

draft-ietf-tls-pake

TLS 1.3 PAKE authentication

Laura Bauman, Apple

David Benjamin, Google

Samir Menon, Apple

Chris Wood, Apple

Since IETF 123

draft-ietf-tls-pake-00

- Draft adopted! 🎉
- Github: <https://github.com/tlswg/tls-pake>
- PRs up for all existing issues

Goals for Today

@ IETF 124

- Get working group feedback on existing issues + proposed solutions
- Identify topics that need to be discussed further

Existing Issues + PRs

- Issue [#35](#): Add CPace (PR up)
- Issue [#23](#): SPAKE2+ Context String? (PR up)
- Issue [#4](#): State properties required for PAKES used (PR up)
- Issue [#21](#): Server choice between PSK/PAKE/Certs (PR up)
- Issue [#6](#): Clarify client + server identity negotiation
- Issue [#26](#): Make future PAKE integration constraints more clear
 - Sub-Issue [#38](#): Generic vs protocol specific extension (PR up)
 - Sub-Issue [#39](#): Multi-Round PAKE Integration

Two Related Problems

1. What do we negotiate? Should we negotiate multiple-round PAKEs?
2. How do we negotiate? Should we use a generic extension for all PAKEs, protocol-specific extensions for each PAKE algorithm, or something else?

Issue #26: Make future PAKE integration constraints more clear

PAKE messages vs. TLS messages

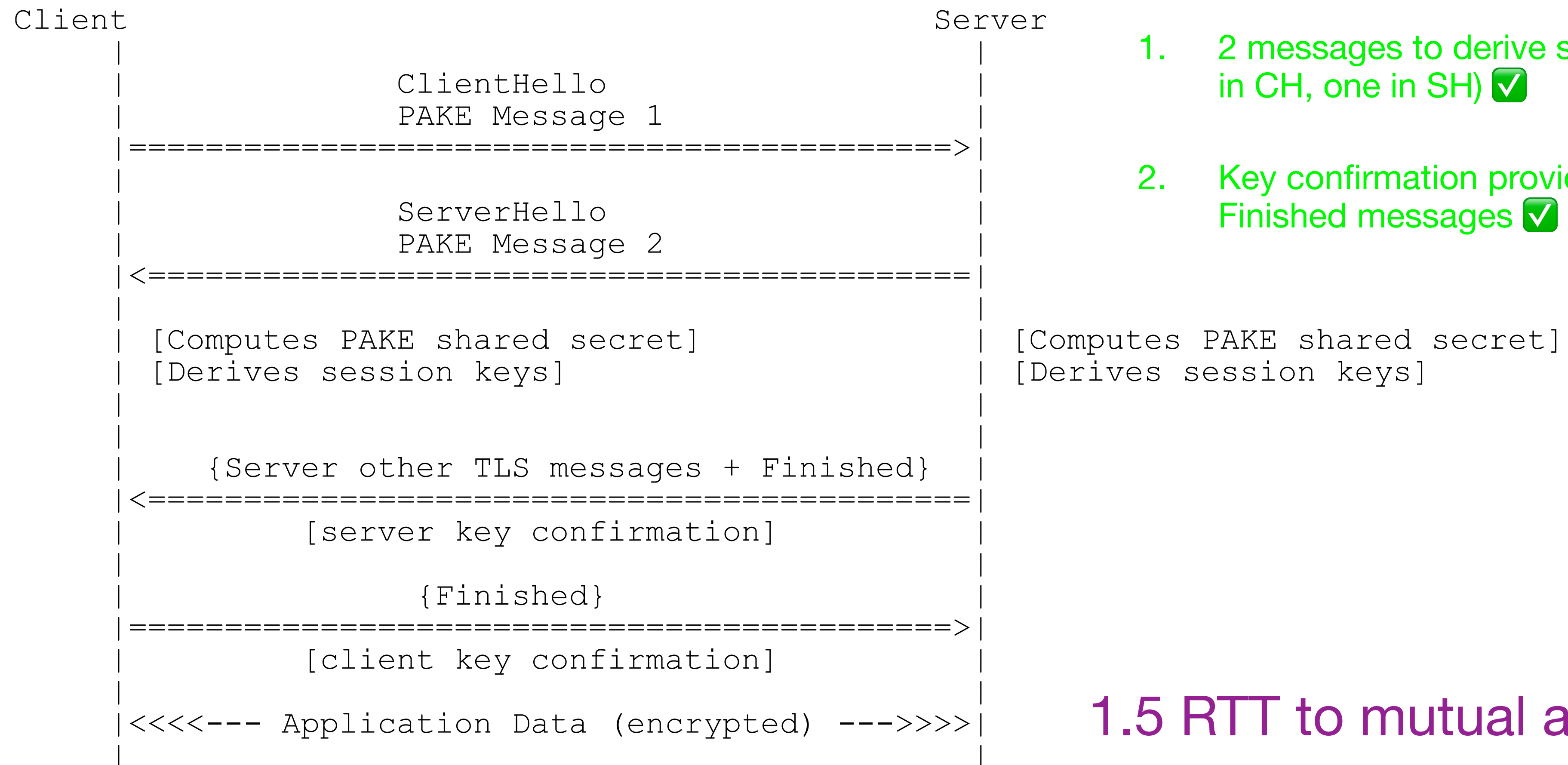
- To fit within the TLS 1.3 handshake message flow a PAKE needs:
 1. 2 messages to derive shared secret (one in CH, one in SH)
 2. Key confirmation provided by TLS Finished messages
 - e.g. if same secret derived —> authenticated
- **Problem:** Some PAKEs require more than 2 messages

