TLS 1.3

Часть 2

Handshake (напоминание)

Client

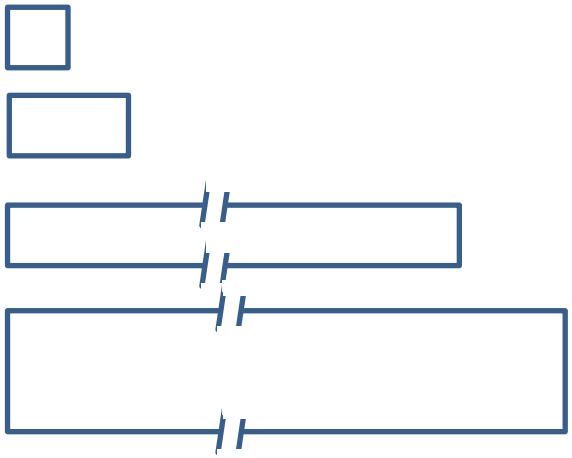
ClientHello □

Server

□ ServerHello
□ EncryptedExtensions
□ CertificateRequest
□ Certificate
□ CertificateVerify
□ Finshed

Certificate □
CertificateVerify □
Finished □
Application Data □□ Application Data

CertificateRequest



T = 13

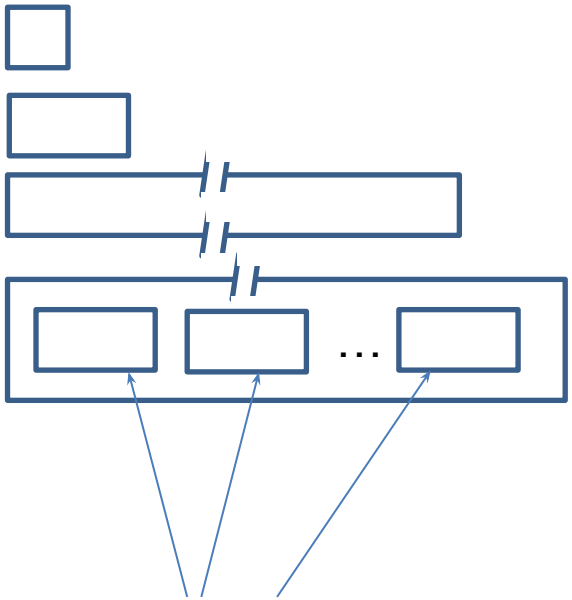
L

ID запроса (может быть пустым)

extensions:

- signature_algorithms
- signature_algorithms_cert
- certificate_authorities

Certificate



T = 11

L

ID запроса

certificate_list

- массив сертификатов
- должен составлять сертификационный путь
- первым должен быть сертификат проверяемой стороны
- получатель должен уметь восстановить нарушение порядка

Certificate

Формат «сертификата» в сообщении Certificate:

- Сертификат X.509
- extensions

Или

- «Сокращенный» сертификат (subjectPublicKeyInfo)
- extensions

extensions:

ocsp_status
signed_certificate_timestamp

Certificate

«Сокращенный» сертификат:

- Использование согласовывается при помощи расширений client_certificate_type и server_certificate_type
- Только один сертификат в массиве
- Подтверждение подлинности средствами вне PKIX
- Стандарты:
  - Сокращенный сертификат – RFC 7250 (2014)
  - DANE – RFC 6698 (2012)

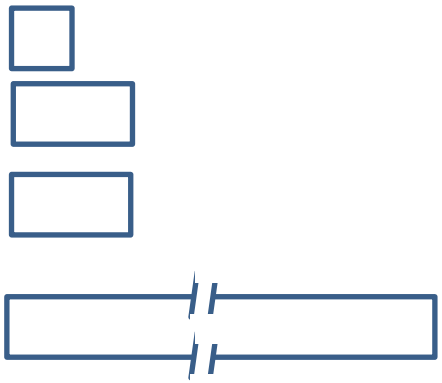
Справка: DANE

DNS-Based Authentication of Named Entities

RFC 6698 (2012)

