Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: 26 August 2021

N. Sullivan
Cloudflare
H. Krawczyk
IBM Research
O. Friel
R. Barnes
Cisco

22 February 2021

OPAQUE with TLS 1.3
draft-sullivan-tls-opaque-01

Abstract

This document describes two mechanisms for enabling the use of the OPAQUE password-authenticated key exchange in TLS 1.3.

Table of Contents

1. Introduction

Note that this draft has not received significant security review and should not be the basis for production systems.

OPAQUE [opaque-paper] is a mutual authentication method that enables the establishment of an authenticated cryptographic key between a client and server based on a user's password, without ever exposing the password to servers or other entities other than the client machine and without relying on a Public Key Infrastructure (PKI). OPAQUE leverages a primitive called a Strong symmetric Password Authenticated Key Exchange (Strong aPAKE) to provide desirable properties including resistance to pre-computation attacks in the event of a server compromise.

In some cases, it is desirable to combine password-based authentication with traditional PKI-based authentication as a defense-in-depth measure. For example, in the case of IoT devices, it may be useful to validate that both parties were issued a certificate from a certain manufacturer. Another desirable property for password-based authentication systems is the ability to hide the client's identity from the network. This document describes the use of OPAQUE in TLS 1.3 [TLS13] both as part of the TLS handshake and post-handshake facilitated by Exported Authenticators [I-D.ietf-tls-exported-authenticator], how the different approaches

satisfy the above properties and the trade-offs associated with each design.

The in-handshake instantiations of OPAQUE can be used to authenticate a TLS handshake with a password alone, or in conjunction with certificate-based (mutual) authentication but does not provide identity hiding for the client. The Exported Authenticators instantiation of OPAQUE provides client identity hiding by default and allows the application to do password authentication at any time during the connection, but requires PKI authentication for the initial handshake and application-layer semantics to be defined for transporting authentication messages.

2. Conventions and Definitions

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.

3. OPAQUE

OPAQUE [opaque-paper] is a Strong Asymmetric Password-Authenticated Key Exchange (Strong aPAKE) built on an oblivious pseudo-random function (OPRF) and authenticated key exchange protocol that is secure against key-compromise impersonation (KCI) attacks. Unlike previous PAKE methods such as SRP [RFC2945] and SPAKE-2 [I-D.irtf-cfrg-spake2], which require a public salt value, a Strong aPAKE leverages the OPRF private key as salt, making it resistant to pre-computation attacks on the password database stored on the server.

TLS 1.3 provides a KCI-secure key agreement algorithm suitable for use with OPAQUE. This document describes two instantiations of OPAQUE in TLS 1.3: one based on digital signatures, called OPAQUE-Sign, and one on Diffie-Hellman key agreement, called OPAQUE-KEX.

OPAQUE consists of two distinct phases: password registration and authentication. We will describe the mechanisms for password registration in this document but it is assumed to have been done outside of a TLS connection. During password registration, the client and server establish a shared set of parameters for future authentication and two private-public key pairs are generated, one for the client and one for the server. The server keeps its private key and stores an encapsulated copy of the client's key pair along with its own public key in an "envelope" that is encrypted with the result of the OPRF operation. Note that it is possible for the

server to use the same key for multiple clients. It may be necessary to permit multiple simultaneous server keys in the even of a key rollover. The client does not store any state nor any PKI information.

In OPAQUE-Sign, the key pairs generated at password registration time are digital signature keys. These signature keys are used in place of certificate keys for both server and client authentication in a TLS handshake. Client authentication is technically optional, though in practice is almost universally required. OPAQUE-Sign cannot be used alongside certificate-based handshake authentication. This instantiation can also be leveraged to do part of a post-handshake authentication using Exported Authenticators [I-D.ietf-tls-exported-authenticator] given an established TLS connection protected with certificate-based authentication.

In OPAQUE-KEX, the key pairs are Diffie-Hellman keys and are used to establish a shared secret that is fed into the key schedule for the handshake. The handshake continues to use Certificate-based authentication and establishes the shared key using Diffie-Hellman. This instantiations is best suited to use cases in which both password and certificate-based authentication are needed during the initial handshake, which is useful in some scenarios. There is no unilateral authentication in this context, mutual authentication is demonstrated explicitly through the finished messages.

4. Password Registration

Password registration is run between a client U and a server S. It is assumed that U can authenticate S during this registration phase (this is the only part in OPAQUE that requires some form of authenticated channel, either physical, out-of-band, PKI-based, etc.) During this phase, clients run the registration flow in [I-D.irtf-cfrg-opaque] using a specific OPAQUE configuration consisting of a tuple (OPRF, Hash, MHF, AKE). The specific AKE is not used during registration. It is only used during login.

During this phase, a specific OPAQUE configuration is chosen, which consists of a tuple (OPRF, Hash, MHF, AKE). See [I-D.irtf-cfrg-opaque] for details about configuration parameters. In this context, AKE is either OPAQUE-Sign or OPAQUE-KEX.

