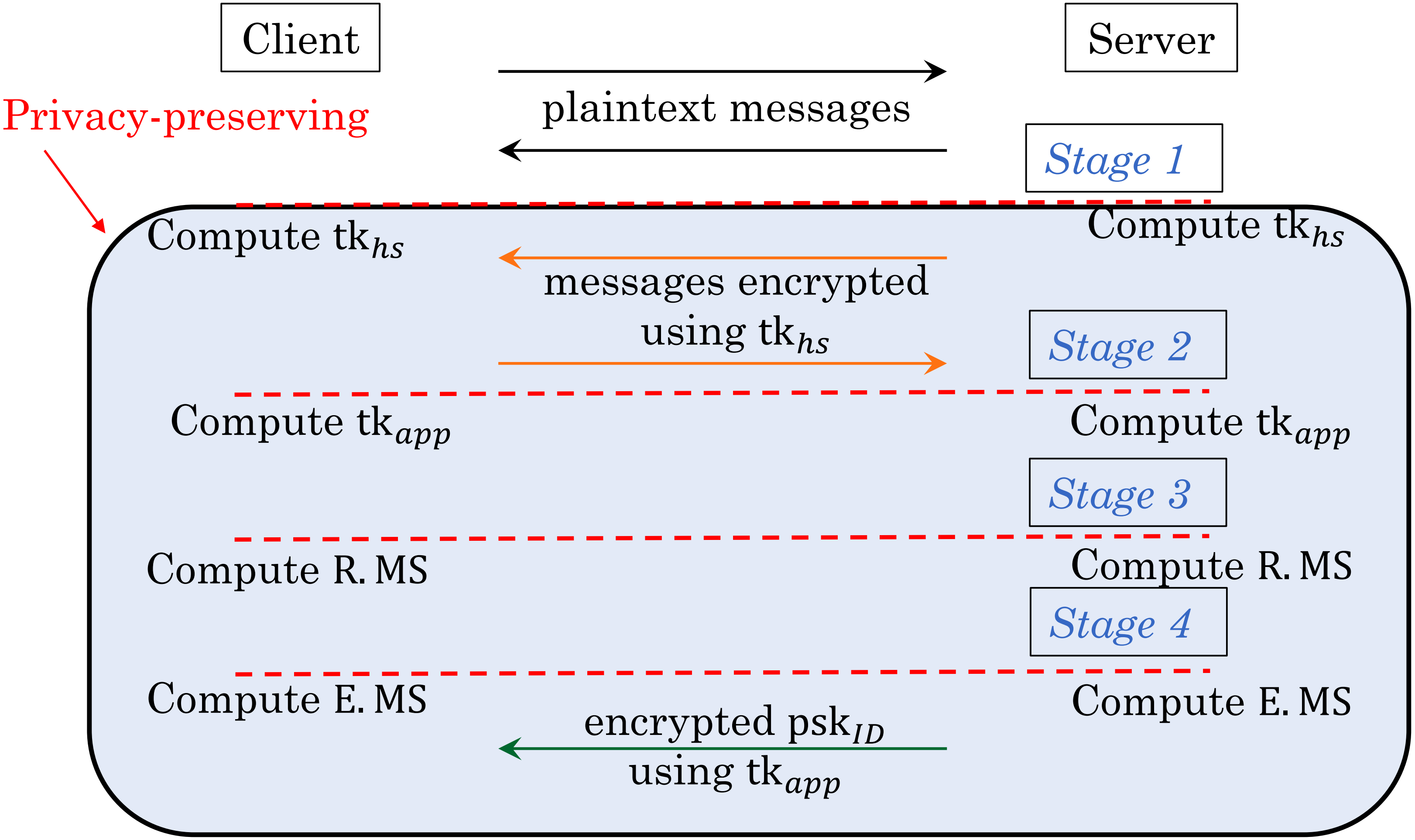
$\{\mathrm{psk}_{ID}\}_{\mathrm{tk}_{app}}$

Pick psk_{ID}

Computed from same MS value
Independent labels => independent keys
Hard to retrieve MS from any of these keys

PRIVACY PRESERVATION

- Several *stages*, one stage per key:



