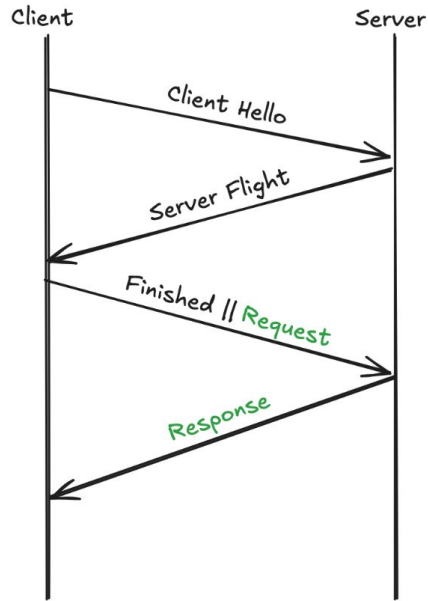


# TLS Preflight

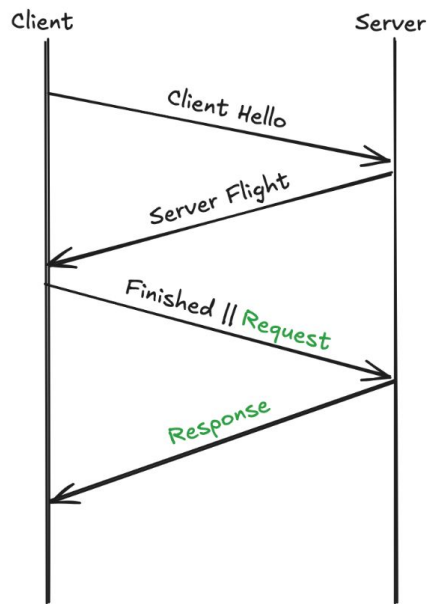
Dennis Jackson  
Mozilla

# Motivation: Faster handshakes

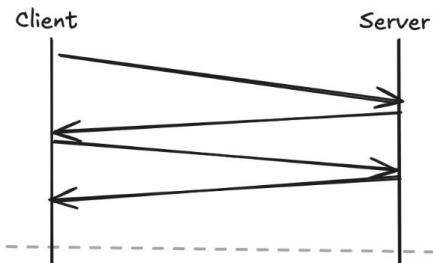


Regular TLS 1.3  
Handshake

# Motivation: Faster handshakes

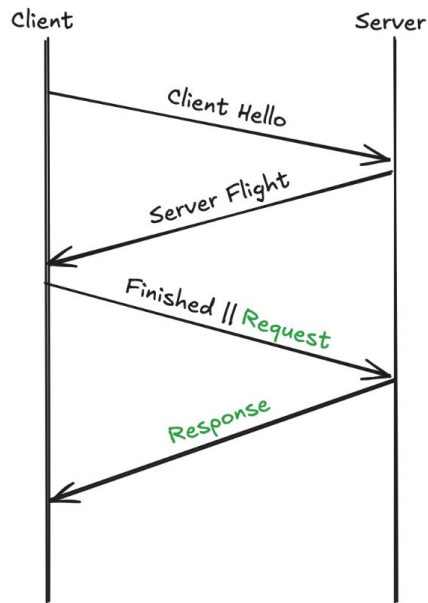


