Transport Security
Part II

Lecture 13, Computer Networks (198:552)

Fall 2019

RUTGERS
UNIVERSITY | NEW BRUNSWICK

Review: Some key security properties

confidentiality: only sender, intended receiver should "understand" message contents

- sender encrypts message
- receiver decrypts message

integrity: sender, receiver want to ensure message not altered (in transit, or afterwards) without detection

authentication: sender, receiver want to confirm identity of each other

non-repudiation: Once someone sends a message, or conducts a transaction, she can't later deny the contents of that message

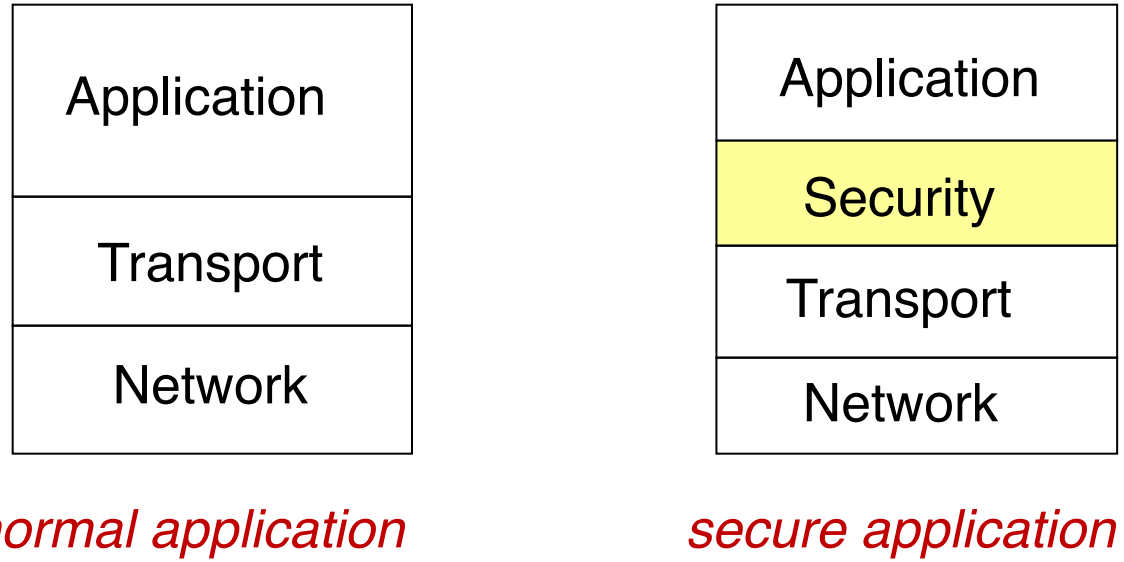
Review: Mechanisms to get properties

- Cryptography: provide confidentiality
  - Symmetric key crypto: $m = dec_{Ks}(enc_{Ks}(m))$
  - public key crypto: $m = dec_{K}^{-}(enc_{K}^{+}(m))$
  - Key exchange problem vs. efficient computations
  - Diffie-Hellman-Merkle key exchange
- Integrity: Message digest with shared secret
- Non-repudiation: digital signature
  - Use public key cryptography over message digest
- Authentication:
  - Use public key cryptography and nonce
  - Certifying authority binds public key with an entity by acting as a trusted third party

How should applications use the security mechanisms described thus far?

Modularity. Implement once, implement well.
Reuse across many applications.

How should apps use security?



normal application *secure application*

- Would be nice to provide an API to applications
- Useful to have security layer *above* transport

