Network Working Group
Internet-Draft
Intended status: Informational
Expires: 8 January 2026

L. Bauman
Apple, Inc.
D. Benjamin
Google LLC
S. Menon
C. A. Wood
Apple, Inc.

7 July 2025

A Password Authenticated Key Exchange Extension for TLS 1.3
draft-bmw-tls-pake13-02

Abstract

The pre-shared key mechanism available in TLS 1.3 is not suitable for usage with low-entropy keys, such as passwords entered by users. This document describes an extension that enables the use of password-authenticated key exchange protocols with TLS 1.3.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 8 January 2026.

Copyright Notice

Table of Contents

1. Introduction

DISCLAIMER: Much of this text is copied from [FIRST-DRAFT] and is in the process of being updated. This is a work-in-progress draft and has not yet seen significant security analysis. See Section 8 and Section 8.4 for more information.

In some applications, it is desirable to enable a client and server to authenticate to one another using a low-entropy pre-shared value, such as a user-entered password.

In prior versions of TLS, this functionality has been provided by the integration of the Secure Remote Password PAKE protocol (SRP) [RFC5054]. The specific SRP integration described in RFC 5054 does not immediately extend to TLS 1.3 because it relies on the Client Key Exchange and Server Key Exchange messages, which no longer exist in 1.3.

TLS 1.3 itself provides a mechanism for authentication with pre-shared keys (PSKs). However, PSKs used with this protocol need to be "full-entropy", because the binder values used for authentication can be used to mount a dictionary attack on the PSK. So while the TLS 1.3 PSK mechanism is suitable for the session resumption cases for which it is specified, it cannot be used when the client and server share only a low-entropy secret.

Enabling TLS to address this use case effectively requires the TLS handshake to execute a password-authenticated key establishment (PAKE) protocol. This document describes a TLS extension pake that can carry data necessary to execute a PAKE.

This extension is generic, in that it can be used to carry key exchange information for multiple different PAKEs. We assume that prior to the TLS handshake the client and server will both have knowledge of the password or PAKE-specific values derived from the password (e.g. augmented PAKEs only require one party to know the actual password). The choice of PAKE and any required parameters will be explicitly specified using IANA assigned values. As a first case, this document defines a concrete protocol for executing the SPAKE2+ PAKE protocol [RFC9383].

2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

The mechanisms described in this document also apply to DTLS 1.3 [RFC9147], but for brevity, we will refer only to TLS throughout.

3. Setup

In order to use the extension specified in this document, a TLS client and server need to have pre-provisioned a password (or derived values as described by the desired PAKE protocol(s)). The details of this pre-provisioned information are specific to each PAKE algorithm and are not specified here.

Servers will of course have multiple instances of this configuration information for different clients. Clients may also have multiple identities, even within a given server.

4. PAKE Integration in TLS

This section describes how the PAKE protocol is integrated and executed in the TLS handshake.

4.1. Client Behavior

To offer support for a PAKE protocol, the client sends a pake extension in the ClientHello carrying a PAKEClientHello value:

```
enum {
    pake(0xTODO), (65535)
} ExtensionType;
```

The payload of the client extension has the following PAKEClientHello structure:

```
enum {
    SPAKE2PLUS_V1 (0xXXXX),
} 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;
```

The PAKEClientHello structure consists of an identity pair under which the client can authenticate alongside a list of PAKE algorithms and the client’s first message for each underlying PAKE protocol. Concretely, these structure fields are defined as follows:

client_shares A list of PAKEShare values, each one with a distinct PAKEScheme algorithm.

client_identity The client identity used for the PAKE. It may be empty.

server_identity The server identity used for the PAKE. It may be

empty.

pake_scheme The 2-byte identifier of the PAKE algorithm.

pake_message The client PAKE message used to initialize the protocol.

The client and server identity fields are common to all PAKEShares to prevent client enumeration attacks; see Section 8.

The PAKEScheme field in the PAKEShare allows implementations to support multiple PAKEs and negotiate which to use in the context of the handshake. For instance, if a client knows a password but not which PAKE the server supports it could send corresponding PAKEShares for each PAKE. If the client sends multiple PAKEShare values, then they MUST be sorted in monotonically increasing order by the NamedPAKE value. Moreover, the client MUST NOT send more than one PAKEShare with the same NamedPAKE value.