Regular TLS 1.3  
Handshake

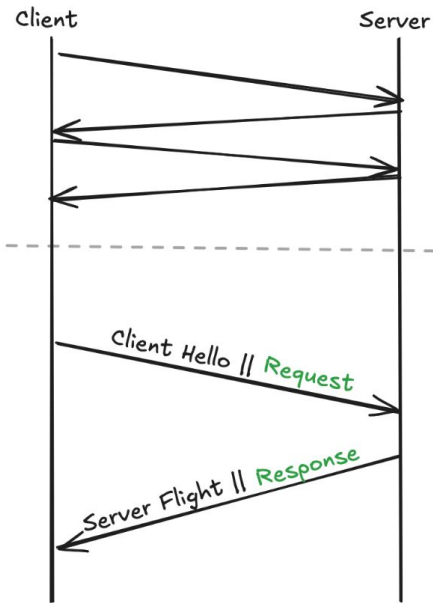


0-RTT on  
Resumption

# Motivation: Faster handshakes

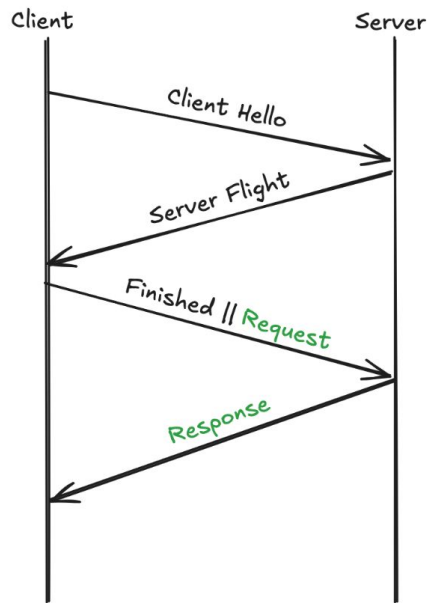


Regular TLS 1.3  
Handshake

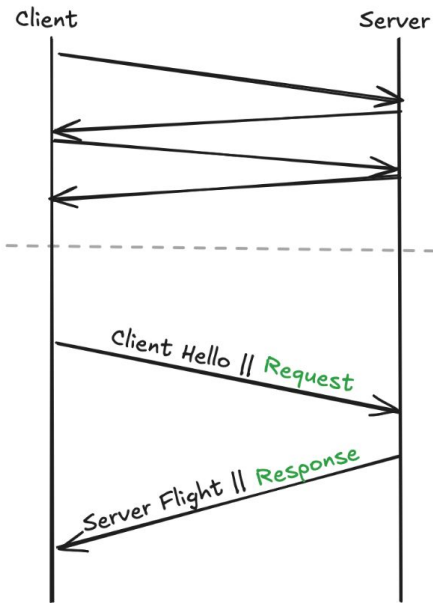


0-RTT on  
Resumption

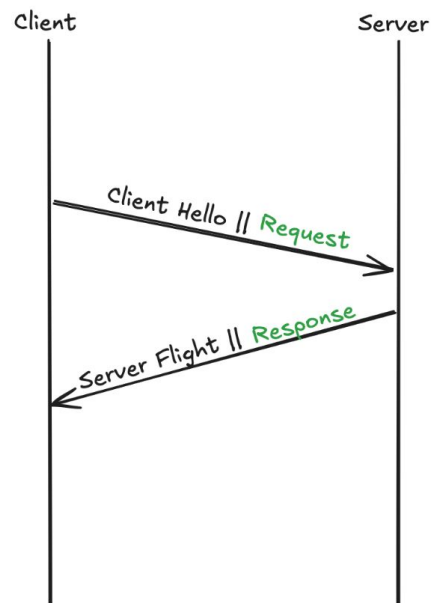