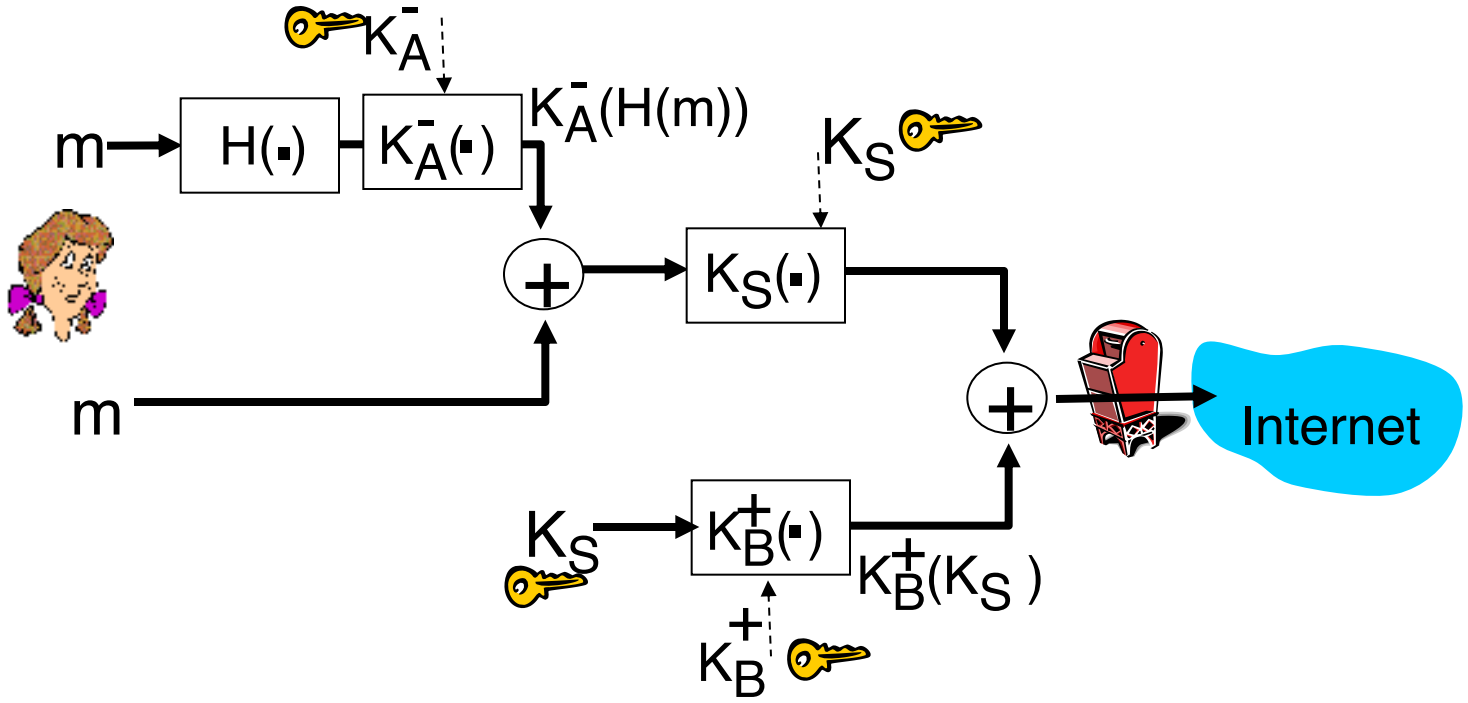
Transport Layer Security (TLS)

Providing security properties in a practical protocol

Goals of TLS

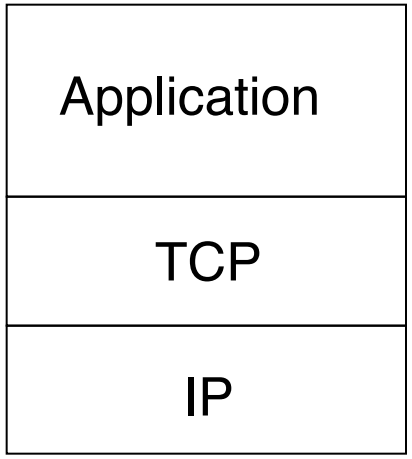
- Confidentiality
- Message integrity
- Server authentication (optionally, client authentication)
- Work in the context of the existing network protocol stack

Could do something like this… (PGP)