Section 9.2 of [TLS13] specifies that a valid ClientHello must include either a pre_shared_key extension or both a signature_algorithms and supported_groups extension. With the addition of the pake extension specified here, the new requirement is that a valid ClientHello must satisfy at least one of the following options:

* includes a pre_shared_key extension

* includes signature_algorithms, supported_groups, and key_share extensions

* includes pake, supported_groups, and key_share extensions

If a client sends the pake extension, then it MUST also send a supported_groups and key_share extension. Like PSK-based authentication in psk_dhe_ke mode as defined in Section 4.2.0 of [TLS13], authentication with the pake extension is always combined with the normal TLS key exchange mechanism. See Section 4.3 for details.

Combining the pake extension with the normal TLS key exchange mechanism using a hybrid or PQ key agreement protects against Harvest Now Decrypt Later Attacks where traffic recorded today may be decrypted by a Cryptographically Relevant Quantum Computer (CRQC) in the future.

A client which sends both a pake and signature_algorithms extension indicates the client requires both PAKE authentication and standard server certificate authentication.

The client MAY also send a pre_shared_key extension along with the pake extension, to allow the server to choose an authentication mode.

The server identity value provided in the PAKEClientHello structure are disjoint from that which the client may provide in the ServerNameIndication (SNI) field.

4.2. Server Behavior

A server that receives a pake extension examines its contents to determine if it is well-formed. In particular, if the list of PAKEShare values is not sorted in monotonically increasing order by PAKEScheme values, or if there are duplicate PAKEScheme entries in this list, the server aborts the handshake with an "illegal_parameter" alert.

If the list of PAKEShare values is well-formed, the server then scans the list of PAKEShare values to determine if there is one corresponding to a server supported PAKEScheme. If the server does not support any of the offered PAKESchemes in the client PAKEShares then the server MUST abort the protocol with an "illegal_parameter" alert.

If the server has a PAKEScheme in common with the client then the server uses the client_identity and server_identity alongside its local database of PAKE registration information to determine if the request corresponds to a legitimate client registration record. If one does not exist, the server MAY simulate a PAKE response as described in Section 4.4. Simulating a response prevents client enumeration attacks on the server’s PAKE database; see Section 8.

If there exists a valid PAKE registration, the server indicates its selection by including a pake extension in its ServerHello. The content of this extension is a PAKEServerHello value, specifying the PAKE the server has selected, and the server’s first message in the PAKE protocol. The format of this structure is as follows:

```
struct {
    PAKEShare server_share;
} PAKEServerHello;
```

The server_share value of this structure is a PAKEShare, which echoes back the PAKE algorithm chosen and the server’s PAKE message generated in response to the client’s PAKE message.

If a server uses PAKE authentication, then it MUST NOT send an extension of type pre_shared_key, or early_data.

Use of PAKE authentication MAY be used with certificate-based authentication of both clients and servers. If use of a PAKE is negotiated and the client included the signature_algorithms extension, then servers MUST include Certificate and CertificateVerify messages in the handshake. The server MAY send a CertificateRequest for client certificate authentication. See Section 8 for a discussion on different security considerations depending on if certificates are used or not.

4.3. Key Schedule Modifications

When the client and server agree on a PAKE to use, a shared secret derived from the PAKE protocol is concatenated with the regular ECDH(E) input and used as part of the ECDH(E) input to the TLS 1.3 key schedule. Details for the shared secret computation are left to the specific PAKE algorithm. See Section 6 for information about how the SPAKE2+ variant operates.

As with client authentication via certificates, the server has not authenticated the client until after it has received the client's Finished message. When a server negotiates the use of this mechanism for authentication, it SHOULD NOT send application data before it has received the client's Finished message, as it would otherwise be sending data to an unauthenticated client.

4.4. Server Simulation

To simulate a fake PAKE response, the server does the following:

* Select a PAKEScheme supported by the client and server, as normal.

