

1 TLS 1.0 - 1.2

TLS Client (pk)

pick n_I

CH: $n_I, max_I, [a_1, \dots, a_n]$

check cert

SH: $n_R, v, q_R, hash_1$

SC: Cert

SKE: $p, g, g^y, sign(sk, hash_1, (n_I/n_R | p | g | g^y))$

verify signature
pick z

CKE: g^z

$(ms, k_1, k_2) \leftarrow kdf(g^{zy}, n_I/n_R)$

CFIN: $Enc(k_1, mac(ms, hash_2(log_1)))$

check mac

check mac

pick n_R, y, p, g

TLS Server (sk, cert)

2 TLS 1.3

TLS Client (pk)

pick n_I, z_i which server chose

CH: $n_I, max_I, [a_1, \dots, a_n], \{g, p, g^y\}$

set z to z_i

SH: $n_R, v, q_R, hash_1, g, p, g^y$

SC: $Enc(k_2, cert)$

$(k_1, k_2) \leftarrow kdf(g^{zy}, log_1)$

verify cert

SCV: $Enc(k_2, sign(sk, hash_1, (hash(log_2))))$

$ms \leftarrow kdf(g^{zy}, log_3)$

SFIN: $Enc(k_2, mac(ms, hash(log_3)))$

check mac

CFIN: $Enc(k_1, mac(ms, hash(log_4)))$

check mac

pick i, n_R, y

TLS Server (sk, cert)

Legend:

n_I/n_R - Initiator (Client) / Receiver (Server) nonce.

max_I - maximal TLS version supported by client.

v - version number of protocol.

$pk/sk/cert$ - public/private signing keys, and certificate of server.

$hash_1$ - hash function chosen by server.

q_R/a_i - a tuple (mac, hash₂, kdf, enc, sign) defining algorithms that will be used/can be used.

p/g - modulus/generator of DH group that will be used.

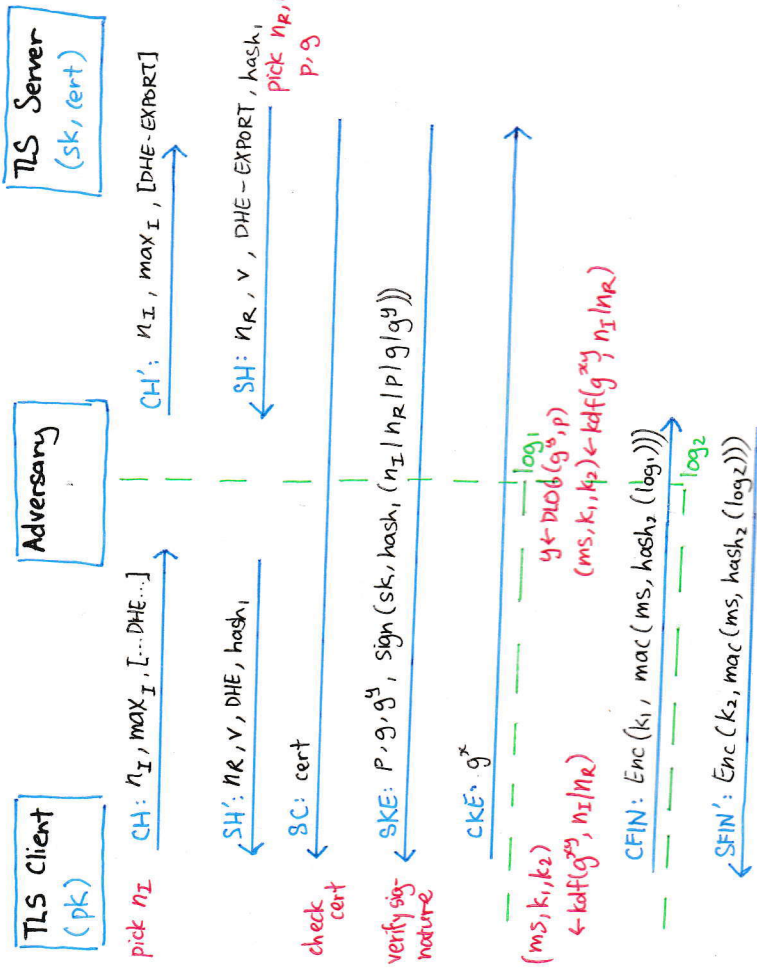
"nonce hack"

pick n_R , then set $n_R \leftarrow v/n_R$.

Note:

The client follows protocol 1 if $v=1.2$, and 2 if $v=1.3$

③ LOGJAM (in TLS 1.2)



④ LOGJAM (in TLS 1.3)