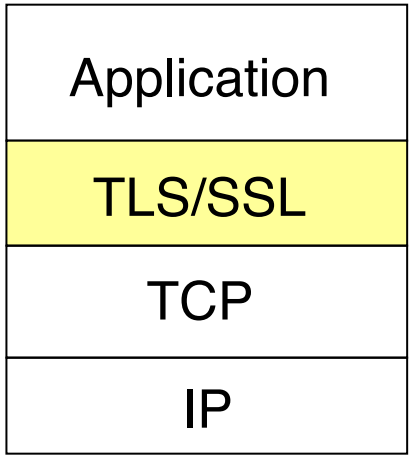


- but want to send byte streams & interactive data
- want set of secret keys for entire connection
- want certificate exchange as part of protocol: handshake phase

TLS/SSL and the rest of the protocol stack



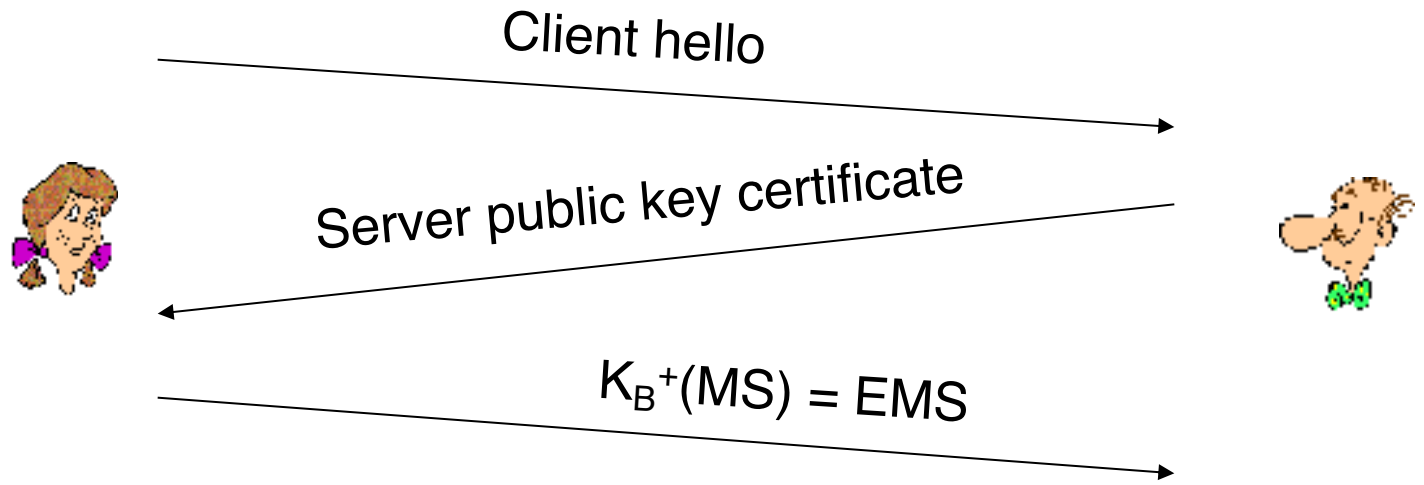
normal application



application with TLS/SSL

- TLS/SSL provides application programming interface (API) to applications
- C and Java libraries/classes readily available
  - Ex: OpenSSL

Step (1): a simple handshake



MS: master secret

EMS: encrypted master secret

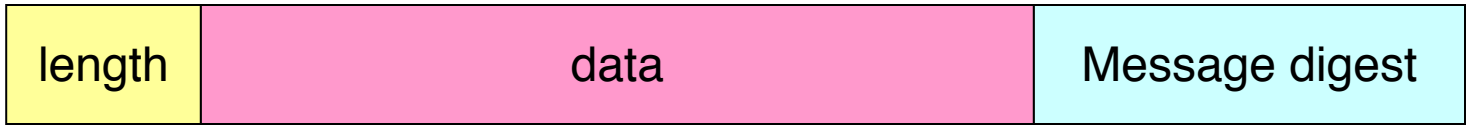
Q: What all might the “master secret” be used for?

