Internet Engineering Task Force
Internet-Draft
Intended status: Standards Track
Expires: 23 April 2026

W. Guo
L. Xia
J. Li
Y. Li
Huawei Technologies

20 October 2025

PAKE Extension for TLS Exported Authenticator
draft-guo-tls-exported-pake-00

Abstract

This document provides a mechanism that enables the use of password-authenticated key exchange (PAKE) with TLS Exported Authenticator, and that supports PAKE algorithm negotiation.

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 23 April 2026.

Copyright Notice

Table of Contents

1. Introduction

In some cases, it is desirable to enable one party and its peers of a TLS or DTLS connection to authenticate mutually using password-authenticated key exchange (PAKE) protocols after the connection has been established. This not only can defend against password leakages caused by PKI failures (e.g., phishing attacks) when passwords are transmitted in plaintext over the (D)TLS connection, but also can make changes to the TLS layer as small as possible when compared to using PAKEs to establish the (D)TLS connection. This strategy is often called defense-in-depth, the security of application-layer password authentication is still guaranteed even if some failures occur at the underlying (D)TLS layer.

Note that the use of OPAQUE [RFC9807] with Exported Authenticator [RFC9261] has been discussed in [I-D.draft-sullivan-tls-opaque-01]. However, it does not consider PAKE algorithms negotiation, which is very important in practice, because some algorithms may be broken in the future and we need some way to negotiate new algorithms. This document provides a mechanism that uses a PAKE extension for (D)TLS Exported Authenticator to execute application-layer password authentication at any time after the (D)TLS handshake has completed, which supports PAKE algorithm negotiation.

The minimum version of TLS and DTLS required to implement the mechanism described in this document are TLS 1.2 [RFC5246] and DTLS 1.2 [RFC6347]. (These were obsoleted by TLS 1.3 [RFC8446] and DTLS 1.3 [RFC9147].)

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.

This document uses terminology such as client, server, connection, handshake, endpoint, and peer that are defined in Section 1.1 of [RFC8446].

3. PAKE Algorithm Negotiation

This document defines a new TLS extension type "pake_algorithm". This extension can only appear in ClientHello and EncryptedExtensions messages, otherwise peers MUST abort the handshake with an "illegal_parameter" alert. The "extension_data" field of this extension contains a "PAKEAlgorithmList" value.

```
enum {
  /* Balanced PAKE algorithms */
  0x0000~0x3FFF

  /* Augmented PAKE algorithms */
  0x4000~0x7FFF

  /* Post-Quantum Balanced PAKE algorithms */
  0x8000~0xBFFF

  /* Post-Quantum Augmented PAKE algorithms */
  0xC000~0xFFFF
  (0xFFFF)
} PAKEAlgorithm;

struct {
    PAKEAlgorithm pake_algorithm_list<2..2^16-1>;
} PAKEAlgorithmList;
```

In this document, two PAKE algorithms are considered: SPAKE2 [RFC9382] and SPAKE2+ [RFC9383]. The former is a symmetric PAKE algorithm and the latter is an asymmetric PAKE algorithm. Here, each PAKE algorithm suite consists of two part, the first part is represented by the first byte and specifies different PAKE algorithms, and the second part is represented by the second byte and specifies different ciphersuites for some PAKE algorithm. For example, the first bytes "0x01" and "0x41" represent SPAKE2 [RFC9382] and SPAKE2+ [RFC9383], respectively. For the SPAKE2+ algorithm with the first byte "0x41", the second bytes "0x00~0xFF" can be used to represent its different ciphersuites (see Section 4 of [RFC9383]).

The "pake_algorithm" extension MAY be included by the client in its ClientHello message, and the corresponding "extension_data" field contains a list of PAKE algorithm suites supported by the client, ordered from most preferred to least preferred. After receiving a ClientHello message containing the "pake_algorithm" extension, the server MAY return a suitable PAKE algorithm selection response to the client. The server will ignore any algorithm that it does not support. The "pake_algorithm" extension MAY be returned by the server in its EncryptedExtensions message, and the corresponding "extension_data" field MUST contain exactly one "PAKEAlgorithm".

Therefore, a full handshake with the "pake_algorithm" extension in the ClientHello and EncryptedExtensions messages has the following flow (see Figure 1).