- Ресурсная запись типа TLSA содержит (либо)
  - Сертификат
  - Открытый ключ (SubjectPublicKeyInfo)
  - Хэш от того или другого
- Подписывается при помощи RRSIG
- Цепь доверия идет не от корневого CA, а от корневой зоны DNS

CertificateVerify



T = 15

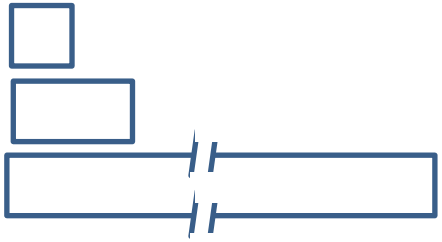
L

algorithm - код алгоритма ЭЦП (как в расширении signature_algotithms)

signature

- Вычисляется подпись значения Hash(), которая (Hash()) берется по всем сообщениям протокола Handshake, предшествующим данному.
- При помощи закрытого ключа отправителя.
- Получатель сообщения проверяет эту подпись открытым ключом отправителя, находящемся в головном сертификате из сообщения Certificate.
- Таким образом, отправитель доказывает получателю, что он действительно владеет закрытым ключом, соответствующим открытому ключу в сертификате.
- При использовании DHE/ECDHE подпись подтверждает подлинность открытого ключа
- ClientHello.random и ServerHello.random защищают от Replay attack

Finished



T = 20

L

verify_data (длина = длина отклика Hash())

verify_data = HMAC(finished_key, Messages)

Здесь:

- Messages – все сообщения Handshake, предшествующие данному
- finished_key – выводится односторонней функцией (на основе HMAC) из текущего ключа {server|client}_handshake_traffic_key
- Получатель сообщения Finished проверяет verify_data
- Таким образом стороны доказывают друг другу, что сеанс установлен успешно, и они договорились об одних и тех же ключах
- В случае PSK нет сообщений CertificateVerify, и Finished – единственная проверка подлинности сторон,
- Стороны доказывают друг другу, что владеют общим секретом PSK
- ClientHello.random и ServerHello.random защищают от Replay attack

Прочие сообщения

- EndOfEarlyData
- KeyUpdate
  - смена текущих ключей

Вычисления ключей

- Для каждой цели в TLS 1.3 используется отдельный ключ, который выводится из основного ключа сеанса (прямо или косвенно) при помощи односторонних функций на основе HMAC. После этого предшествующие ключи (включая основной) удаляют.

Вычисления ключей - функции

HKDF-Extract

PRK = HKDF-Extract(salt, IKM) =
    HMAC-Hash(salt, IKM)
salt – случайная, длины Hash.Length, несекретная
IKM – произв. длины, секретная
PRK – фиксированной длины

RFC 5869 (2010)

HKDF – Hash-based Key Derivation Function
PRK – pseudo-random key
IKM – input key materials

Вычисления ключей - функции

HKDF-Expand

OKM = HKDF-Expand(PRK, info, L)

PRK – фикс. длины
info – произв. длины, контекстная информация приложения

L – требуемая длина

OKM – output key materials, требуемой длины L

Вычисления ключей - функции

HKDF-Expand

T(0) = пустая строка

T(1) = HMAC-Hash(PRK, T(0) | info | 0x01)

T(2) = HMAC-Hash(PRK, T(1) | info | 0x02)

T(3) = HMAC-Hash(PRK, T(2) | info | 0x03)

…

Далее берутся первые L байт из
T(1) | T(2) | T(3) | …

RFC 5869 (2010)

Вычисления ключей - функции

HKDF-Expand-Label

HKDF-Expand-Label(Secret, Label, Context, Length) =

HKDF-Expand(Secret, _info_, Length)