# Motivation: Faster handshakes



Regular TLS 1.3  
Handshake



0-RTT on  
Resumption



Want:  
Always 0-RTT

# TLS Preflight

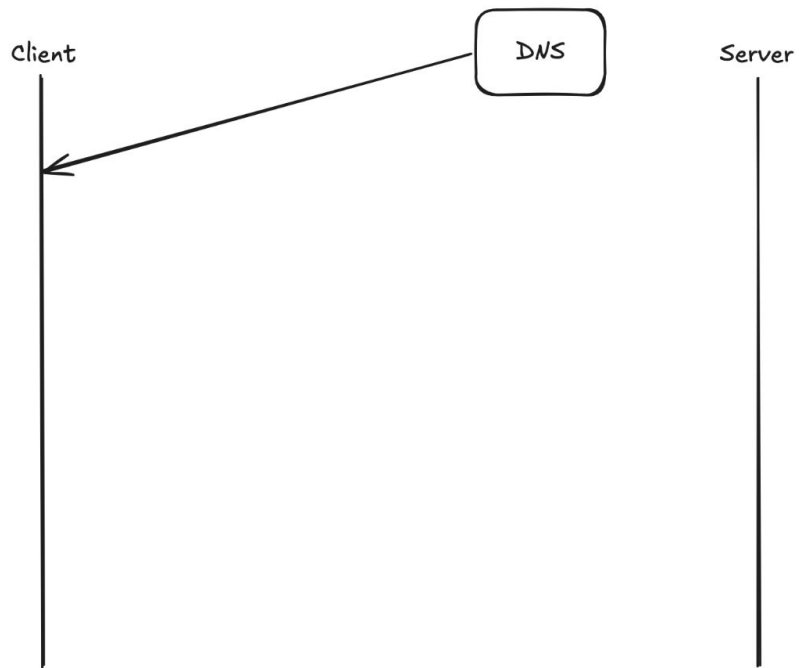
Client



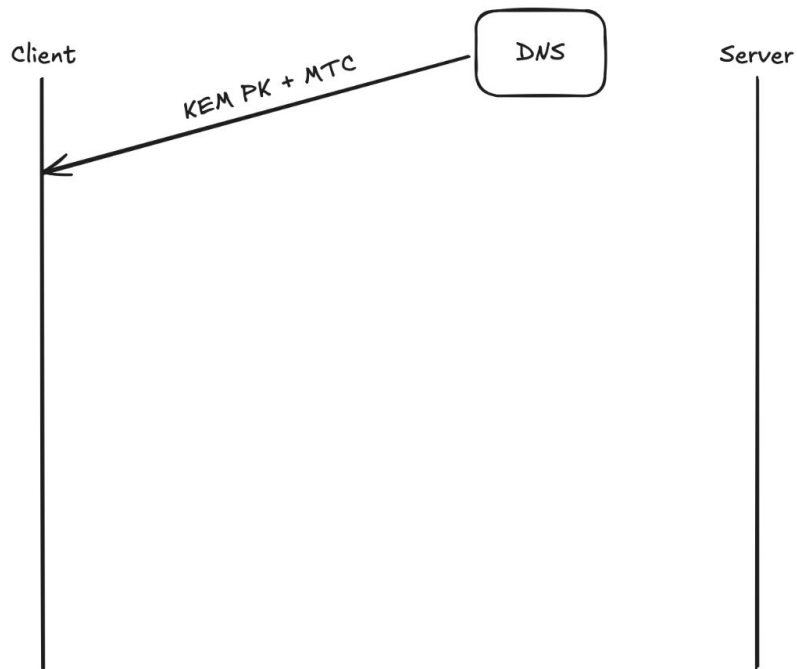
Server



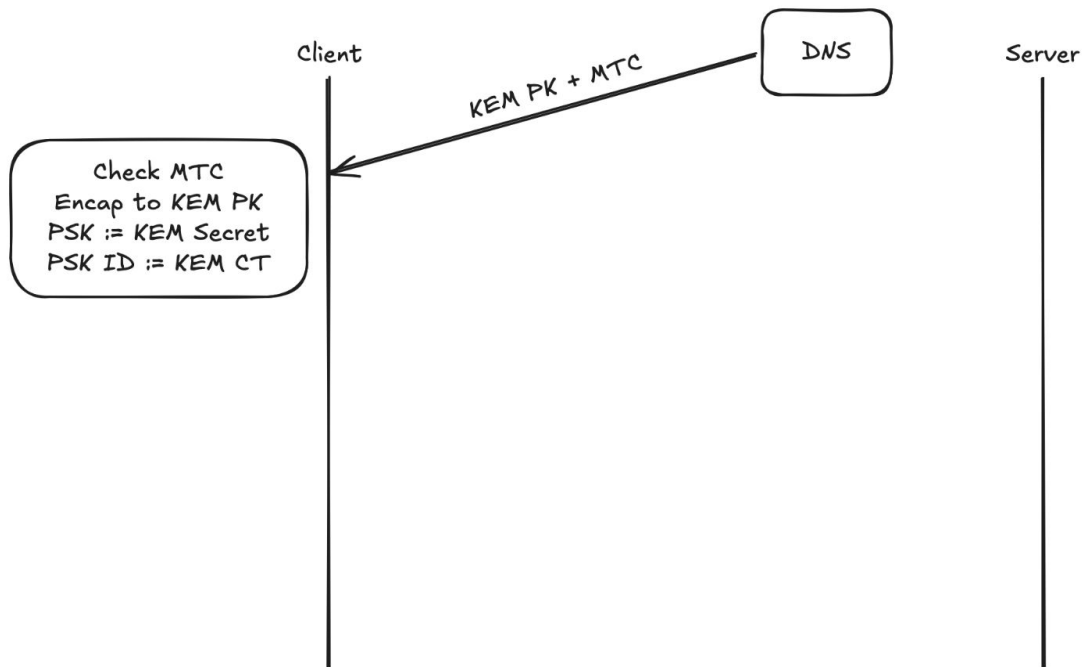
# TLS Preflight



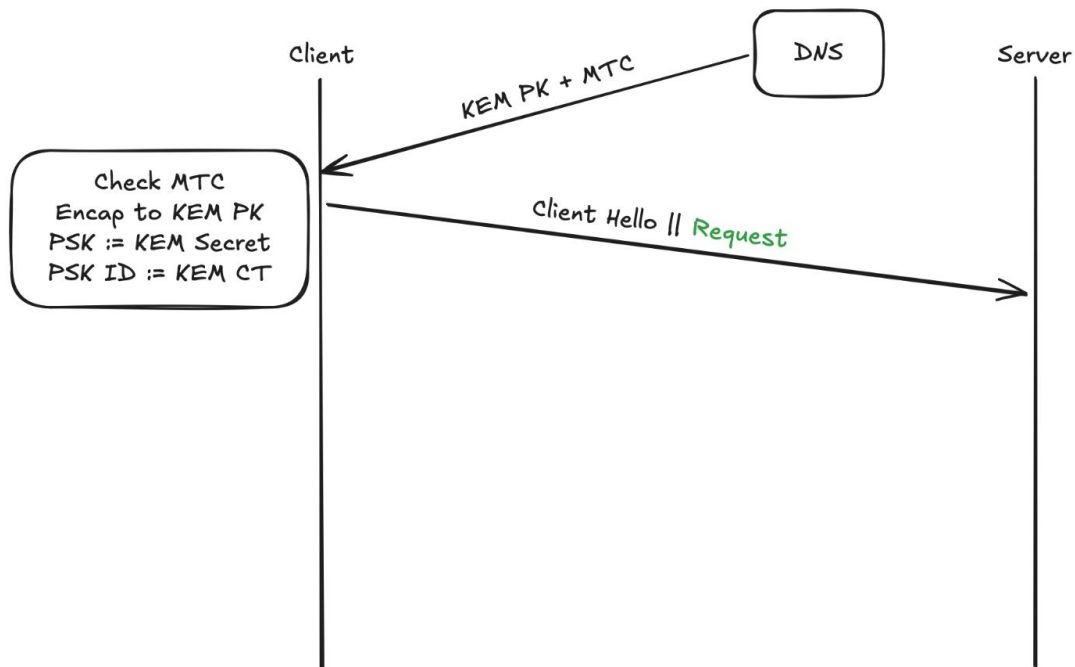