Supporting Multi-Round PAKEs

- To support multi-round PAKEs we need a new handshake message
- Risks/Concerns:
 - Increased handshake time
 - Increased handshake complexity

2 Message PAKE w/ TLS-Finished Conf.

e.g. SPAKE2+ (classical, augmented), CPace (classical, symmetric) or OQUAKE (PQ, symmetric)

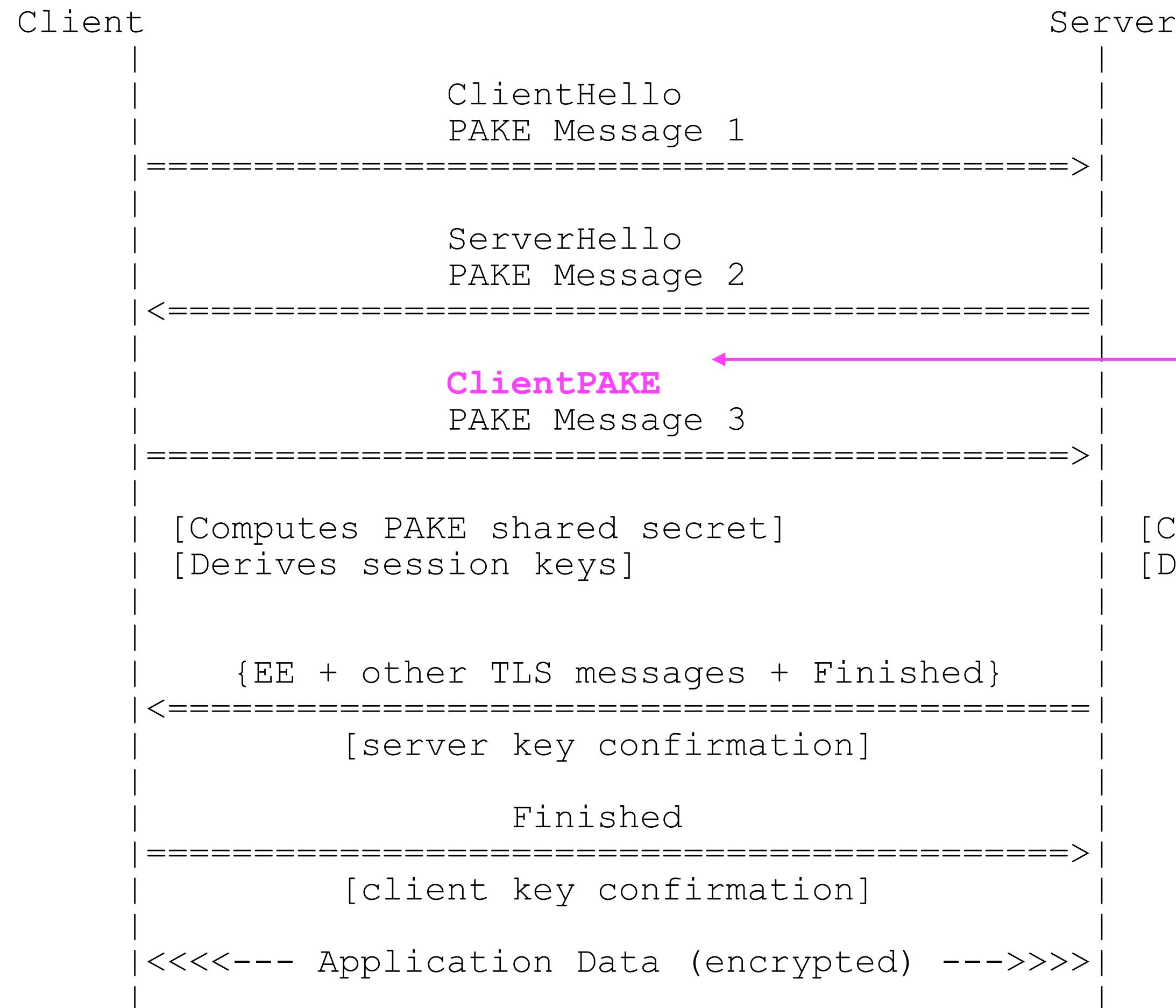