* Include the pake extension in its ServerHello, containing a PAKEShare value with the selected PAKEScheme and corresponding pake_message. To generate the pake_message for this PAKEShare value, the server selects a value uniformly at random from the set of possible values of the PAKE algorithm shares.

* Perform the rest of the protocol as normal.

Because the server's share was selected uniformly at random, the server will reject the client's Finished message with overwhelming probability.

A server that performs the simulation of the protocol acts only as an all-or-nothing oracle for whether a given (identity, password) pair is correct. If an attacker does not supply a correct pair, they do not learn anything beyond this fact.

5. Compatible PAKE Protocols

In order to be usable with the pake extension, a PAKE protocol must specify some syntax for its messages, and the protocol itself must be compatible with the message flow described above. A specification describing the use of a particular PAKE protocol with TLS must provide the following details:

* A PAKEScheme registered value indicating pre-provisioned parameters;

* Content of the pake_message field in a ClientHello;

* Content of the pake_message field in a ServerHello;

* How the PAKE protocol is executed based on those messages; and

* How the outputs of the PAKE protocol are used to create the PAKE portion of the(EC)DHE input to the TLS key schedule.

In addition, to be compatible with the security requirements of TLS 1.3, PAKE protocols defined for use with TLS 1.3 MUST provide forward secrecy.

Several current PAKE protocols satisfy these requirements, for example:

* CPace [CPACE]

* SPAKE2+ (described in Section 6) [RFC9383]

* OPAQUE [OPAQUE]

6. SPAKE2+ Integration

This section describes the SPAKE2+ instantiation of the pake extension for TLS. The SPAKE2+ protocol is described in [SPAKE2PLUS]. Section 6.1 describes the setup required before the protocol runs, and Section 6.2 describes the protocol execution in TLS.

6.1. Protocol Setup

The TLS client and server roles map to the Prover and Verifier roles in the SPAKE2+ specification, respectively. Clients are configured with a client identity, server identity, and password verifier (w0 and w1 according to [SPAKE2PLUS]). Similarly, servers are configured with a list of client identity, server identity, and password registration values (w0 and L according to [SPAKE2PLUS]). Servers use this list when completing the SPAKE2+ protocol. The values for the password verifiers and registration records (w0, w1, and L) are not specified here; see Section 3.2 of [SPAKE2PLUS] for more information.

The PAKEScheme value for SPAKE2+ fully defines the parameters associated with the protocol, including the prime-order group G, cryptographic hash function Hash, key derivation function KDF, and message authentication code MAC. Additionally, the PAKEScheme value for SPAKE2+ fully defines the constants for M and N as needed for the protocol; see Section 4 of [SPAKE2PLUS].

6.2. Protocol Execution

The content of one PAKEShare value in the PAKEClientHello structure consists of the PAKEScheme value SPAKE2PLUS_V1 and the value shareP as computed in Section 3.3 of [SPAKE2PLUS].

The content of the server PAKEShare value in the PAKEServerHello structure consists of the PAKEScheme value SPAKE2PLUS_V1 and the value shareV || confirmV, i.e., shareV and confirmV concatenated, as computed in Section 3.3 of [SPAKE2PLUS].

Given shareP and shareV, the client and server can then both compute K_main, the root secret in the protocol as described in Section 3.4 of [SPAKE2PLUS]. The "Context" value for SPAKE2+ may be specified by the application to include additional context in the protocol transcript or left empty. See Section 3 of [SPAKE2PLUS]. The rest of the values needed for the transcript derivation are as configured in Section 6.1, exchanged over the wire, or computed by client and server.

Using K_main, the client and server both compute K_shared which is combined with the (EC)DHE shared secret as input to the TLS 1.3 key schedule, where the (EC)DHE shared secret is as specified in Section 7.1 of [TLS13] or as the concatenated_shared_secret as specified in Section 3.3 of [I-D.ietf-tls-hybrid-design]. Specifically, K_shared || (EC)DHE is used as the (EC)DHE input to the key schedule in Section 7.1 of [TLS13], as shown below.