Step (2): key derivation

- considered bad to use same key for more than one cryptographic operation
  - use different keys for message integrity and encryption
- four keys:
  - K_c = encryption key for data sent from client to server
  - M_c = integrity digest key for data sent from client to server
  - K_s = encryption key for data sent from server to client
  - M_s = integrity digest key for data sent from server to client
- keys derived from key derivation function (KDF)
  - Takes master secret and (possibly) some additional random data and creates the keys

Step (3): Data records

- why not encrypt data in constant stream as we write it to TCP?
  - where would we put the message digest? If at end, no message integrity until all data processed.
  - e.g., with instant messaging, how can we do integrity check over all bytes sent before displaying?
- instead, break stream in series of records
  - each record carries a message digest
  - receiver can act on each record as it arrives
- How does receiver distinguish the digest from data within a record?
  - want to use variable-length records



Goals of TLS

- Confidentiality
- Message integrity
- Server authentication (optionally, client authentication)
- Work in the context of the existing network protocol stack
- Evolve as new security standards are put in place
  - TLS implements cipher negotiation

TLS/SSL “Cipher Suite”

- Cipher suite
  - public-key algorithm
  - symmetric encryption algorithm
  - Integrity hashing algorithm
- TLS/SSL supports several cipher suites
- Negotiation: client, server agree on cipher suite
  - client offers choices
  - server picks one

Common symmetric ciphers

- AES – Advanced Encryption Standard
- DES – Data Encryption Standard: block
- 3DES – Triple strength: block
- ChaCha: stream
- RC4 – Rivest Cipher 4: stream