1. 2 messages to derive shared secret (one in CH, one in SH) ✓
2. Key confirmation provided by TLS Finished messages ✓

1.5 RTT to mutual auth

3 Message PAKE w/ TLS-Finished Conf.

e.g. CPaceOQUAKE (hybrid, symmetric)



1. 2 messages to derive shared secret (one in CH, one in SH) ❌

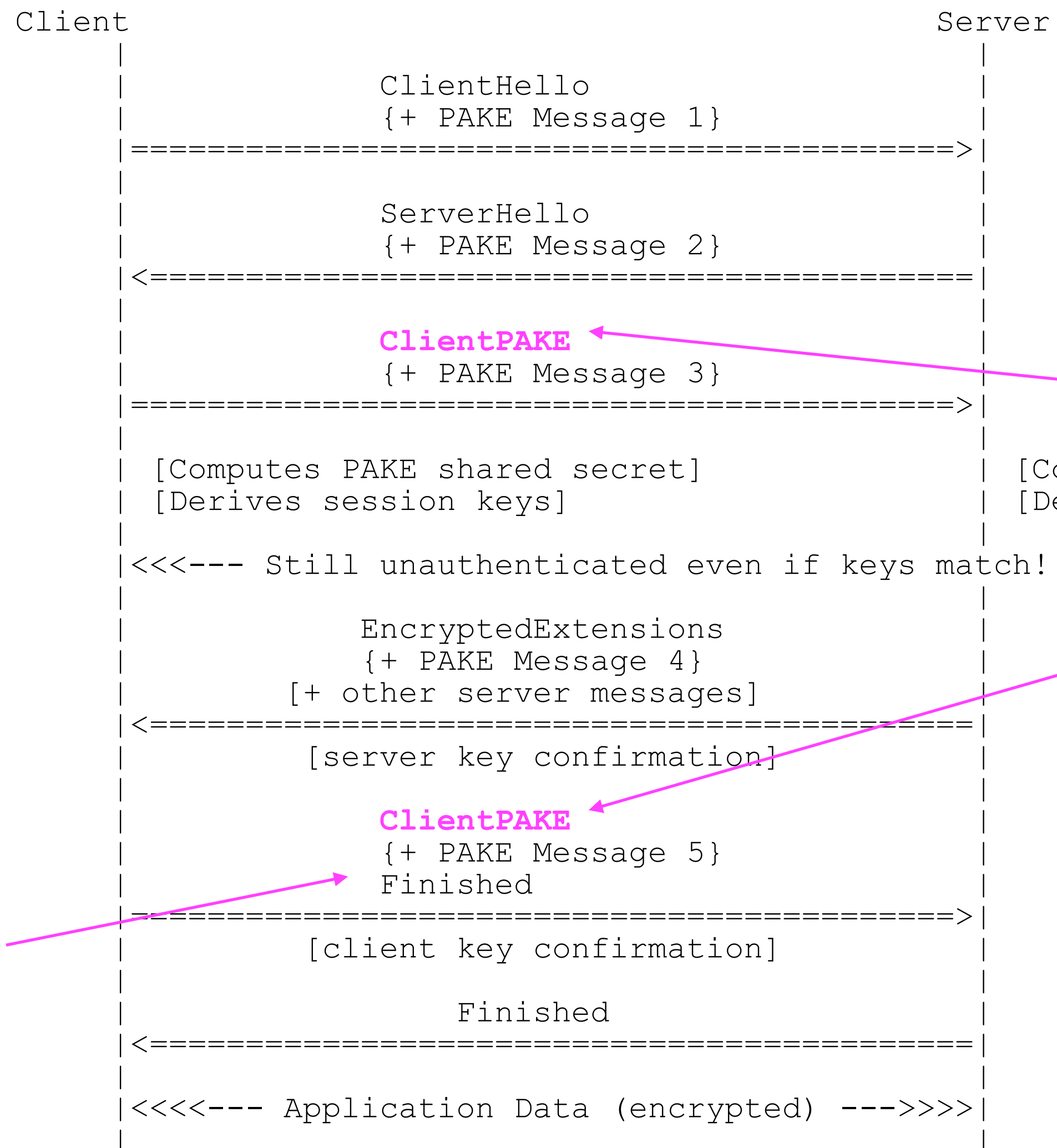
2. Key confirmation provided by TLS Finished messages ✅