Где _info_ = Length | “tls13” | Label | Context

Вычисления ключей - функции

Derive-Secret

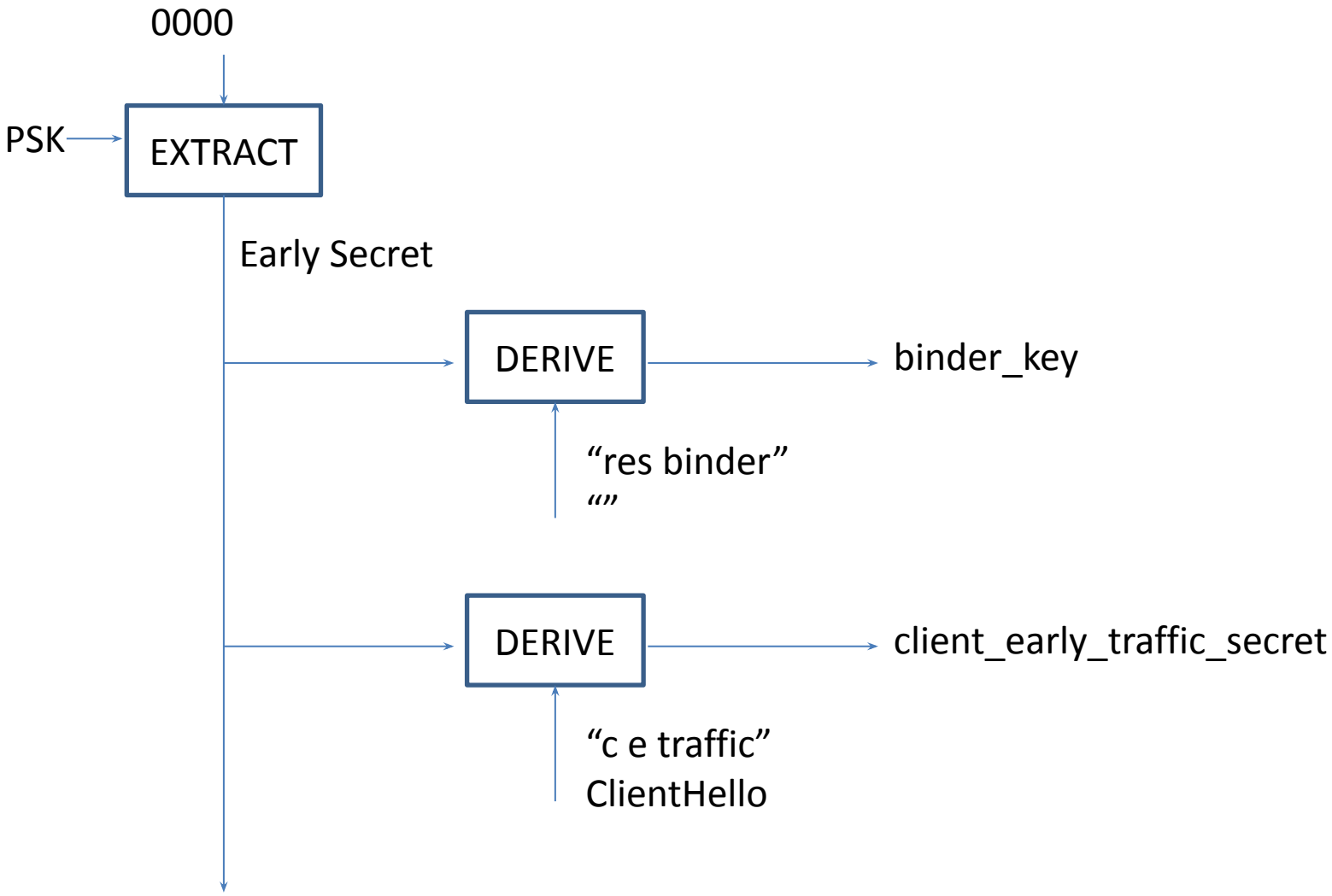
Secret’ =

Derive-Secret(Secret, Label, Messages) =