SSL Public key encryption

- RSA with DH

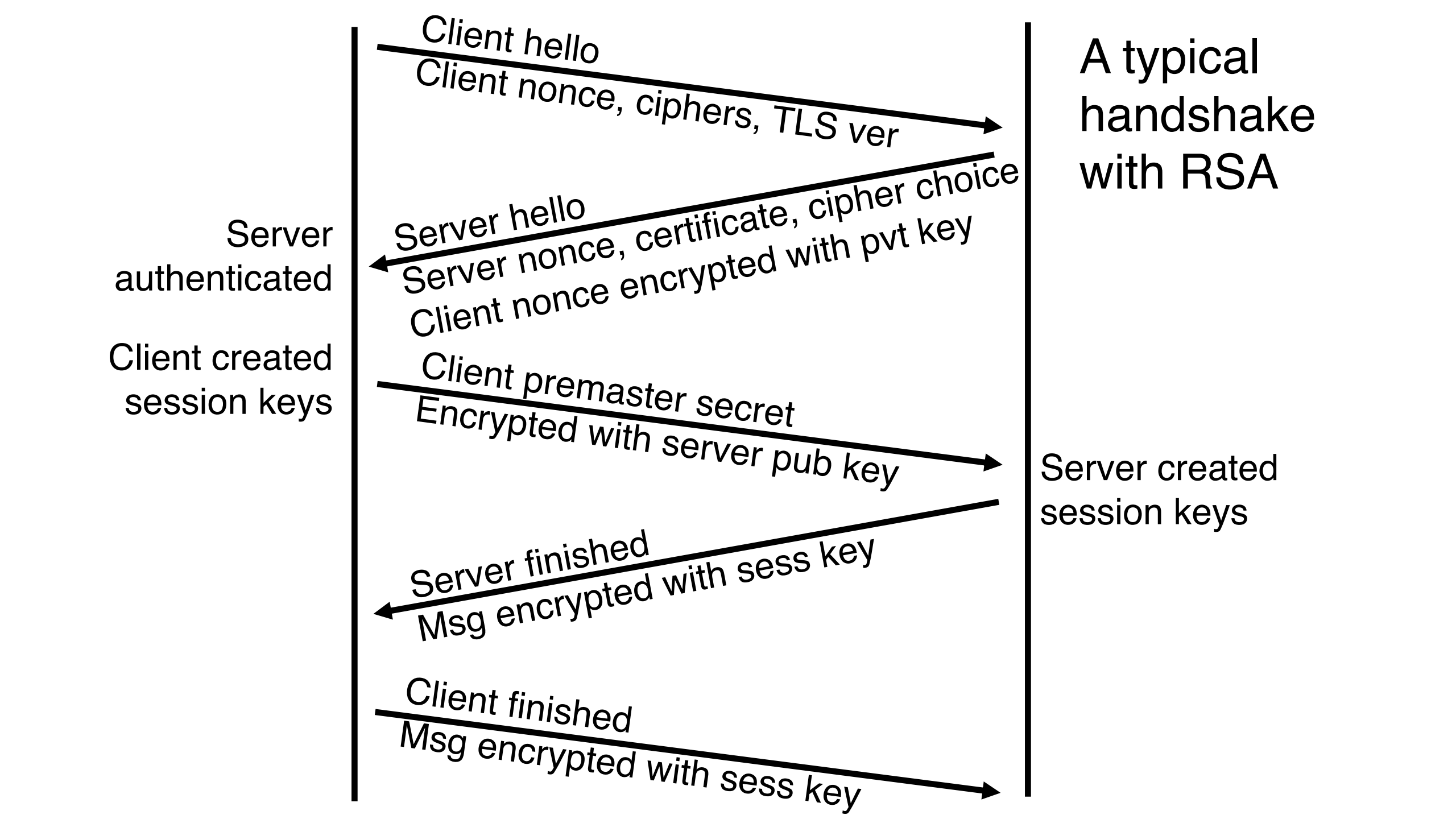
Integrity hashing algorithms

- HMAC-MD5 and others

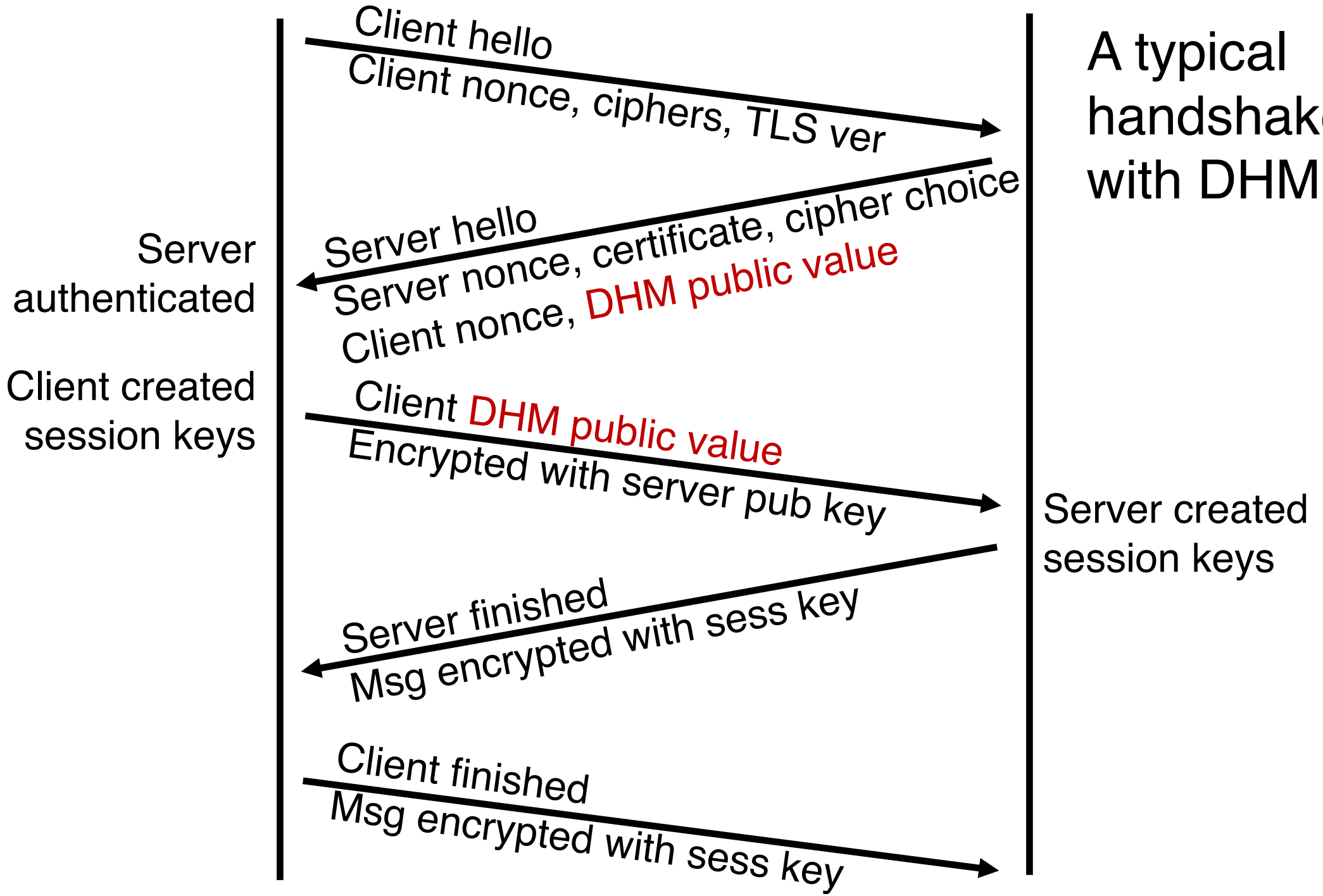
Handshake with Negotiation

1. server authentication
2. negotiation: agree on cipher suite
3. establish necessary keys
4. client authentication (optional)

TLS 1.2: All this takes a few round trip times to accomplish!


A typical handshake with RSA
Client hello
Client nonce, ciphers, TLS ver
Server hello
Server nonce, certificate, cipher choice
Client nonce encrypted with pvt key
Server authenticated
Client created session keys
Client premaster secret