New handshake message

2.5 RTT to mutual auth

5 Message PAKE w/ Explicit Key Conf.

e.g. CPaceOQUAKE+ (hybrid, augmented)



- 1. 2 messages to derive shared secret (one in CH, one in SH) ✗
- 2. Key confirmation provided by TLS Finished messages ✗

New handshake message

Client Finished sent before Server Finished

3 RTT to mutual auth

Sub-Issue #38: Generic vs protocol specific extension (PR up)

Options

- Protocol Specific extension (works for a specific PAKE)
- Generic extension (works for any PAKE)
- Message Count extension (works for n-round PAKEs)

Protocol Specific extension (works for a specific PAKE)

```
enum {
    pake_spake2plus(0xTODO),
    pake_cspace(0xTODO),
    pake_oquake(0xTODO),
    pake_cspaceoquake(0xTODO),
    pake_cspaceoquakeplus(0xTODO),
    ...
} ExtensionType;

struct {
    opaque    client_identity<0..2^16-1>;
    opaque    server_identity<0..2^16-1>;
    opaque    spake2plus_message<1..2^16-1>;
} PAKEClientHelloSPAKE2PLUS
```

- Good choice regardless of multi-round PAKE decision
 - Preferred for supporting multi-round PAKEs
- Each extension can specify allowed messages independently
- Not possible to “factor out” identities → duplication if offering multiple

Generic extension (works for any PAKE)

```
enum {
    pake(0xTODO),
} ExtensionType;

enum {
    SPAKE2PLUS_V1 (0XXXX),
    CPACE_X25519_SHA512 (0XXXX),
    OQUAKE (0XXXX),
    CPACE_OQUAKE (0XXXX),
    CPACE_OQUAKEPLUS (0XXXX),
    ...
} PAKEScheme;

struct {
    PAKEScheme    pake_scheme;
    opaque       pake_message<1..2^16-1>;
} PAKEShare;

struct {
    opaque       client_identity<0..2^16-1>;
    opaque       server_identity<0..2^16-1>;
    PAKEShare    client_shares<0..2^16-1>;
} PAKEClientHello;
```

- Existing design
- Just one extension definition
- Possible to consolidate identities between algorithms
 - Issue #6: Clarify client + server identity negotiation
- Potentially simpler server logic
- Awkward if we are supporting multi-round PAKEs

Per-Message

```
enum {  
    pake(0xTODO), // default 2 message  
    pake_three_msg(0xTODO),  
    pake_five_msg(0xTODO)  
} ExtensionType;
```

```
enum {  
    SPAKE2PLUS_V1 (0XXXX),  
    CPACE_X25519_SHA512 (0XXXX),  
    OQUAKE (0XXXX)  
} PAKESchemeTwoMessage
```

```
enum {  
    CPACE_OQUAKE (0XXXX)  
} PAKESchemeThreeMessage
```

```
enum {  
    CPACE_OQUAKEPLUS (0XXXX)  
} PAKESchemeFiveMessage;
```

- Several extensions
- Groups PAKEs by message flow
- Not possible to “factor out” identities → duplication if offering multiple

**Issue #21: Server choice between PSK/
PAKE (PR up)**

Issue #21: Server choice between PSK/PAKE (PR up)

- Client enumeration risk
 - If a client offers a PSK and a PAKE, server needs to choose.
 - If server decides based on whether it recognizes `client_identity`, attacker can perform enumeration
- Protocol-specific extension (pake_spake2plus, pake_cspace) would also be part of the problem
- **Solution:** Server SHOULD choose between PSK/PAKE extension(s) based on preference, not whether client_identity recognized.

**Issue #23: SPAKE2+ Context
String (PR up)**

Issue #23: SPAKE2+ Context String

- **Current:** context string passed to SPAKE2+ is empty or determined by application above TLS
- **Proposed:** “tls” | context