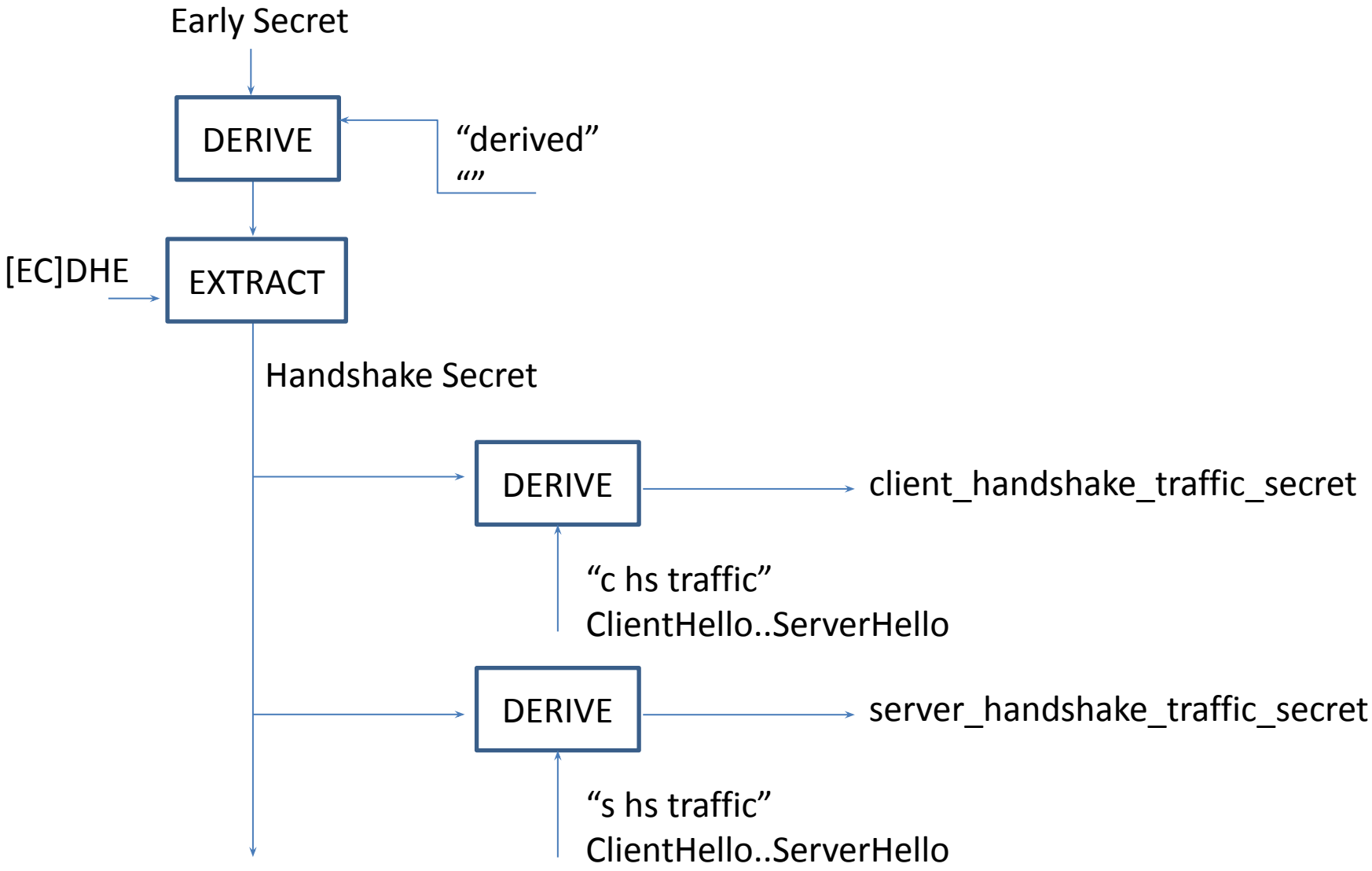
HKDF-Expand-Label(Secret, Label, Hash(Messages), Hash.length)

Здесь Messages – сообщения Handshake от начала до определенного сообщения (текущего).