Client

Server

Privacy-preserving

plaintext messages

Stage 1

Compute tk_{hs}

Compute tk_{hs}

messages encrypted using tk_{hs}

Stage 2

Compute tk_{app}

Compute tk_{app}

Stage 3

Compute R. MS

Compute R. MS

Stage 4

Compute E. MS

Compute E. MS

encrypted psk_{ID} using tk_{app}

PART 5
SESSION RESUMPTION & 0-RTT

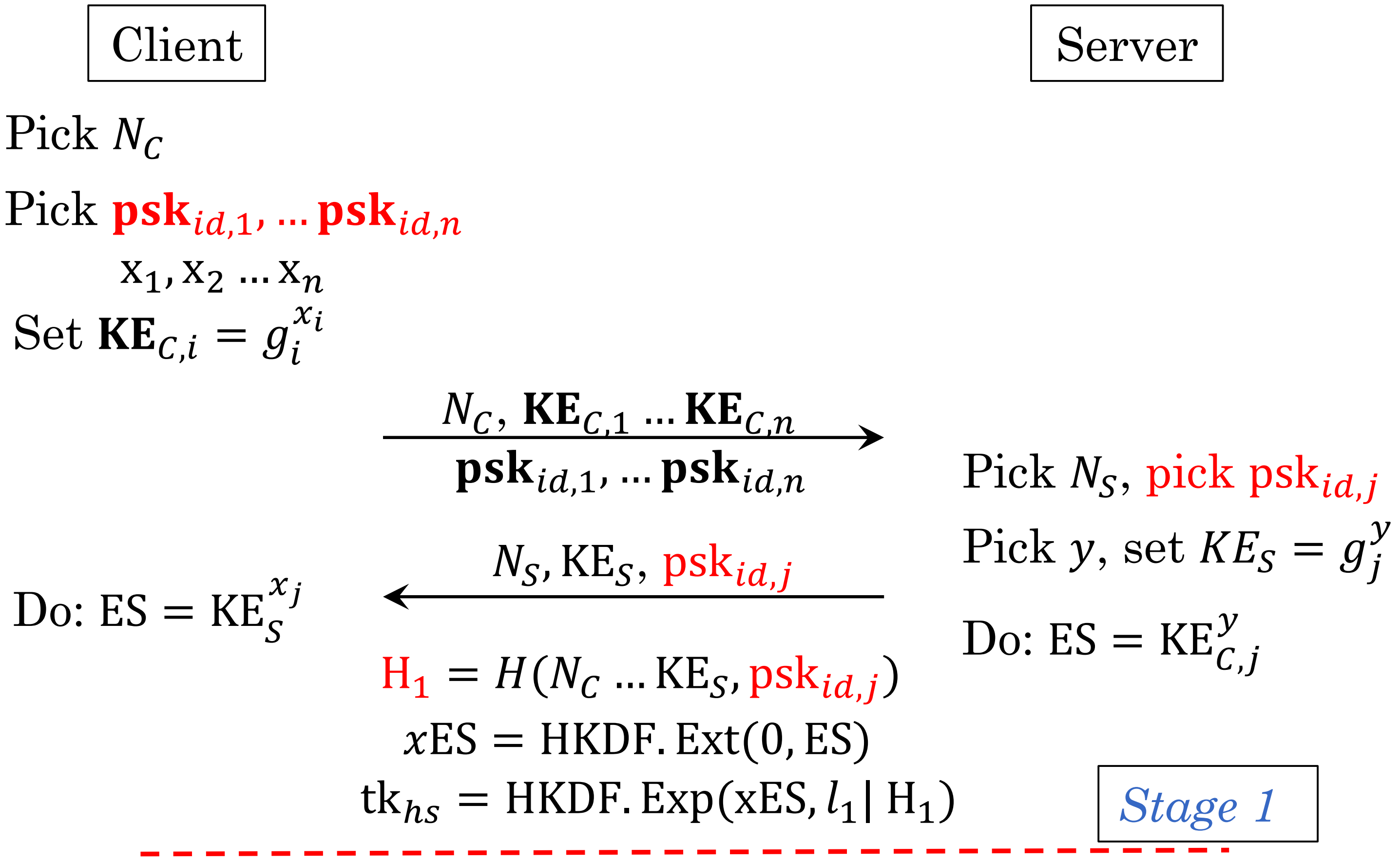
TWO TYPES OF SESSION RESUMPTION

- Simple Pre-Shared-Key (PSK) mode:
  - Client asks for PSK mode
  - Server sends a psk_{id} value, associated with RMS
  - For that handshake: ES = SS = RMS
  - Handshakes change a little (include psk_{id})