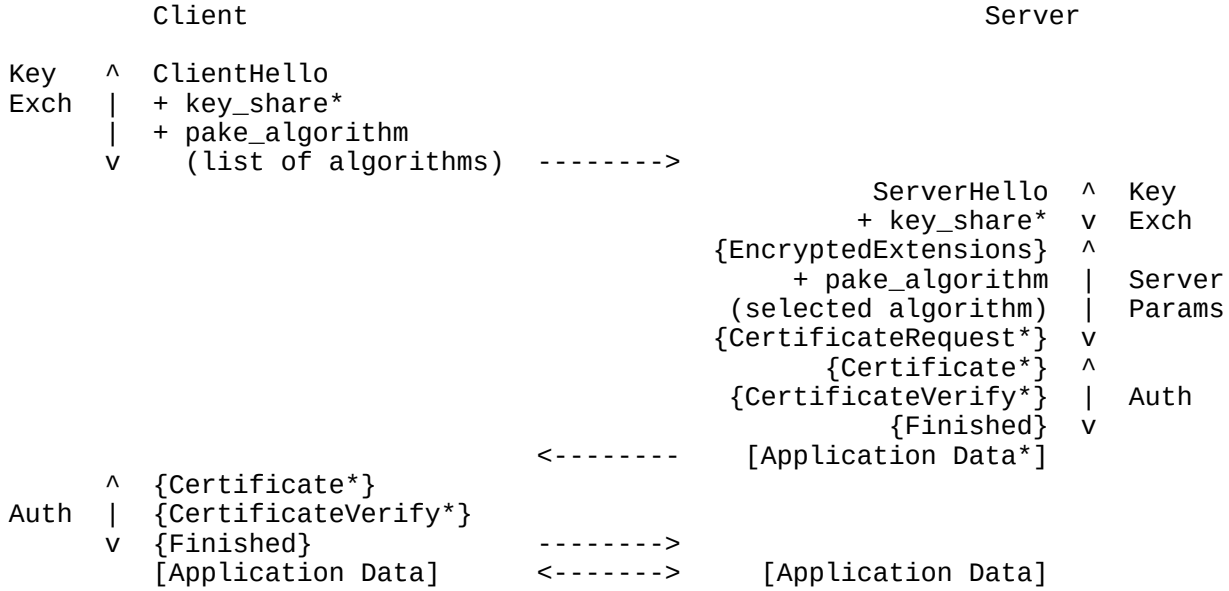

Figure 1: Message Flow for a Full TLS Handshake with the pake_algorithm Extension

4. PAKE with TLS Exported Authenticator

4.1. TLS CertificateRequest Message Extension

This document also defines a new TLS extension type "pake_share". This extension can only appear in CertificateRequest (or ClientCertificateRequest) message during Exported Authenticator-based post-handshake authentication, otherwise peers MUST abort the handshake with an "illegal_parameter" alert.

The "pake_share" extension contains the endpoint's PAKE parameters. The "PAKEShareEntry" value is defined as follows.

```
struct {
    PAKEAlgorithm pake_algorithm;
    opaque pake_message<1..2^16-1>;
} PAKEShareEntry;
```

* The "pake_algorithm" field indicates the PAKE algorithm used.

* The "pake_message" field indicates the PAKE key exchange information, its contents are determined by the specified PAKE algorithm, as shown below.

```
struct {
    select (pake_algorithm) {
    case SPAKE2:
        opaque dh_share;
    case SPAKE2+:
        opaque dh_share;
    };
} PAKEMessage;
```

* The "dh_share" field indicates a (EC)DH public element that is determined by the specified PAKE algorithm and the underlying group. 1) for SPAKE2, this field indicates the value pA or pB in Section 3.3 of [RFC9382]; 2) for SPAKE2+, this field indicates the value shareP or shareV in Section 3.3 of [RFC9383].

(1) In a ClientCertificateRequest message, the "extension_data" field of this extension contains a "ClientPAKEShare" value.

```
struct {
    opaque client_identity<0..2^16-1>;
    PAKEShareEntry client_share;
} ClientPAKEShare;
```

* The "client_identity" field indicates a client's identity used in the PAKE algorithm.

* The "client_share" field contains a single "PAKEShareEntry" value, where the "pake_algorithm" field is set to the negotiated PAKE algorithm in the (D)TLS handshake.