5. Password Authentication

Password authentication integrates TLS into OPAQUE in such a way that clients prove knowledge of a password to servers. In this section, we describe TLS extensions that support this integration for both OPAQUE-KEX and OPAQUE-Sign.

5.1. TLS Extensions

We define several TLS extensions to signal support for OPAQUE and transport the parameters. The extensions used here have a similar structure to those described in Usage of PAKE with TLS 1.3 [I-D.barnes-tls-pake]. The permitted messages that these extensions are allowed and the expected protocol flows are described below.

First, this document specifies extensions used to convey OPAQUE client and server messages, called "opaque_client_auth" and "opaque_server_auth" respectively.

```
enum {
  ...
  opaque_client_auth(TBD),
  opaque_server_auth(TBD),
  (65535)
} ExtensionType;
```

The "opaque_client_auth" extension contains a "PAKEClientAuthExtension" struct and can only be included in the "CertificateRequest" and "Certificate" messages.

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

The "opaque_server_auth" extension contains a "PAKEServerAuthExtension" struct and can only be included in the "ClientHello", "EncryptedExtensions", "CertificateRequest" and "Certificate" messages, depending on the type.

```
struct {
  opaque idU<0..2^16-1>;
  CredentialRequest request;
} PAKEShareClient;

struct {
  opaque idS<0..2^16-1>;
  CredentialResponse response;
} PAKEShareServer;

struct {
  select (Handshake.msg_type) {
    ClientHello:
      PAKEShareClient client_shares<0..2^16-1>;
      OPAQUEType types<0..2^16-1>;
    EncryptedExtensions, Certificate:
      PAKEShareServer server_share;
      OPAQUEType type;
  }
} PAKEServerAuthExtension;
```

This document also defines the following set of types;

```
enum {
  OPAQUE-Sign(1),
  OPAQUE-KEX(2),
} OPAQUEType;
```

Servers use PAKEShareClient.idU to index the user's record on the server and create the PAKEShareServer.response. The types field indicates the set of supported auth types by the client. PAKEShareClient.request and PAKEShareServer.response, of type CredentialRequest and CredentialResponse, respectively, are defined in [I-D.irtf-cfrg-opaque].

This document also describes a new CertificateEntry structure that corresponds to an authentication via a signature derived using OPAQUE. This structure serves as a placeholder for the PAKEServerAuthExtension extension.

```
struct {
  select (certificate_type) {
    case OPAQUESign:
      /* Defined in this document */
      opaque null<0>

    case RawPublicKey:
      /* From RFC 7250 ASN.1_subjectPublicKeyInfo */
      opaque ASN1_subjectPublicKeyInfo<1..2^24-1>;

    case X509:
      opaque cert_data<1..2^24-1>;
  };
  Extension extensions<0..2^16-1>;
} CertificateEntry;
```

We request that IANA add an additional type to the "TLS Certificate Types" registry for this OPAQUESign.

Support for the OPAQUESign Certificate type for server authentication can be negotiated using the server_certificate_type [RFC7250] and the Certificate type for client authentication can be negotiated using the client_certificate_type extension [RFC7250].

Note that there needs to be a change to the client_certificate_type row in the IANA "TLS ExtensionType Values" table to allow client_certificate_type extension to be used as an extension to the CertificateRequest message.

6. Use of extensions in TLS handshake flows

6.1. OPAQUE-KEX

In this mode, OPAQUE private keys are used for key agreement algorithm and the result is fed into the TLS key schedule. Password validation is confirmed by the validation of the finished message. These modes can be used in conjunction with optional Certificate-based authentication.

It should be noted that since the identity of the client it is not encrypted as it is sent as an extension to the ClientHello. This may present a privacy problem unless a mechanism like Encrypted Client Hello [ECH] is created to protect it.

Upon receiving a PAKEServerAuth extension, the server checks to see if it has a matching record for this identity. If the record does not exist, the handshake is aborted with a "illegal_parameter" alert. If the record does exist, but the key type of the record does not

match any of the supported_groups sent in the key_share extension of the ClientHello, an HRR is sent containing the set of valid key types that it found records for.

Given a matching key_share and an identity with a matching supported_group, the server returns its PAKEServerAuth as an extension to its EncryptedExtensions. Both parties then derive a shared OPAQUE key as follows:

```
 U computes
   K = H(g^y ^ PrivU || PubU ^ x || PubS ^ PrivU || IdU || IdS )
 S computes
   K = H(g^x ^ PrivS || PubS ^ y || PubU ^ PrivS || IdU || IdS )
```

IdU, IdS represent the identities of user (sent as identity in PAKEShareClient) and server (Certificate message). H is the HKDF function agreed upon in the TLS handshake.