- PSK + DHE mode:
  - Start as in PSK mode (sending psk_{id})
  - Hybrid mode: also send g^x, g^y (same group as in psk_{id})
  - ES computed as in full handshake, and SS = RMS

STAGE 1 OF PSK+DHE

- Stage 1: handshake keys



Client

Server

Pick N_C

Pick $\mathbf{psk}_{id,1}, \dots \mathbf{psk}_{id,n}$

$\mathrm{x}_1, \mathrm{x}_2 \dots \mathrm{x}_n$

Set $\mathbf{KE}_{C,i} = g_i^{x_i}$

$$\xrightarrow[\mathbf{psk}_{id,1}, \dots \mathbf{psk}_{id,n}]{N_C, \mathbf{KE}_{C,1} \dots \mathbf{KE}_{C,n}}$$

Pick N_S, pick $\mathrm{psk}_{id,j}$

Pick y, set $KE_S = g_j^y$

$$\xleftarrow{N_S, \mathrm{KE}_S, \mathrm{psk}_{id,j}}$$

Do: $\mathrm{ES} = \mathrm{KE}_S^{x_j}$

Do: $\mathrm{ES} = \mathrm{KE}_{C,j}^{y}$

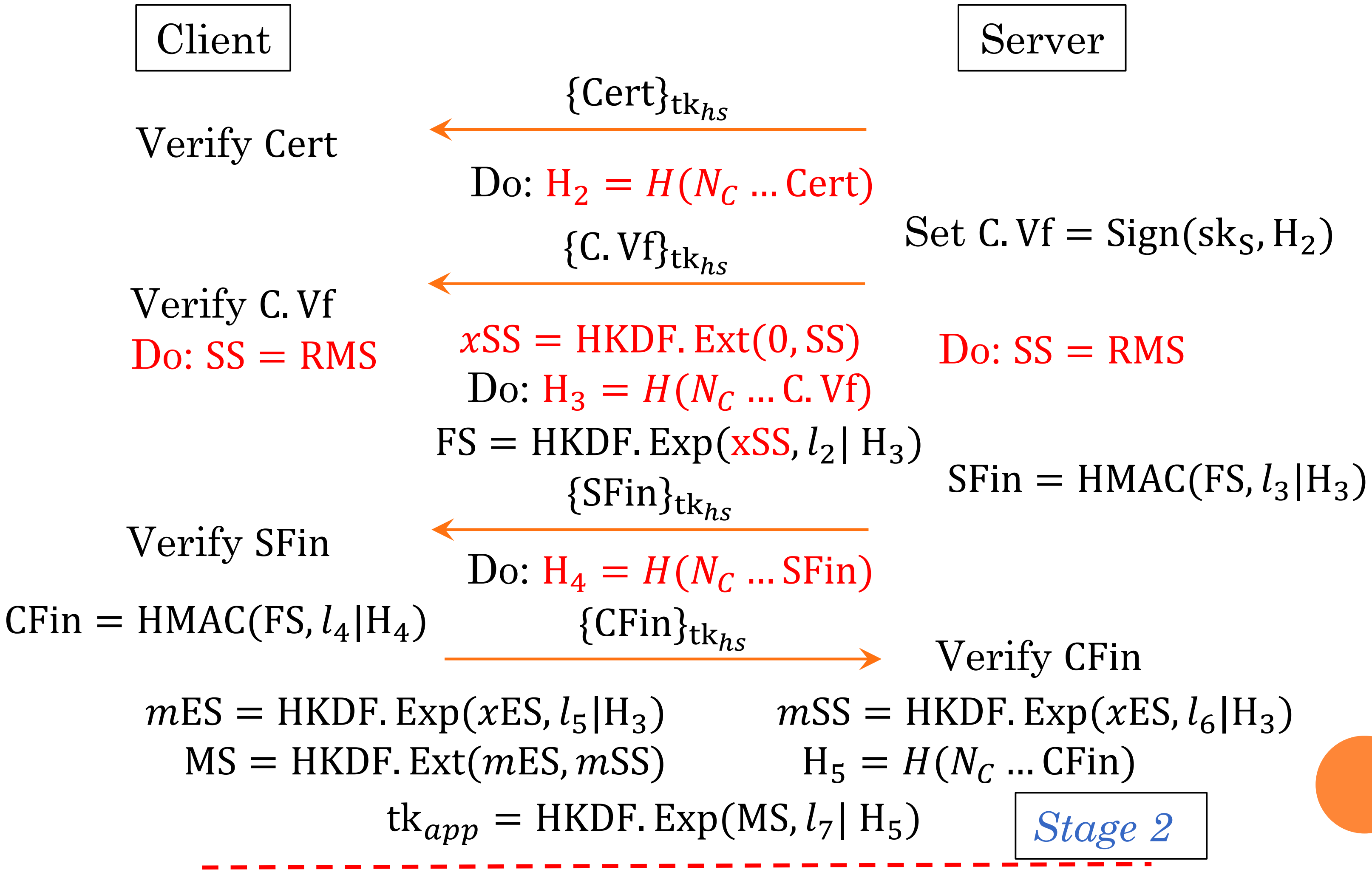
$$\mathrm{H}_1 = H(N_C \dots \mathrm{KE}_S, \mathrm{psk}_{id,j})$$