(2) In a CertificateRequest message, the "extension_data" field of this extension contains a "ServerPAKEShare" value.

```
struct {
    PAKEShareEntry server_share;
} ServerPAKEShare;
```

* The "server_share" field contains a single "PAKEShareEntry" value, where the "pake_algorithm" field is also set to the negotiated PAKE algorithm in the (D)TLS handshake.

4.2. Message Sequence

Note that two messages are defined in [RFC9261]: authenticator requests and authenticators. After the (D)TLS handshake (with the "pake_algorithm" extension) has completed, these messages can be combined in the following sequence to achieve PAKE-based mutual authentication at the application layer.

* The client generates an authenticator request (ClientCertificateRequest with the "pake_share" extension).

* The server generates an authenticator request (CertificateRequest with the "pake_share" extension) and an authenticator (server Finished) corresponding to the client's authenticator request.

* The client validates the server's authenticator and generates an authenticator (client Finished) corresponding to the server's authenticator request.

* The server validates the client's authenticator.

Figure 2 shows a full message sequence of using PAKE with TLS Exported Authenticator, followed by a detailed description.

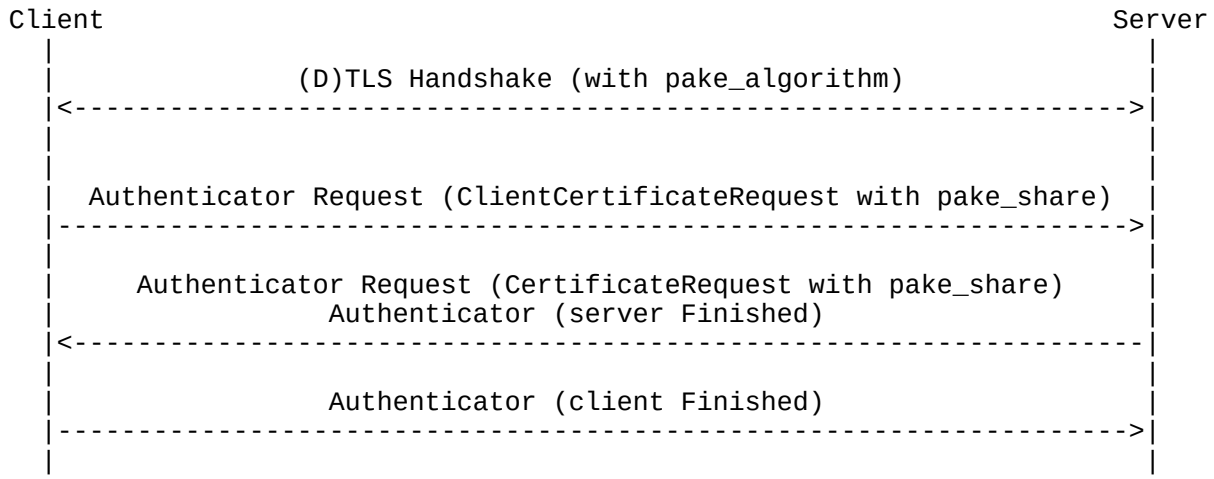

Figure 2: Message Sequence of Using PAKE with TLS Exported Authenticator

(1) When PAKE-based mutual authentication at the application layer needs to be performed, it is REQUIRED to generate a ClientCertificateRequest message. The client sets the "certificate_request_context" field to a randomly generated string, and sets the "pake_share" extension to a "ClientPAKEShare" value, where the "client_identity" is its identity used to authenticate and the "client_share" field is constructed by setting the negotiated PAKE algorithm suite and computing its corresponding PAKE share. The computation of PAKE share SHOULD conform to the specification of the used PAKE algorithm. The client then sends the ClientCertificateRequest message to the server.

(2) After receiving the ClientCertificateRequest message, the server first parses it to obtain "certificate_request_context", "client_identity" and "client_share" fields, uses the "client_identity" value to search a match password or password file. To generate a CertificateRequest message, the server sets the "certificate_request_context" field to a randomly generated string, and sets the "pake_share" extension to a "ServerPAKEShare" value, where the "server_share" field is constructed by setting the negotiated PAKE algorithm suite and computing its corresponding PAKE share. Based on the received client’s share and its own secret, the server first derives a PAKE shared secret, and then derives two keys as follows.

client_key || server_key = KDF(pake_shared_secret, H(ClientCertificateRequest || CertificateRequest), "TLSExportedPAKE", L)

Here, KDF(ikm, salt, info, L) and H(.) indicate the key-derivation function (KDF) and the hash function, respectively, which are negotiated in the PAKE algorithm suite; the PAKE shared secrets for different PAKEs are defined as follows:

* For SPAKE2, this secret indicates the concatenated value Ke || Ka in Section 3.3 of [RFC9382].