Note that the server does compute and send confirmV as defined in Section 3.4 of [SPAKE2PLUS] since it can do so within the structure of the TLS 1.3 handshake and the client MUST verify it. If verification of confirmV fails, clients SHOULD abort the handshake with a "decrypt_error" alert. The client and server do not additionally compute or verify confirmP as described in Section 3.4 of [SPAKE2PLUS]. See Section 8.4 for more information about the safety of this approach.

7. Privacy Considerations

Client and server identities are sent in the clear in the PAKEClientHello extension. While normally the TLS server identity is already in the clear -- carried in the SNI extension -- TLS client identities are encrypted under the TLS handshake secrets. Thus, the PAKEClientHello extension reveals more information to a passive network attacker than normal, mutually-authenticated TLS handshakes.

The implications of leaking the client identity to a passive network attacker vary. For instance, a successful TLS handshake after negotiating use of a PAKE indicates that the chosen client identity is valid. This is relevant in settings where client enumeration may be a concern.

Applications for which this leak is a problem can use the TLS Encrypted ClientHello (ECH) extension to encrypt the PAKEClientHello extension in transit to the server [ECH].

8. Security Considerations

8.1. Dictionary attack mitigation

Because PAKE security is based on knowledge of a low-entropy secret, an attacker can perform a "dictionary attack" by repeatedly attempting to guess the low-entropy secret.

Clients and servers SHOULD apply mitigations against dictionary attacks. Reasonable mitigations include rate-limiting authentication attempts, imposing a backoff time between attempts, limiting the number of failed attempts, or limiting the total number of attempts.

Clients SHOULD treat each time they receive an invalid PAKEServerHello as a failed authentication attempt for the identity in the previously sent PAKEClientHello. Servers SHOULD treat each time they send a PAKEServerHello extension as a failed authentication attempt for the selected identity, until they receive a correct Finished message from the client. Once the server receives a correct Finished message, the authentication attempt MAY be treated as successful.

8.2. Protection of client identities

Many of the security properties of this protocol will derive from the PAKE protocol being used. Security considerations for PAKE protocols are noted in Section 5.

If a server doesn’t recognize the identity supplied by the client in the ClientHello pake extension, the server MAY abort the handshake with an "illegal_parameter" alert. In this case, the server acts as an oracle for identities, in which each handshake allows an attacker to learn whether the server recognizes a given identity.

Alternatively, if the server wishes to hide the fact that a client identity is unrecognized, the server MAY simulate the protocol as if an identity was recognized, but the password was incorrect. This is similar to the procedure outlined in [RFC5054]. The simulation mechanism is described in Section 4.4.

8.3. Ramifications of low entropy secret compromise

As with PSK based authentication, if only PAKE authentication is in use, then an attacker that learns the low entropy secret could impersonate either the client or the server. In situations where a notion of stable identity is available, then certificate-based authentication MAY be used as well to reduce this risk. For example, requiring the server to authenticate with a certificate in addition to PAKE authentication means an attacker that learns the password could only impersonate a client to a server, but could not impersonate a server to a client. This is an important distinction in situations where the client sends sensitive data to the server.

8.4. SPAKE2+ Security Considerations

Section 6 describes how to integrate SPAKE2+ into TLS using the pake extension in this document. This integration deviates from the SPAKE2+ protocol in [SPAKE2PLUS] in one important way: the explicit key confirmation checks required in [SPAKE2PLUS] are replaced with the TLS Finished messages. This is because the TLS Finished messages compute a MAC over the TLS transcript, which includes both the shareP and shareV values exchanged for SPAKE2+.

[[OPEN ISSUE: this requires formal analysis to confirm.]]

9. IANA Considerations

This document requests that IANA add a value to the TLS ExtensionType Registry with the following contents:

Value	Extension Name	TLS 1.3	Reference
0xTODO	pake	CH, SH	(this document)

Table 1

[[RFC EDITOR: Please replace "TODO" in the above table with the value assigned by IANA, and replace "(this document)" with the RFC number assigned to this document.]]