$$x\mathrm{ES} = \mathrm{HKDF.Ext}(0, \mathrm{ES})$$

$$\mathrm{tk}_{hs} = \mathrm{HKDF.Exp}(\mathrm{xES}, l_1 | \, \mathrm{H}_1)$$

Stage 1

STAGE 2 OF PSK + DHE



Client

Server

$\{\text{Cert}\}_{\text{tk}_{hs}}$

Verify Cert

Do: $H_2 = H(N_C \ldots \text{Cert})$

Set $\text{C.Vf} = \text{Sign}(\text{sk}_S, H_2)$

$\{\text{C.Vf}\}_{\text{tk}_{hs}}$

Verify C.Vf

Do: SS = RMS

$x\text{SS} = \text{HKDF.Ext}(0, \text{SS})$

Do: SS = RMS

Do: $H_3 = H(N_C \ldots \text{C.Vf})$

$\text{FS} = \text{HKDF.Exp}(\text{xSS}, l_2 | H_3)$

$\text{SFin} = \text{HMAC}(\text{FS}, l_3 | H_3)$

$\{\text{SFin}\}_{\text{tk}_{hs}}$

Verify SFin

Do: $H_4 = H(N_C \ldots \text{SFin})$

$\text{CFin} = \text{HMAC}(\text{FS}, l_4 | H_4)$

$\{\text{CFin}\}_{\text{tk}_{hs}}$

Verify CFin

