Network Working Group
Internet-Draft
Intended status: Informational
Expires: January 17, 2019

R. Barnes
O. Friel
Cisco
July 16, 2018

Usage of PAKE with TLS 1.3
draft-barnes-tls-pake-04

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

Copyright Notice

Table of Contents

1. Introduction

DISCLAIMER: This is a work-in-progress draft and has not yet seen significant security analysis. It should not be used as a basis for building production systems.

In some applications, it is desireable 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 the client and server have pre-negotiated a choice of PAKE (and any required parameters) in addition to the password itself. As a first case, this document defines a concrete protocol for executing the SPAKE2+ PAKE protocol [I-D.irtf-cfrg-spake2].

2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

The mechanisms described in this document also apply to DTLS 1.3 [I-D.ietf-tls-dtls13], but for brevity, we will refer only to TLS throughout.

3. Setup

In order to use this protocol, a TLS client and server need to have pre-provisioned the values required to execute the protocol:

o A choice of PAKE protocol

o Any parameters required by the PAKE protocol

o A password (or a derived value as described by the PAKE protocol)

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. We assume that in either case, a single opaque "identity" value is sufficient to identify the required parameters.

4. TLS Extensions

A client offers to authenticate with PAKE by including a "pake" extension in its ClientHello. The content of this exension is a "PAKEClientHello" value, providing a list of identities under which the client can authenticate, and for each identity, the client's first message from the underlying PAKE protocol.

If a client sends the "pake" extension, then it MAY also send the "key_share" and "pre_shared_key" extensions, to allow the server to choose an authentication mode. Unlike PSK-based authentication,

however, authentication with PAKE cannot be combined with the normal TLS ECDH mechanism. Forward secrecy is provided by the PAKE itself.

```
struct {
    opaque identity<0..2^16-1>;
    opaque pake_message<1..2^16-1>;
} PAKEShare;

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

A server that receives a "pake" extension examines the list of client shares to see if there is one with an identity the server recognizes. If so, the server may indicate its choice of PAKE authentication by including a "pake" extension in its ServerHello. The content of this exension is a "PAKEServerHello" value, specifying the identity value for the password the server has selected, and the server's first message in the PAKE protocol.

Use of PAKE authenication is compatible with standard certificate-based authentication of both clients and servers. If a server includes an "pake" extension in its ServerHello, it may still send the Certificate and CertificateVerify messages, and/or send a CertificateRequest message to the client.

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

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

Based on the messages exchanged in the ClientHello and ServerHello, the client and server execute the specified PAKE protocol to derive a shared key. This key is used as the "ECHD(E)" input to the TLS 1.3 key schedule.

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 MUST NOT send application data before it has received the client's Finished message.

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:

- Parameters that must be pre-provisioned
- 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
- How the outputs are of the PAKE protocol are used to populate the "PSK" and "ECDH(E)" inputs to the TLS key schedule.

The underlying cryptographic protocol must be compatible with the message flow described above:

- It must be possible to execute in one round-trip, with the client speaking first
- The Finished MAC must provide sufficient key confirmation for the protocol, taking into account the contents of the handshake messages

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:

- SPAKE2+ (described below) [I-D.irtf-cfrg-spake2]
- SPEKE and derivatives such as Dragonfly [speke] [I-D.harkins-tls-dragonfly]
- OPAQUE [opaque]
- SRP [RFC2945]

6. SPAKE2+ Implementation

7. Pre-provisioned Parameters

In order to use SPAKE2+, a TLS client and server need to have pre-provisioned the values required to execute the SPAKE2+ protocol (see Section 3.1 of [I-D.irtf-cfrg-spake2]):

- o A DH group of order "p*h", with "p" a large prime, and generator "G"
- o Fixed elements "M" and "N" for the group
- o A hash function "H"
- o A password "pw"

Note that the hash function "H" might be different from the hash function associated with the ciphersuite negotiated by the two parties. The hash function "H" MUST be a hash function suitable for hashing passwords, e.g., Argon2 or scrypt [I-D.irtf-cfrg-argon2] [RFC7914].

The TLS client and server roles map to the "A" and "B" roles in the SPAKE2+ specification, respectively. The identity of the server is the domain name sent in the "server_name" extension of the ClientHello message. The identity of the client is an opaque octet string, specified in the "spake2" ClientHello extension, defined below.

From the shared password, each party computes two shared integers "w0" and "w1" by running the following algorithm twice (changing the "context" value each time):

```
struct {
  uint16 context;
  opaque client\_identity<0..255>;
  opaque server\_name<0..255>;
  opaque password<0..255>;
} PasswordInput;
```

- o Encode the following values into a "PasswordInput" structure:
  - * "client_identity": The client's identity, as described above.
  - * "server_name": The server's identity, as described above.
  - * "password": The password "pw"

      * "context": One of the following values:

         + 0x7730, when generating "w0"

         + 0x7731, when generating "w1"

   o Use the hash function "H" with the encoded "PasswordInput" structure as input to derive an "n"-byte string, where "n" is the byte-length of "p".

   o Interpret the "n"-bit string as an integer "w" in network byte order. Return the result "(w % p) * h" of reducing "w" mod p and multiplying it by "h".

Servers MUST store only the value "w0" and the product "L = w1*G", where "G" is the fixed generator of the group. Clients will need to have access to the values "w0" and "w1" directly, but SHOULD generate these values dynamically, rather than caching them.

8. Content of the TLS Extensions

The content of a "pake_message" in a ClientHello is the client's key share "T". The value "T" is computed as specified in [I-D.irtf-cfrg-spake2], as "T = w*M + X", where "M" is a fixed value for the DH group and "X" is the public key of a fresh DH key pair. The format of the key share "T" is the same as for a "KeyShareEntry.key_exchange" value from the same group.

The content of a "pake_message" in a ServerHello is the server's key share "S". The value "S" is computed as specified in [I-D.irtf-cfrg-spake2], as "S = w*N + Y", where "N" is a fixed value for the DH group and "Y" is the public key of a fresh DH key pair. The format of the key share "S" is the same as for a "KeyShareEntry.key_exchange" value from the same group.

Based on these messages, both the client and server can compute the two shared values as specified in [I-D.irtf-cfrg-spake2].

Name	Value	Client	Server
Z	x*y*G	x*(S - w0*N)	x*(T - w0*M)
V	w1*y*G	w1*(S - w0*N)	y*L

The following value is used as the "(EC)DHE" input to the TLS 1.3 key schedule:

```
K = H(Z || V)
```

Here "H" is the hash function corresponding to the TLS cipher suite in use and "||" represents concatenation of octet strings.

9. Security Considerations

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.

The mechanism defined in this document does not provide protection for the client's identity, in contrast to TLS client authentication with certificates.

TLS servers that offer this mechanism can be used by third party attackers as an oracle for two questions:

1. Whether the server knows about a given identity

2. Whether the server recognizes a given (identity, password) pair

The former is signaled by whether the server returns a "pake" extension.

[[TODO: Similar to https://tools.ietf.org/html/rfc5054#section-2.5.1.3, the server could run through a complete handshake calculation and fail at the end so that the attacker only knows that the identity/password pair is incorrect, but does not know if the identity is recognized or not. This requires that the server can interpret the pake_message and ascertain the associated PAKE algorithm, group parameters, etc., which requires a reworking of some text in this draft as the identity is currently defined as providing a map to said group parameters. This is related to the discussion in the Open Items section.]]

The latter is signaled by whether the connection succeeds. These oracles are all-or-nothing: If the attacker does not have the correct identity or password, he does not learn anything about the correct value.

9.1. Security when using SPAKE2+

For the most part, the security properties of the password-based authentication described in this document are the same as those described in the Security Considerations of [I-D.irtf-cfrg-spake2]. The TLS Finished MAC provides the key confirmation required for the security of the protocol. Note that all of the elements covered by

the example confirmation hash listed in that document are also covered by the Finished MAC:

- "A", "B", and "T" are included via the ClientHello
- "S" via the ServerHello
- "K", and "w" via the TLS key schedule

The "x" and "y" values used in the SPAKE2 protocol MUST have the same ephemerality properties as the key shares sent in the "key_shares" extension. In particular, "x" and "y" MUST NOT be equal to zero. This ensures that TLS sessions using SPAKE2 have the same forward secrecy properties as sessions using the normal TLS (EC)DH mechanism.

10. Open Items

10.1. PAKE Algorithm Negotiation

It is possible that a client may know the password to use, but may not know in advance which PAKE protocols(s) a particular server supports. A potential solution to this is similar to TLS1.3 ClientHello "key_share" operation: the client may send an empty "client_shares" vector in its PAKEClientHello extension. The server can then send an HelloRetryRequest indicating which PAKE protocol, and associated group parameters, the client should use. The client then sends another ClientHello that includes "pake_message" in the PAKEClientHello extension calculated using the correct algorithm. This requires definition of a suitable field for transporting PAKE algorithm and group parameters.

As an optimaisation, similar to TLS1.3 key_share operation, the client could guess the PAKE protocol and include a "pake_message" derived from its guess in the initial ClientHello. If the server does not support the selected PAKE protcol (or protocol group parameter, etc.), the server can send an HelloRetryRequest indicating the supported PAKE protocol and group parameters. Note: it is TBD if sending two different "pake_messages" derived from two different protocol and/or group parameters in two different ClientHello messages constitutes a significant attack vector. This needs cryptographic review.

11. 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
TBD	pake	CH, SH	RFC XXXX

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

12. References

12.1. Normative References

[I-D.ietf-tls-dtls13]
Rescorla, E., Tschofenig, H., and N. Modadugu, "The Datagram Transport Layer Security (DTLS) Protocol Version 1.3", draft-ietf-tls-dtls13-28 (work in progress), July 2018.

[I-D.irtf-cfrg-spake2]
Ladd, W. and B. Kaduk, "SPAKE2, a PAKE", draft-irtf-cfrg-spake2-05 (work in progress), February 2018.

[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/info/rfc2119>.

12.2. Informative References

[I-D.harkins-tls-dragonfly]
Harkins, D., "Secure Password Ciphersuites for Transport Layer Security (TLS)", draft-harkins-tls-dragonfly-03 (work in progress), July 2018.

[I-D.irtf-cfrg-argon2]
Biryukov, A., Dinu, D., Khovratovich, D., and S. Josefsson, "The memory-hard Argon2 password hash and proof-of-work function", draft-irtf-cfrg-argon2-03 (work in progress), August 2017.

[opaque] Xu, J., "OPAQUE: An Asymmetric PAKE Protocol Secure Against Pre-Computation Attacks", 2018.

[RFC2945] Wu, T., "The SRP Authentication and Key Exchange System", RFC 2945, DOI 10.17487/RFC2945, September 2000, <https://www.rfc-editor.org/info/rfc2945>.

[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/info/rfc5054>.

[RFC7914] Percival, C. and S. Josefsson, "The scrypt Password-Based Key Derivation Function", RFC 7914, DOI 10.17487/RFC7914, August 2016, <https://www.rfc-editor.org/info/rfc7914>.

[speke] Jablon, D., "Extended Password Key Exchange Protocols Immune to Dictionary Attacks", 1997.

Authors' Addresses

Richard Barnes
Cisco

Email: rlb@ipv.sx

Owen Friel
Cisco

Email: ofriel@cisco.com