9.1. PAKE Scheme registry

This document requests that IANA create a new registry called "PAKE Schemes" with the following contents:

Value	PAKEScheme	Reference	Notes
0xTODO	SPAKE2PLUS_V1	(this document)	N/A

Table 2

The SPAKE2PLUS_V1 PAKEScheme variant has the following parameters associated with it:

* G: P-256
* Hash: SHA256
* KDF: HKDF-SHA256
* MAC: HMAC-SHA256

Additionally, it uses the M and N values from Section 4 of [SPAKE2PLUS], included below, as compressed points on the P-256 curve, for completeness.

M =
02886e2f97ace46e55ba9dd7242579f2993b64e16ef3dcab95afd497333d8fa12f

N =
03d8bbd6c639c62937b04d997f38c3770719c629d7014d49a24b4f98baa1292b49

Acknowledgments

The authors would like to thank the original authors of [FIRST-DRAFT] for providing a firm basis for the extension mechanism specified in this document.

Change Log

Since draft-bmw-tls-pake13-01

* Require standard TLS Key exchange to be combined with pake
* Allow combining PAKEs and certificates

References

Normative References

[CPACE] Abdalla, M., Haase, B., and J. Hesse, "CPace, a balanced composable PAKE", Work in Progress, Internet-Draft, draft-irtf-cfrg-cpace-14, 16 April 2025, <https://datatracker.ietf.org/doc/html/draft-irtf-cfrg-cpace-14>.

[I-D.ietf-tls-hybrid-design] Stebila, D., Fluhrer, S., and S. Gueron, "Hybrid key exchange in TLS 1.3", Work in Progress, Internet-Draft, draft-ietf-tls-hybrid-design-13, 17 June 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-tls-hybrid-design-13>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/rfc/rfc2119>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/rfc/rfc8174>.

[RFC9147] Rescorla, E., Tschofenig, H., and N. Modadugu, "The Datagram Transport Layer Security (DTLS) Protocol Version 1.3", RFC 9147, DOI 10.17487/RFC9147, April 2022, <https://www.rfc-editor.org/rfc/rfc9147>.

[RFC9383] Taubert, T. and C. A. Wood, "SPAKE2+, an Augmented Password-Authenticated Key Exchange (PAKE) Protocol", RFC 9383, DOI 10.17487/RFC9383, September 2023, <https://www.rfc-editor.org/rfc/rfc9383>.

[SPAKE2PLUS] Taubert, T. and C. A. Wood, "SPAKE2+, an Augmented Password-Authenticated Key Exchange (PAKE) Protocol", RFC 9383, DOI 10.17487/RFC9383, September 2023, <https://www.rfc-editor.org/rfc/rfc9383>.

[TLS13] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <https://www.rfc-editor.org/rfc/rfc8446>.

Informative References

[ECH] Rescorla, E., Oku, K., Sullivan, N., and C. A. Wood, "TLS Encrypted Client Hello", Work in Progress, Internet-Draft, draft-ietf-tls-esni-25, 14 June 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-tls-esni-25>.

[FIRST-DRAFT] Barnes, R. and O. Friel, "Usage of PAKE with TLS 1.3", Work in Progress, Internet-Draft, draft-barnes-tls-pake-04, 16 July 2018, <https://datatracker.ietf.org/doc/html/draft-barnes-tls-pake-04>.

[OPAQUE] Bourdrez, D., Krawczyk, H., Lewi, K., and C. A. Wood, "The OPAQUE Augmented PAKE Protocol", Work in Progress, Internet-Draft, draft-irtf-cfrg-opaque-18, 21 November 2024, <https://datatracker.ietf.org/doc/html/draft-irtf-cfrg-opaque-18>.

[RFC5054] Taylor, D., Wu, T., Mavrogiannopoulos, N., and T. Perrin, "Using the Secure Remote Password (SRP) Protocol for TLS Authentication", RFC 5054, DOI 10.17487/RFC5054, November 2007, <https://www.rfc-editor.org/rfc/rfc5054>.

Authors' Addresses

Laura Bauman
Apple, Inc.
Email: l_bauman@apple.com

David Benjamin
Google LLC
Email: davidben@google.com

Samir Menon
Apple, Inc.
Email: samir_menon@apple.com

Christopher A. Wood
Apple, Inc.
Email: caw@heapingbits.net