$m\text{ES} = \text{HKDF.Exp}(x\text{ES}, l_5 | H_3)$

$m\text{SS} = \text{HKDF.Exp}(x\text{ES}, l_6 | H_3)$

$\text{MS} = \text{HKDF.Ext}(m\text{ES}, m\text{SS})$

$H_5 = H(N_C \ldots \text{CFin})$

$\text{tk}_{app} = \text{HKDF.Exp}(\text{MS}, l_7 | H_5)$

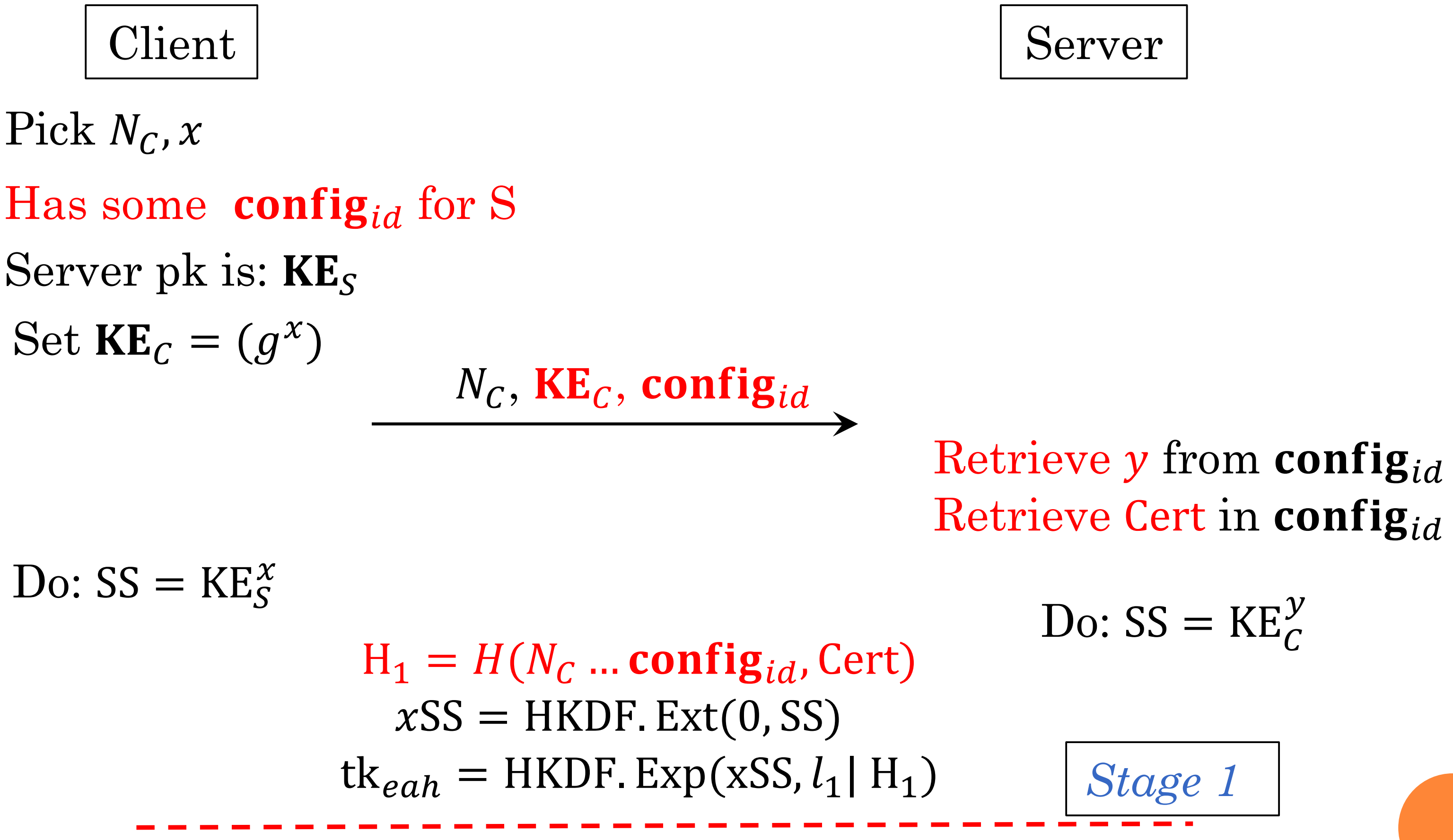
Stage 2

VERY FAST HANDSHAKE – 0-RTT

- Zero Roundtrip time – 0-RTT mode
  - Designed so client can encrypt from the first message
  - A main characteristic of modern AKE schemes
  - Requires knowledge of some public or private value corresponding to a server

- In TLS, 4-stage protocol turns into 6-stage one
  - Use pre-shared key to compute early data key
  - Use that key to execute the remainder of handshake
  - Generate keys as before, including EMS, RMS