# TLS Preflight



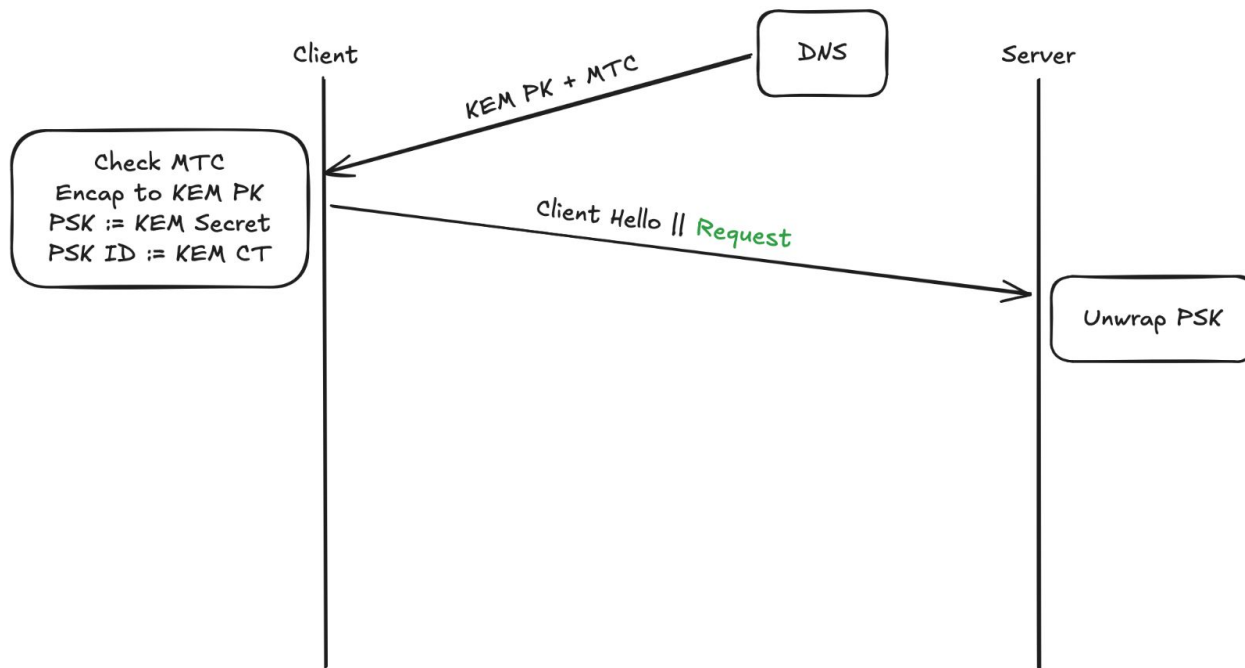
# TLS Preflight



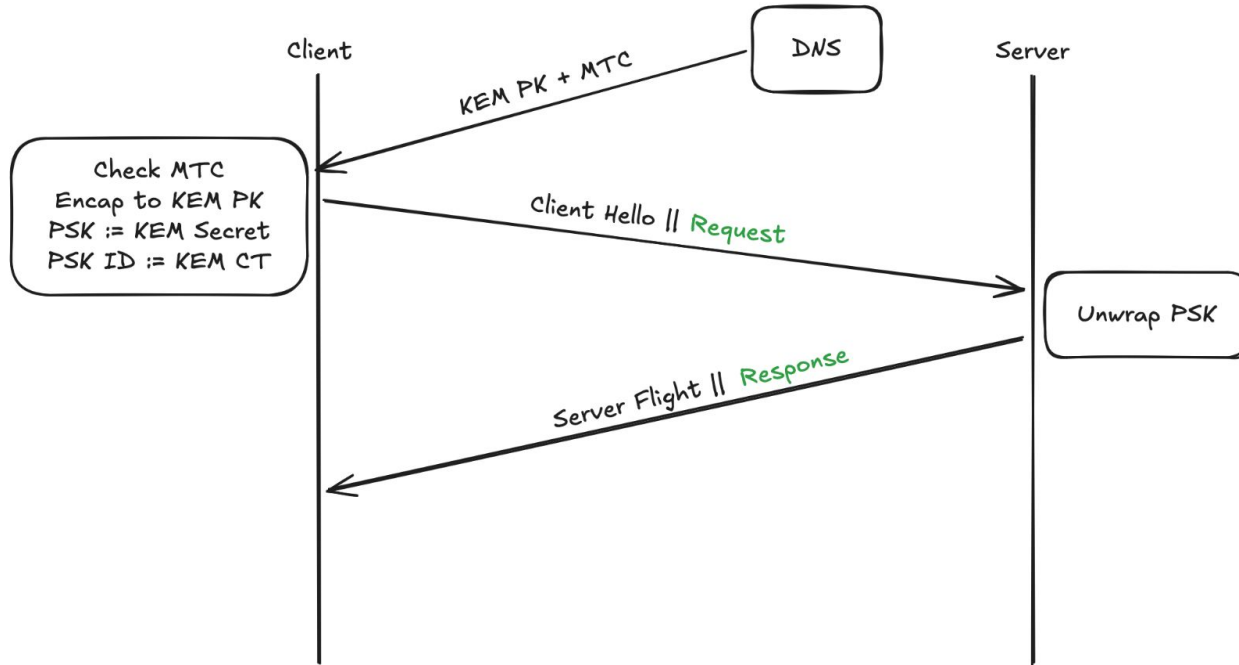
# TLS Preflight



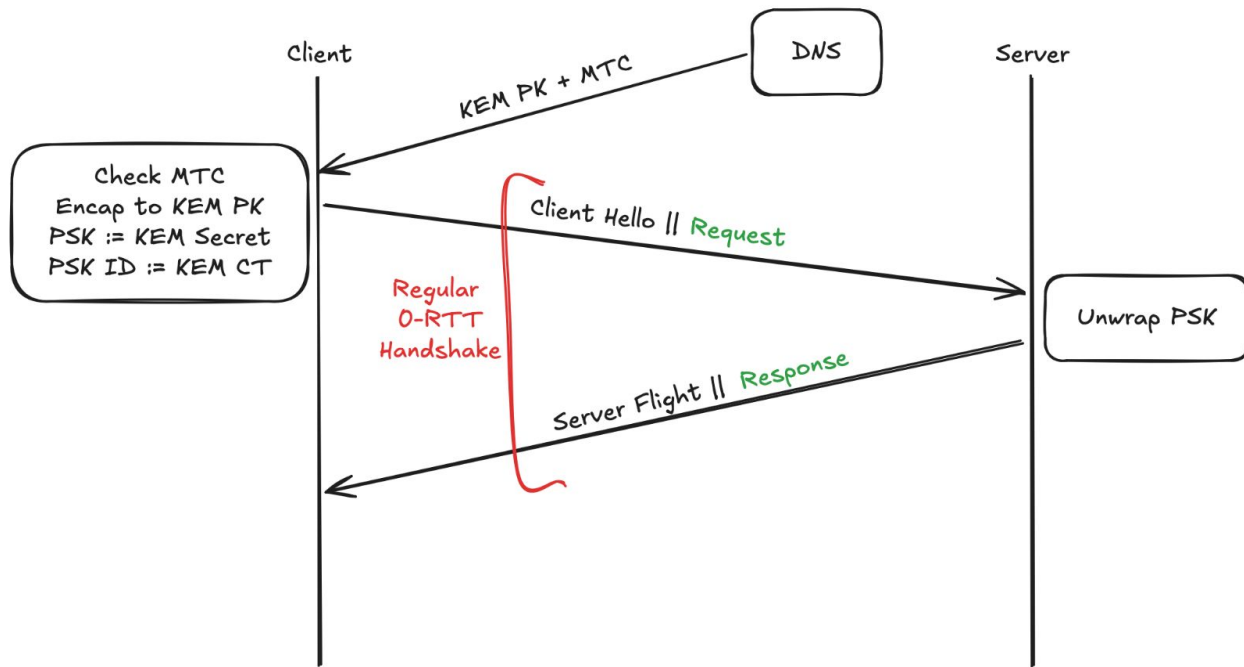
# TLS Preflight



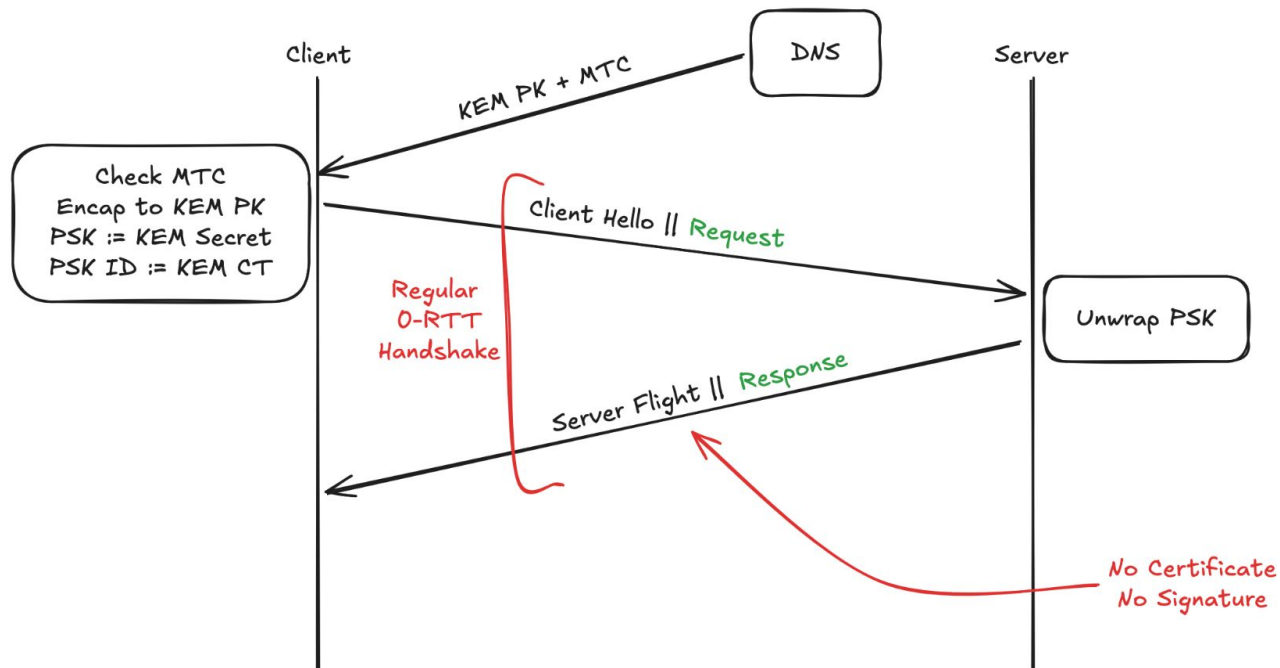
# TLS Preflight



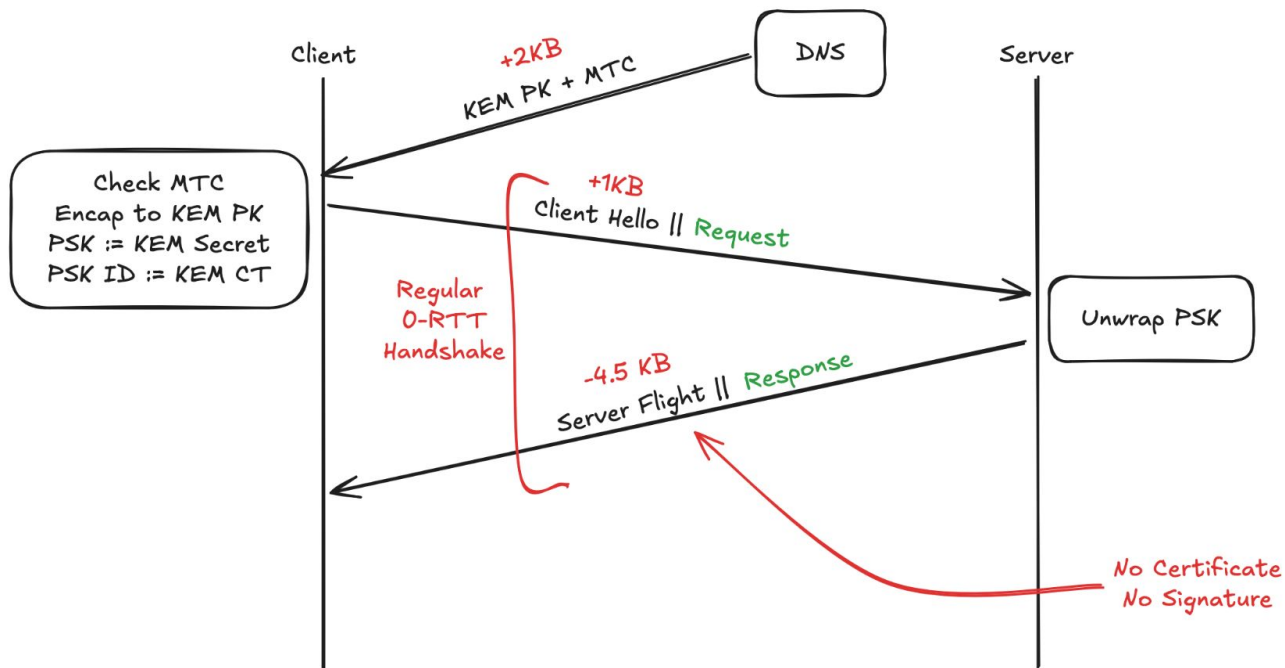