* For SPAKE2+, this secret indicates the value K_main in Section 3.3 of [RFC9383].

Then, the server computes its "Finished" value as follows, where the Handshake Context, Finished MAC Key, Hash are defined in Section 5.1 of [RFC9261] and HMAC is defined in Section 5.2.3 of [RFC9261]. Finally, the server sends the CertificateRequest and server Finished messages to the client.

server Finished = HMAC(Hash(server Finished MAC Key || server_key), Hash(server Handshake Context || client authenticator request))

(3) After receiving the CertificateRequest and server Finished messages, the client first parses the former to obtain the "certificate_request_context" and "server_share" fields. Based on the received server’s share and its own secret, the client derives a PAKE shared secret, and then derives two keys using the same way as the server side. The client then authenticates the server by verifying the server "Finished" value. If this verification succeeds, the client computes its "Finished" value as follows, and sends the client Finished message to the server.

client Finished = HMAC(Hash(client Finished MAC Key || client_key), Hash(client Handshake Context || server authenticator request))

Otherwise, the client MAY terminate the (D)TLS connection.

(4) After receiving the client Finished message, the server authenticates the client by verifying the client "Finished" value. If this verification succeeds, the server accepts the client’s request at the application layer, otherwise rejects.

5. IANA Considerations

This document defines two new TLS extension types "pake_algorithm" and "pake_share" with the following contents (see Table 1), and requests that IANA add the two values to the "TLS ExtensionType Values" Registry defined in [RFC8446] and [RFC8447].

Value	Extension Name	TLS 1.3	Reference
TBD1	pake_algorithm	CH, EE	RFC XXXX
TBD2	pake_share	CR	RFC XXXX

Table 1: New TLS Extension Types

6. References

6.1. Normative References

[RFC9807] Bourdrez, D., Krawczyk, H., Lewi, K., and C. A. Wood, "The OPAQUE Augmented Password-Authenticated Key Exchange (aPAKE) Protocol", RFC 9807, DOI 10.17487/RFC9807, July 2025, <https://www.rfc-editor.org/rfc/rfc9807>.

[RFC9261] Sullivan, N., "Exported Authenticators in TLS", RFC 9261, DOI 10.17487/RFC9261, July 2022, <https://www.rfc-editor.org/rfc/rfc9261>.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <https://www.rfc-editor.org/rfc/rfc5246>.

[RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, DOI 10.17487/RFC6347, January 2012, <https://www.rfc-editor.org/rfc/rfc6347>.

[RFC8446] 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>.

[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>.

[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>.

[RFC9382] Ladd, W., "SPAKE2, a Password-Authenticated Key Exchange", RFC 9382, DOI 10.17487/RFC9382, September 2023, <https://www.rfc-editor.org/rfc/rfc9382>.

[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>.

[RFC8447] Salowey, J. and S. Turner, "IANA Registry Updates for TLS and DTLS", RFC 8447, DOI 10.17487/RFC8447, August 2018, <https://www.rfc-editor.org/rfc/rfc8447>.

6.2. Informative References

[I-D.draft-sullivan-tls-opaque-01] Sullivan, N., Krawczyk, H., Friel, O., and R. Barnes, "OPAQUE with TLS 1.3", Work in Progress, Internet-Draft, draft-sullivan-tls-opaque-01, 22 February 2021, <https://datatracker.ietf.org/doc/html/draft-sullivan-tls-opaque-01>.

Contributors

Hui Ye
Huawei Technologies
yehui.ustc@huawei.com

Feng Geng
Huawei Technologies
gengfeng@huawei.com

Authors’ Addresses

Wei Guo
Huawei Technologies
Email: guowei90@huawei.com

Liang Xia
Huawei Technologies
Email: frank.xialiang@huawei.com

Ji Li
Huawei Technologies
Email: liji100@huawei.com

Yong Li
Huawei Technologies
Email: Yong.Li1@huawei.com