STAGE 1 IN 0-RTT



Client

Server

Pick N_C, x

Has some $\mathbf{config}_{id}$ for S

Server pk is: $\mathbf{KE}_S$

Set $\mathbf{KE}_C = (g^x)$

$$\xrightarrow{N_C,\ \mathbf{KE}_C,\ \mathbf{config}_{id}}$$

Retrieve y from $\mathbf{config}_{id}$

Retrieve Cert in $\mathbf{config}_{id}$

Do: $\mathrm{SS} = \mathrm{KE}_S^x$

Do: $\mathrm{SS} = \mathrm{KE}_C^y$

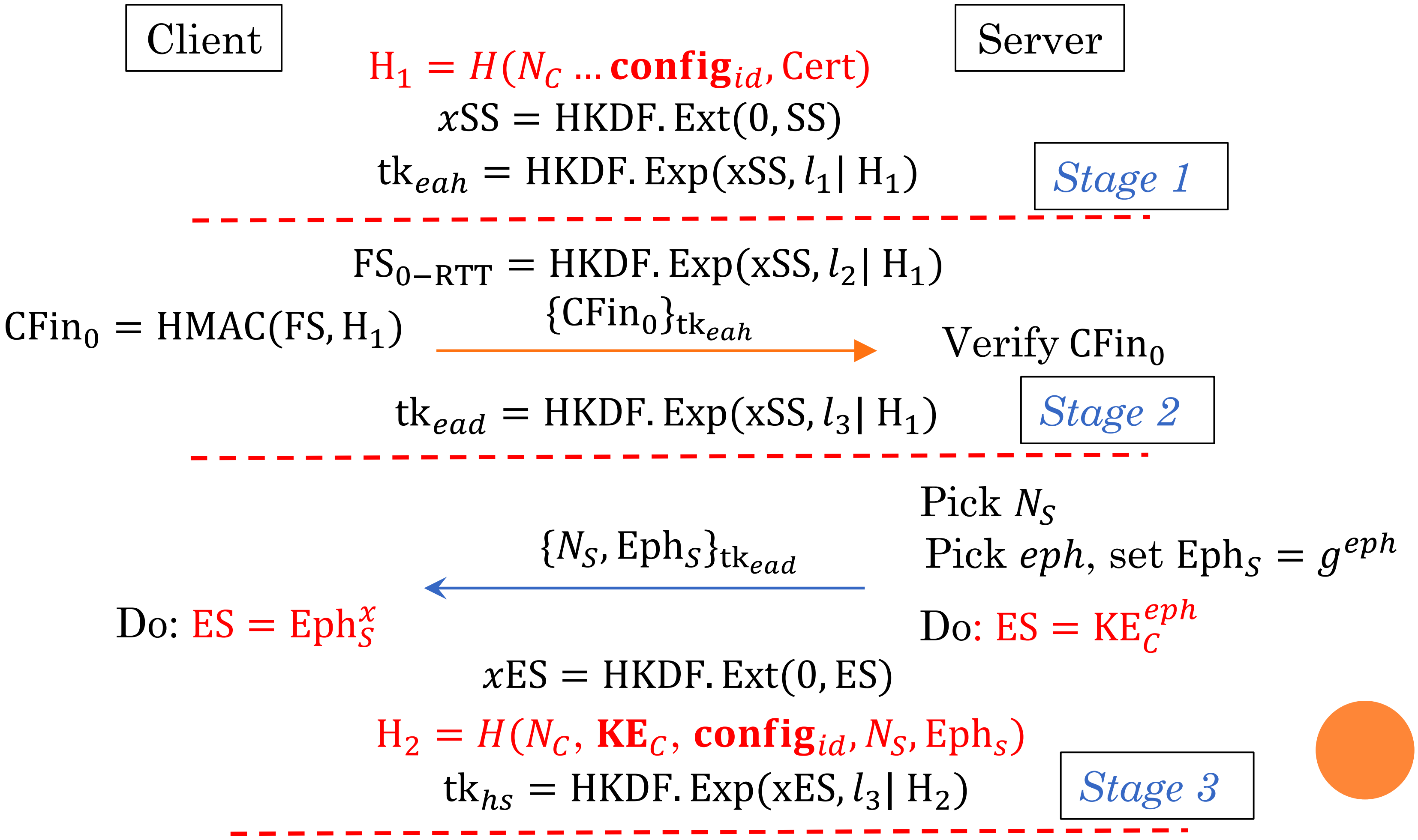
$\mathrm{H}_1 = H(N_C \ldots \mathbf{config}_{id}, \mathrm{Cert})$