# Minimal Changes



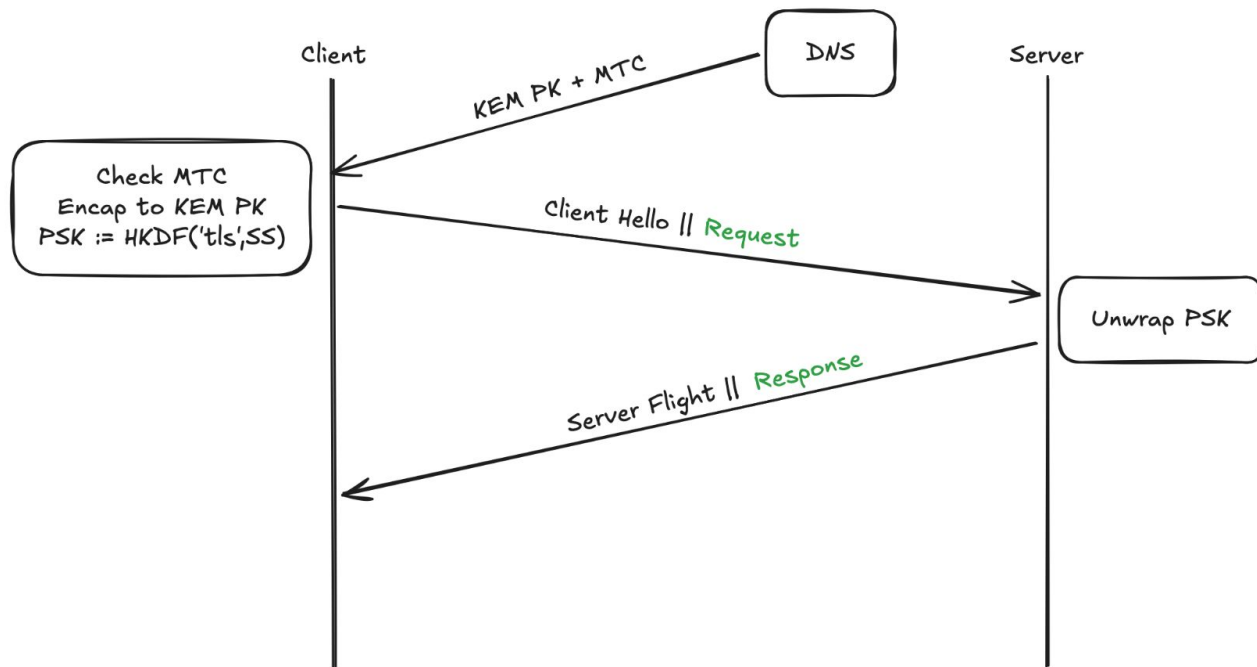
# Small



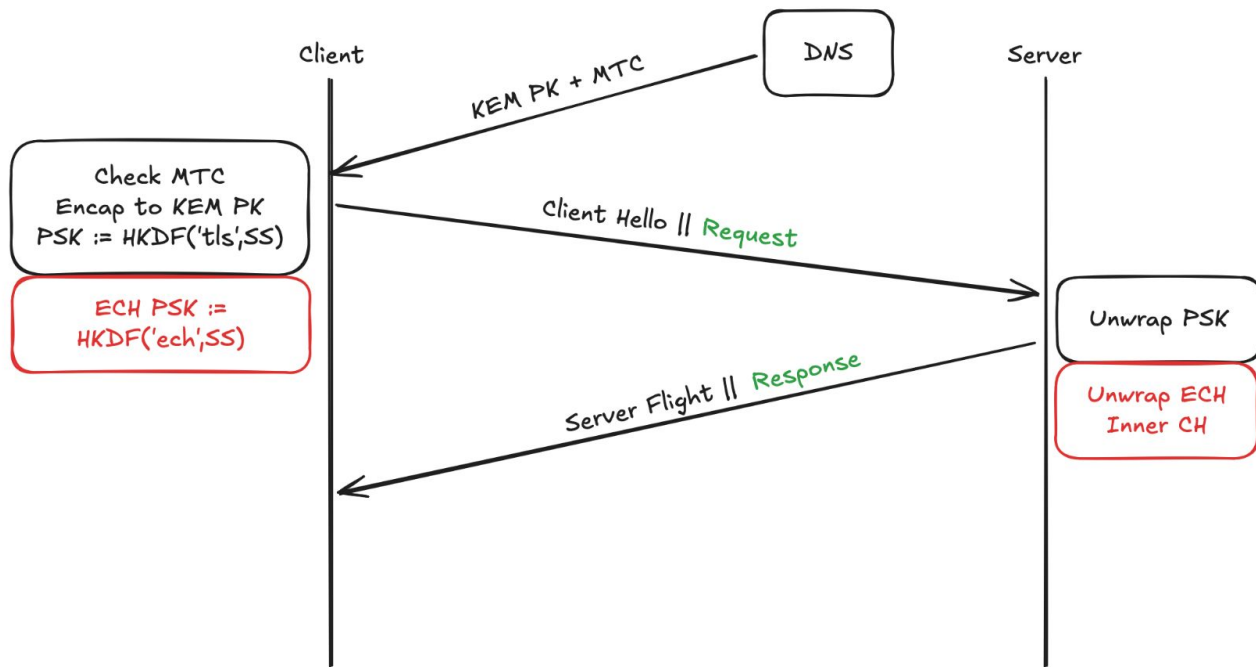
# Small



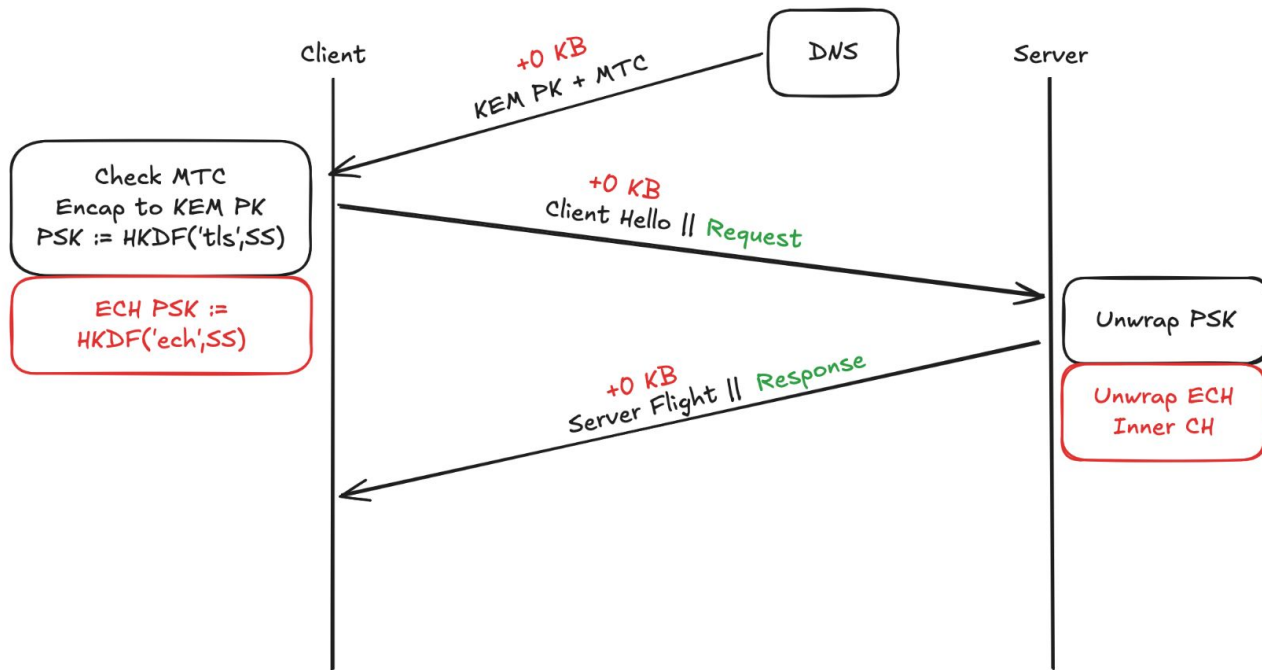
## Bonus: ECH for free



## Bonus: ECH for free



## Bonus: ECH for free



# Standardisation and Integration

- TLS Preflight needs three components:
  - MTC + 0-RTT Config in DNS HTTP RR
  - Minting a TLS PSK from KEM PK
  - ECH PSK Mode (optional)
- Preflight integrates cleanly with existing standards:
  - TLS 1.3 / Session Resumption / 0-RTT
  - ClientAuth (RFC 9973: Certificates with PSKs)
  - ECH

# Benefits

- More 0-RTT giving faster handshakes
- Size reduction of AuthKEM without the complexity
- ECH for no added cost
- No state machine changes in TLS
- Clean integration with existing standards
- Size benefits even with 0-RTT disabled

## Prior Work

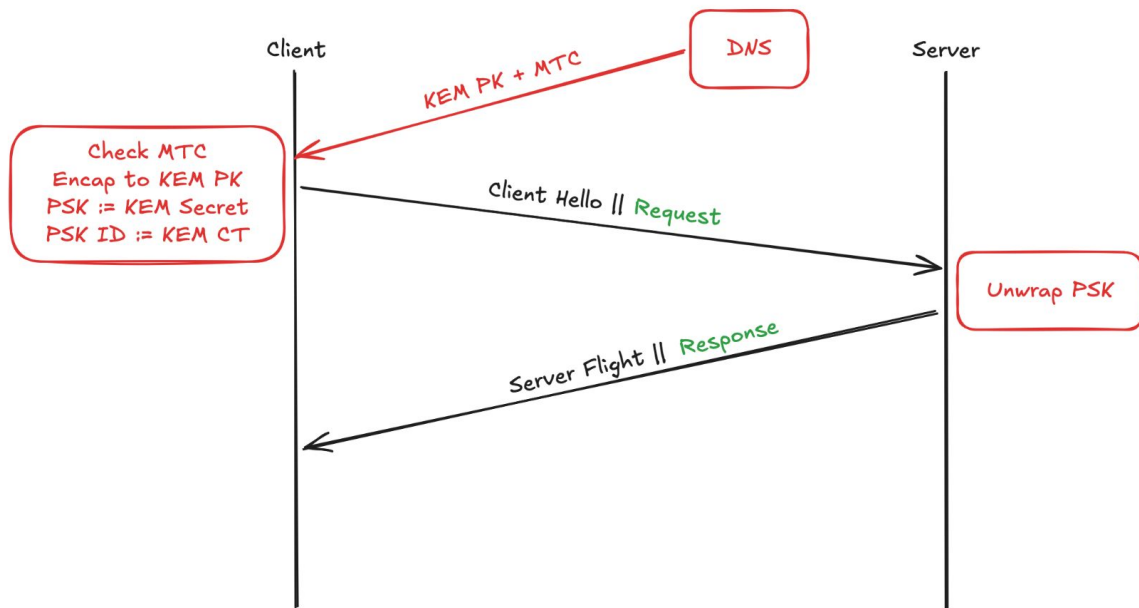
- gQUIC (2013) had a similar mechanism for 0-RTT
- AUTHKEM-PSK (2023) proposes a similar approach with its own extension

## **Why now?**

- PQ / MTCs make the size tradeoff more interesting
- ACME is widely deployed and certs have shorter lifetimes
- 0-RTT is a necessary property for OHTTP Streams

# FIN

## TLS Preflight



For little effort, we can have more 0-RTT and smaller handshakes