

# draft-bmw-tls-pake13-02

TLS 1.3 PAKE authentication extension

Laura Bauman, Apple

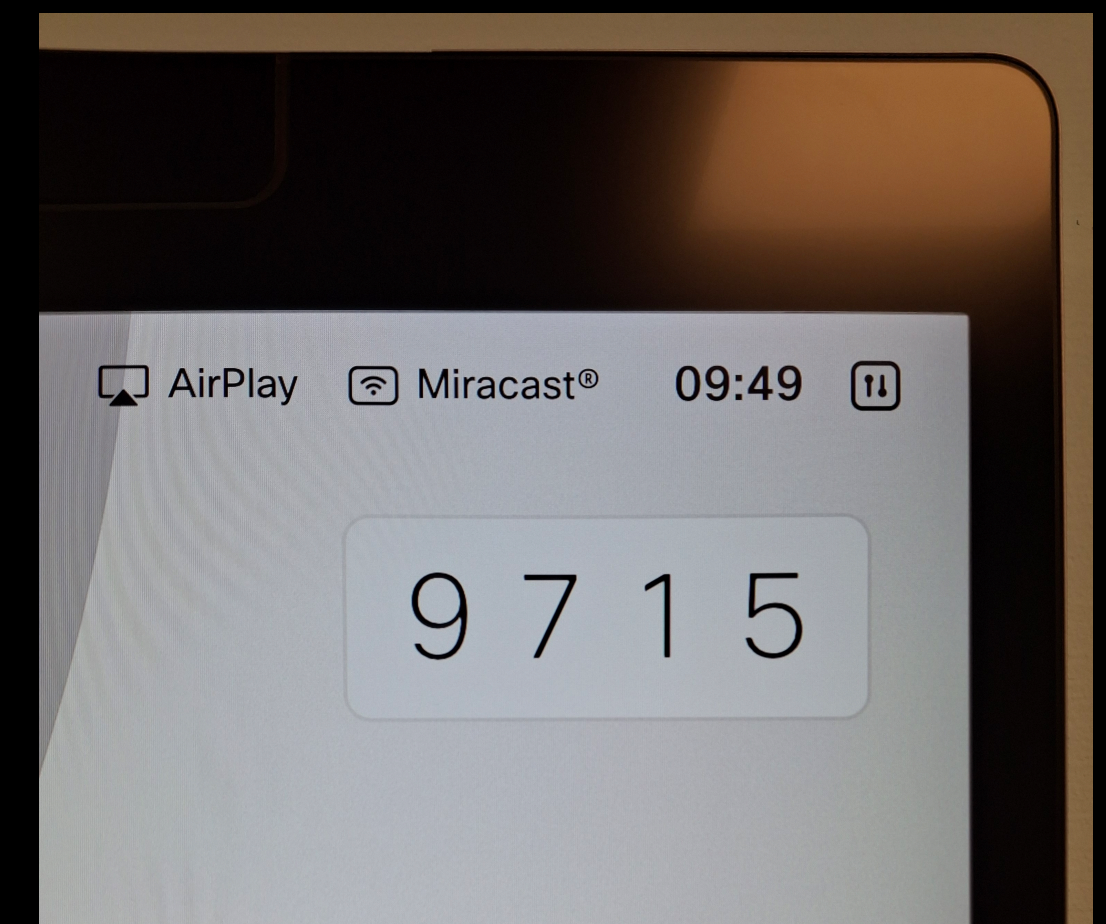
David Benjamin, Google

Samir Menon, Apple

Chris Wood, Apple

# Goals and Motivation

- TLS 1.3 provides a mechanism for authentication with pre-shared keys (PSKs), but the keys must be high entropy.
- There is no mechanism to directly establish a TLS 1.3 connection from a low entropy shared secret
- This is particularly useful for connections authenticated with a user-entered secret (PIN / passcode)
- NOT aiming for general web login use case
  - May be used to bootstrap long-term authentication
  - Could be used for a one time connection
  - Or as a connection over which long term credentials are exchanged



# Proposal

- TLS extension *pake* that can carry data necessary to execute a PAKE within the handshake.
- PAKE registration phase is application specific
- The choice of PAKE and any required parameters will be explicitly specified using IANA assigned values.
- Each PAKE is used as a black box
- As a first case, we've defined a concrete protocol for executing the SPAKE2+ PAKE protocol [RFC9383].

# Changes in version -02

- Certificates (Issue [#25](#))
  - Specifies interaction of pake extension with certificate authentication
- Combine with TLS key exchange (Issue [#28](#))
  - Specifies a mechanism of combining the secret derived from the PAKE with standard key exchange secret input to the key schedule

# PAKE Requirements

- A compatible PAKE MUST provide forward secrecy
- And conform to the message flow of the extension\*
- Several current PAKE protocols satisfy these requirements including:
  - CPace [CPACE] (PR up)
  - SPAKE2+ [RFC9383] (specified in the draft already)
  - OPAQUE [OPAQUE]

\* To support a hybrid PQ-PAKE the specified message flow would need to be modified to allow for an additional round trip ([draft-vos-cfrg-pqpake](#))

# Status + Next Steps

- Two implementations completed with support for SPAKE2+
  - Including an open source implementation in BoringSSL
- Adoption call?

# Client

0. (OOB or application specific registration phase)

# Server

1. Client sends an array of PAKEShares (PAKEScheme identifier + first message) it is willing to use

e.g. [{SPAKE2PLUS\_V1, SPAKE2+ client msg}]

**Client Hello**  
pake:  
client\_identity  
server\_identity  
[PAKEShares]

2. Server selects PAKEScheme from those offered and sends back next message

e.g. {SPAKE2PLUS\_V1, SPAKE2+ server msg}

**Server Hello**  
pake:  
PAKEShare

3. PAKE shared secret is concatenated with the (EC)DHE (or concatenated\_shared\_secret) input to the key schedule

e.g. SPAKE2+ derived shared secret

**Server Finished**

**Client Finished**