Encrypted with server pub key
Server created session keys
Server finished
Msg encrypted with sess key
Client finished
Msg encrypted with sess key

A typical handshake with DHM



QUIC handshake

Adding security in 1 (or 0) RTT to your transport connection

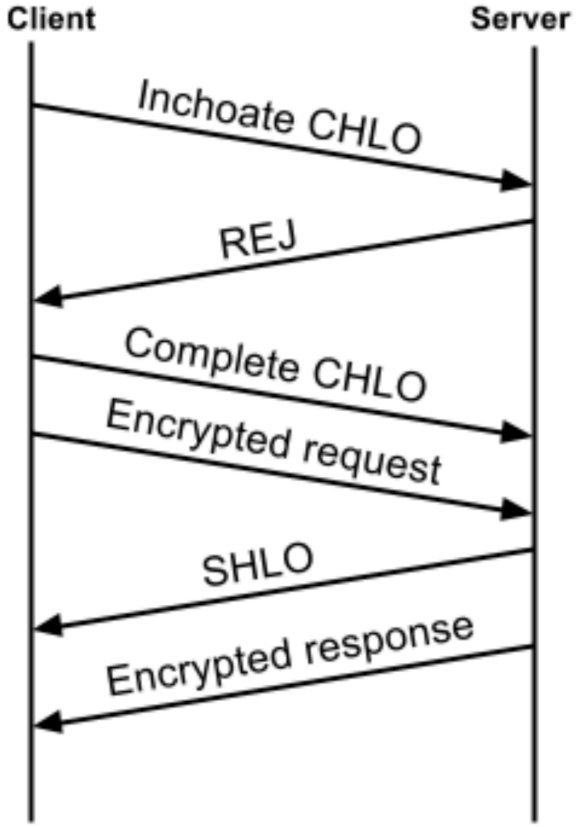
Optimize the common case

Goals of QUIC handshake

- Reduce round-trip times to get set up with secure connection
- Initial handshake takes 1 RTT, starting from the first byte received from the client, before server can send data
- Later, server can send data immediately upon receiving the first byte from the client (“0-RTT” handshake)