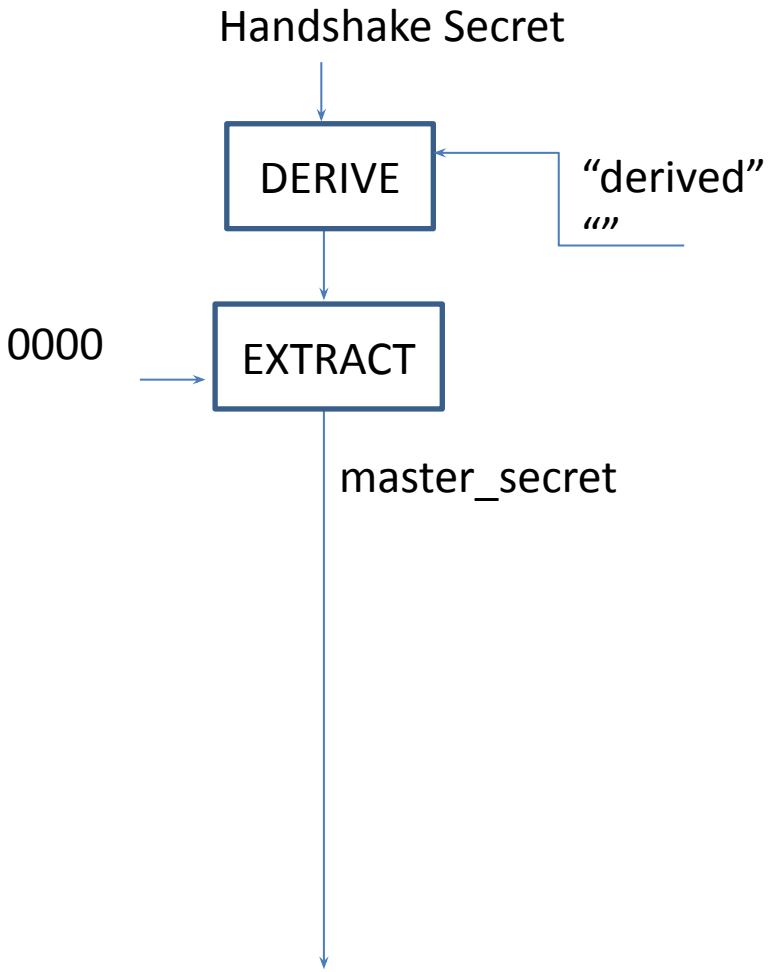
Вычисление ключей - схема



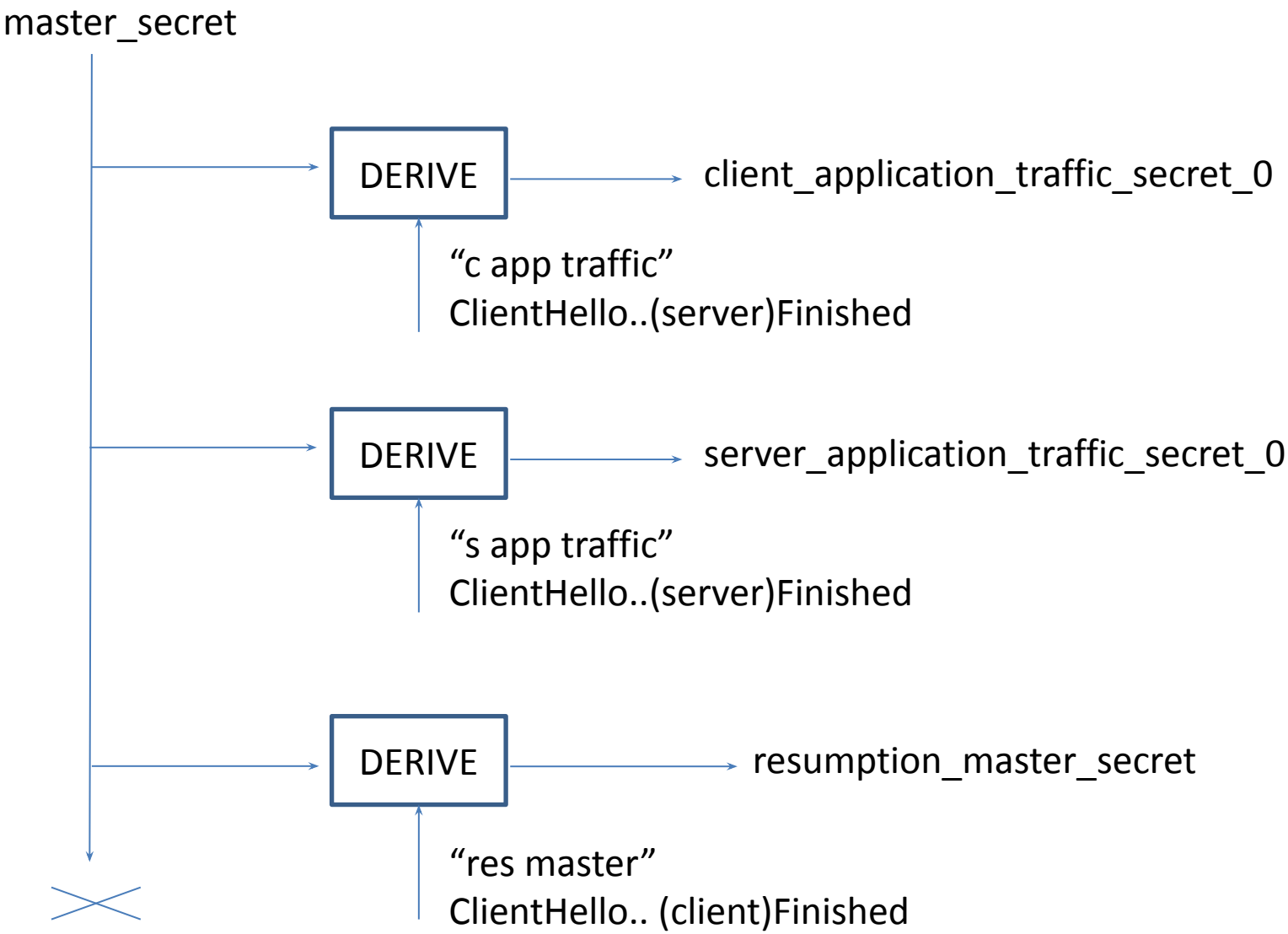
Вычисление ключей - схема



Вычисление ключей - схема



Вычисление ключей - схема



Вычисление ключей - KeyUpdate

{client|server}_application_traffic_secret_{n+1}
=

HKDF-Expand-Label(

{client|server}_application_traffic_secret_{n},

“traffic upd”, Hash.Length)

Вычисление ключей – finished_key

server_finished_key =
HKDF-Expand-Label(
    server_handshake_traffic_secret,
    “finished”, “”, Hash.length)

client_finished_key =
HKDF-Expand-Label(
    client_handshake_traffic_secret,
    “finished”, “”, Hash.length)

Вычисление ключей – рабочие ключи

${sender}_${phase}_write_key =
HKDF-Expand-Label(
    ${sender}_${phase}_traffic_secret,
    “key”, “”, Key.length)

${sender}_${phase}_write_IV =
HKDF-Expand-Label(
    ${sender}_${phase}_traffic_secret,
    “iv”, “”, IV.length)

${sender} = client | server
${phase} = handshake | application_{n} | early (только client)