$x\mathrm{SS} = \mathrm{HKDF.Ext}(0, \mathrm{SS})$

$\mathrm{tk}_{eah} = \mathrm{HKDF.Exp}(\mathrm{xSS}, l_1 | \mathrm{H}_1)$

Stage 1

STAGE 2 IN 0-RTT MODE



Client

Server

$\mathrm{H}_1 = H(N_C \ldots \mathbf{config}_{id}, \mathrm{Cert})$

$x\mathrm{SS} = \mathrm{HKDF.Ext}(0, \mathrm{SS})$

$\mathrm{tk}_{eah} = \mathrm{HKDF.Exp}(\mathrm{xSS}, l_1 | \mathrm{H}_1)$

Stage 1

$\mathrm{FS}_{0-\mathrm{RTT}} = \mathrm{HKDF.Exp}(\mathrm{xSS}, l_2 | \mathrm{H}_1)$

$\mathrm{CFin}_0 = \mathrm{HMAC}(\mathrm{FS}, \mathrm{H}_1)$

$\xrightarrow{\{\mathrm{CFin}_0\}_{\mathrm{tk}_{eah}}}$

Verify CFin_0

$\mathrm{tk}_{ead} = \mathrm{HKDF.Exp}(\mathrm{xSS}, l_3 | \mathrm{H}_1)$

Stage 2

Pick N_S

Pick eph, set $\mathrm{Eph}_S = g^{eph}$

$\xleftarrow{\{N_S, \mathrm{Eph}_S\}_{\mathrm{tk}_{ead}}}$

Do: $\mathrm{ES} = \mathrm{Eph}_S^x$

Do: $\mathrm{ES} = \mathrm{KE}_C^{eph}$

$x\mathrm{ES} = \mathrm{HKDF.Ext}(0, \mathrm{ES})$

$\mathrm{H}_2 = H(N_C, \mathbf{KE}_C, \mathbf{config}_{id}, N_S, \mathrm{Eph}_s)$

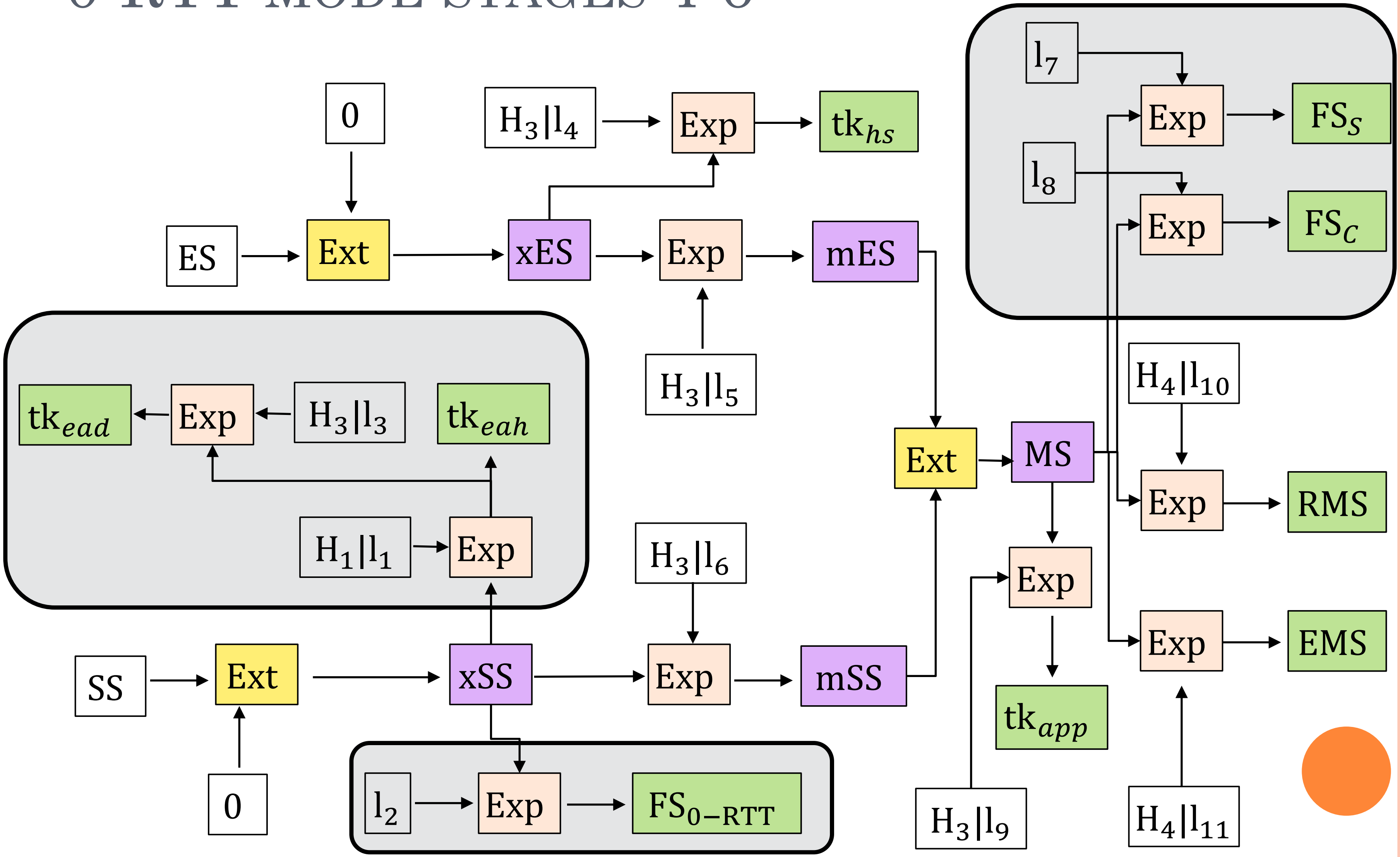
$\mathrm{tk}_{hs} = \mathrm{HKDF.Exp}(\mathrm{xES}, l_3 | \mathrm{H}_2)$