Initial 1-RTT handshake

- Client first sends an “inchoate” (incomplete) client hello message



Initial 1-RTT Handshake

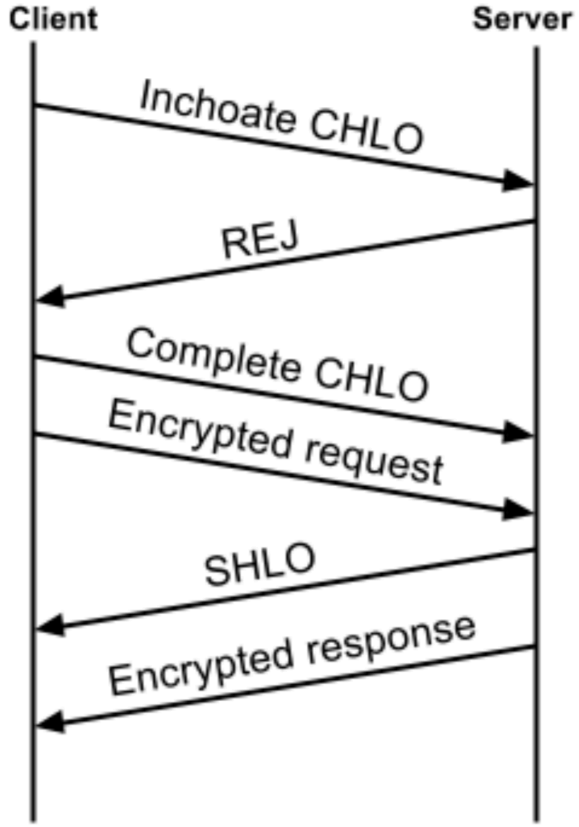
Server REJ message contains:

(1) server’s long-term DHM public value
(2) Certificate chain authenticating the server
(3) Signature of DHM public value using the certified (private) key
(4) Client source address token signed by server

What properties does each of these (help) establish?

Initial 1-RTT handshake

- Client first sends an “inchoate” (incomplete) client hello message



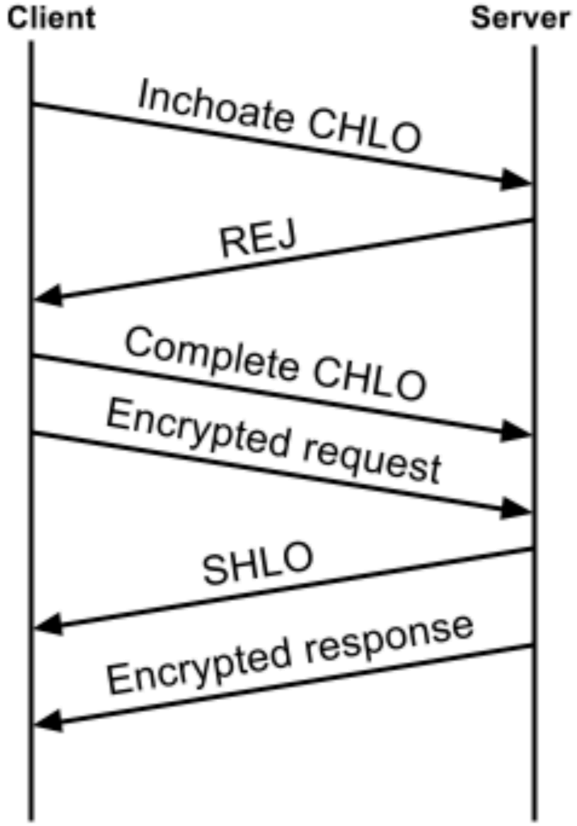
Initial 1-RTT Handshake

Server REJ message contains:
(1) server’s long-term DHM public value
(2) Certificate chain authenticating the server
(3) Signature of DHM public value using the certified (private) key
(4) Client source address token signed by server: session ticket

Useful later on to authenticate the client
Prevent DoS (of server and other clients)

Initial 1–RTT handshake

- Finish the handshake



Initial 1-RTT Handshake

Client sends its ephemeral DHM public value to server in a complete CHLO.

Server then responds with its ephemeral DHM public value.