Stage 3

STAGES 4-6

- Stage 3 ends like stage 1 of full handshake
- Some differences:
  - One intermediate & one long-term client Finished
  - Finished keys for server & client are different
  - Some keys take as input just labels, not hashes
  - Master secret yields five different keys
  - A much more complicated key-scheduling mechanism

0-RTT MODE STAGES 4-6



0

$H_3||l_4$

Exp

tk_{hs}

l_7

Exp

FS_S

l_8

Exp

FS_C

ES

Ext

xES

Exp

mES

tk_{ead}

Exp

$H_3||l_3$

tk_{eah}

$H_1||l_1$

Exp

$H_3||l_5$

Ext

MS

$H_4||l_{10}$

Exp

RMS

Exp

$H_3||l_6$

SS

Ext

xSS

Exp

mSS

Exp

EMS

tk_{app}

0

l_2

Exp

FS_{0-RTT}

$H_3||l_9$

$H_4||l_{11}$

PART 6
SAFELY EXPORTING KEYS

EXPORT KEYS IN AKE

- Authenticated Key-Exchange:
  - Allow two parties to establish a secure channel
  - Output: a set of channel keys, to use for AE
  - Can sometimes also provide export keys

- "Good" export keys:
  - Indistinguishable from random
  - Do not reveal anything about secret channel keys
  - The channel keys do not reveal anything about the export keys
  - In short: it is best to have independent export keys

“TLS-LIKE” PROTOCOLS [BJS16]

- Recall ACCE security:
  - Mutual authentication (otherwise SACCE)
  - Channel security

- TLS-like protocols:
  - ACCE-secure authenticated key-exchange
  - Both parties generate randomness at every session
  - During the protocol, both parties compute MS
  - Keys computed as $K \coloneqq \mathrm{KDF}(\mathrm{MS}, \mathrm{nonces}, F(\mathrm{T}))$
    - T is protocol transcript, F is publicly computable

TLS 1.2 GOOD EXPORT KEYS

- Given a TLS-like protocol (e.g. TLS 1.2)
  - Nonces: N_C, N_S
  - Master secret msk
  - Keys derived as: $HMAC(msk; N_C|N_S)$

- Consider the following export keys:
  - $\text{EK} \coloneqq PRF(\text{msk}; N_C|N_S, \text{aux})$ s.t. $\text{EK} \neq Keys$

- Then these keys are good export keys
  - The main reason is: MS remains hidden at all times

EXPORT KEYS FOR TLS 1.3

- Exercise 1:
  - Is TLS 1.3 "TLS-like" [BJS16]?

- Exercise 2:
  - Assume TLS 1.3 is secure (proofs by DFG+15, FG16), which means tk_{app} is indistinguishable from random
  - What does this mean for the master secret ?
  - What can you say about EMS, RMS