Ephemeral DH values provide forward secrecy for the communication

Other handshake details

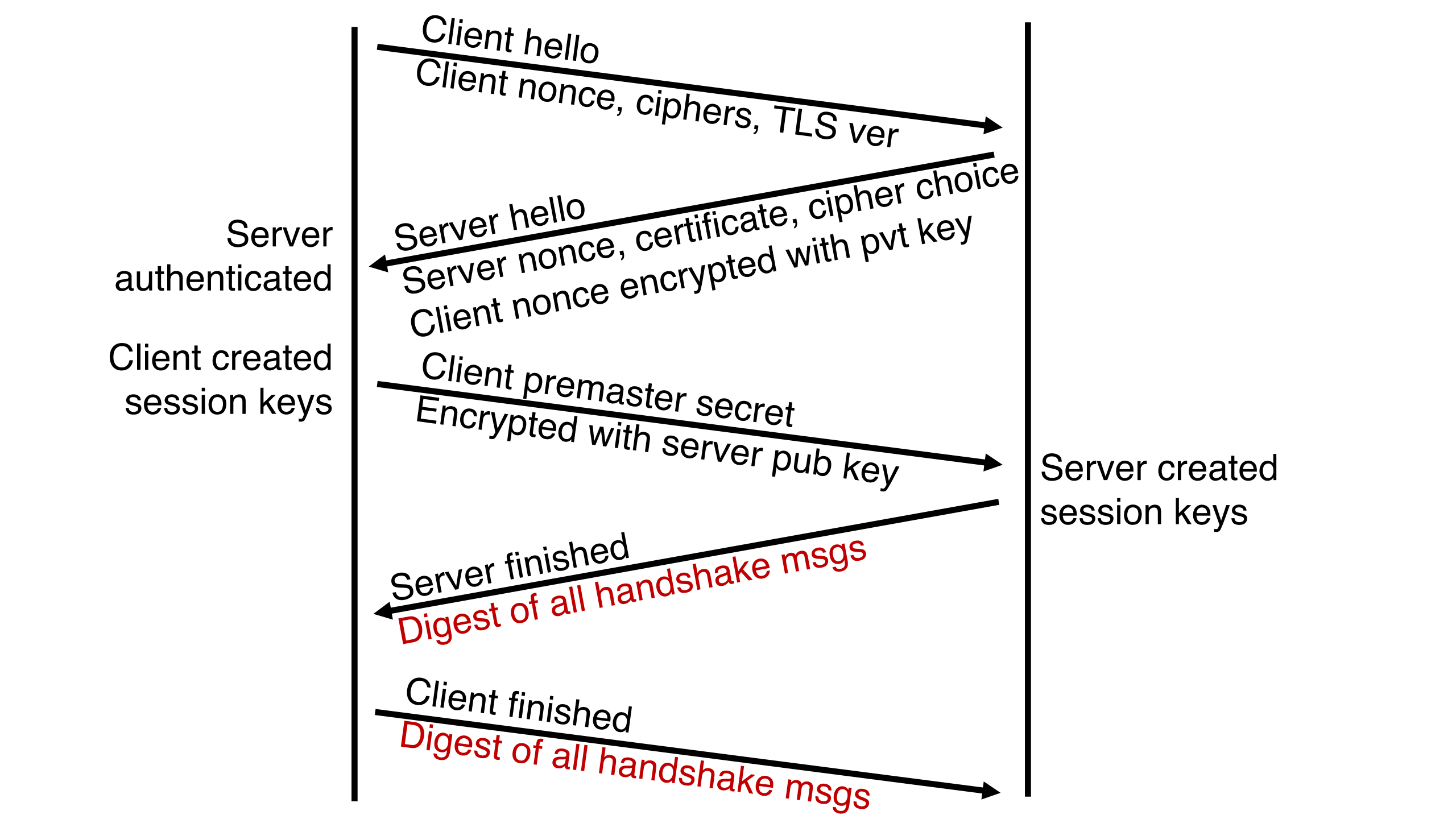
- In later handshakes, client optimistically sends its source address token to the server
- Optimistic version negotiation: client proposes a cipher suite and encrypts first request with that cipher suite
  - If server can’t speak that cipher suite, it forces version renegotiation, similar to TLS
- Both techniques optimize the common case when clients speak to a known server with an agreed-upon cipher suite

Important principle

- Use a hash of all the plaintext data that was exchanged in the handshake when generating the symmetric key for the connection
  - True in both TLS and QUIC

- Why?

- Example: cipher downgrade attacks
  - Provide “backward integrity” over all prior messages


Client hello
Client nonce, ciphers, TLS ver
Server hello
Server nonce, certificate, cipher choice
Client nonce encrypted with pvt key
Server authenticated
Client created session keys
Client premaster secret
Encrypted with server pub key
Server created session keys
Server finished
Digest of all handshake msgs
Client finished
Digest of all handshake msgs