The result, K, is then added as an input to the Master Secret in place of the 0 value defined in TLS 1.3. Specifically,

```
  0 -> HKDF-Extract = Master Secret
```

becomes

```
  K -> HKDF-Extract = Master Secret
```

In this construction, the finished messages cannot be validated unless the OPAQUE computation was done correctly on both sides, authenticating both client and server.

6.2. OPAQUE-Sign

In this modes of operation, the OPAQUE private keys are used for digital signatures and are used to define a new Certificate type and CertificateVerify algorithm. Like the OPAQUE-KEX instantiations above, the identity of the client is sent in the clear in the client's first flight unless a mechanism like Encrypted Client Hello [ECH] is created to protect it.

Upon receiving a PAKEServerAuth extension, the server checks to see if it has a matching record for this identity. If the record does not exist, the handshake is aborted with a TBD error message. If the record does exist, but the key type of the record does not match any of the supported_signatures sent in the the ClientHello, the handshake must be aborted with a "illegal_parameter" error.

We define a new Certificate message type for an OPAQUE-Sign authenticated handshake.

```
enum {
  X509(0),
  RawPublicKey(2),
  OPAQUE-Sign(3),
  (255)
} CertificateType;
```

Certificates of this type have CertificateEntry structs of the form:

```
struct {
  Extension extensions<0..2^16-1>;
} CertificateEntry;
```

Given a matching signature_scheme and an identity with a matching key type, the server returns a certificate message with type OPAQUE-Sign with PAKEServerAuth as an extension. The private key used in the CertificateVerify message is set to the private key used during account registration, and the client verifies it using the server public key contained in the client's envelope.

It is RECOMMENDED that the server includes a CertificateRequest message with a PAKEClientAuth and the identity originally sent in the PAKEServerAuth extension from the client hello. On receiving a CertificateRequest message with a PAKEClientAuth extension, the client returns a CertificateVerify message signed by PrivC which is validated by the server using PubC.

7. Integration into Exported Authenticators

Neither of the above mechanisms provides privacy for the user during the authentication phase, as the user id is sent in the clear. Additionally, OPAQUE-Sign has the drawback that it cannot be used in conjunction with certificate-based authentication.

It is possible to address both the privacy concerns and the requirement for certificate-based authentication by using OPAQUE-Sign in an Exported Authenticator [I-D.ietf-tls-exported-authenticator] flow, since exported authenticators are sent over a secure channel that is typically established with certificate-based authentication. Using Exported Authenticators for OPAQUE has the additional benefit that it can be triggered at any time after a TLS session has been established, which better fits modern web-based authentication mechanism.

The ClientHello contains PAKEServerAuth, PAKEClientAuth with empty identity values to indicate support for these mechanisms.

1. Client creates Authenticator Request with CR extension PAKEServerAuth.

2. Server creates Exported Authenticator with OPAQUE-Sign (PAKEServerAuth) and CertificateVerify (signed with the OPAQUE private key).

If the server would like to then establish mutual authentication, it can do the following:

1. Server creates Authenticator Request with CH extension PAKEClientAuth (identity)

2. Client creates Exported Authenticator with OPAQUE-Sign Certificate and CertificateVerify (signed with user private key derived from the envelope).

Support for Exported Authenticators is negotiated at the application layer.

8. Summary of properties

Variant \ Property	Identity hiding	Certificate auth	Server-only auth	Post-handshake auth	Minimum round trips
OPAQUE-Sign with EA	yes	yes	yes	yes	2-RTT
OPAQUE-Sign	no	no	yes	no	1-RTT
OPAQUE-KEX	no	no	no	no	1-RTT

Table 1

9. Privacy considerations

TBD: cleartext identity, etc

10. Security Considerations

TODO: protecting against user enumeration

11. IANA Considerations

* Existing IANA references have not been updated yet to point to this document.

  IANA is asked to register a new value in the "TLS Certificate Types" registry of Transport Layer Security (TLS) Extensions (TLS-Certificate-Types-Registry), as follows:

* Value: 4 Description: OPAQUE Authentication Reference: This RFC

Correction request: The client_certificate_type row in the IANA TLS ExtensionType Values table to allow client_certificate_type extension to be used as an extension to the CertificateRequest message.

12. References

12.1. Normative References

[I-D.ietf-tls-exported-authenticator]
Sullivan, N., "Exported Authenticators in TLS", Work in Progress, Internet-Draft, draft-ietf-tls-exported-authenticator-14, 25 January 2021, <https://www.ietf.org/archive/id/draft-ietf-tls-exported-authenticator-14.txt>.

[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>.

[RFC7250] Wouters, P., Ed., Tschofenig, H., Ed., Gilmore, J., Weiler, S., and T. Kivinen, "Using Raw Public Keys in Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)", RFC 7250, DOI 10.17487/RFC7250, June 2014, <https://www.rfc-editor.org/info/rfc7250>.

[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/info/rfc8174>.

[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/info/rfc8446>.

12.2. 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-09, 16 December 2020, <https://www.ietf.org/archive/id/draft-ietf-tls-esni-09.txt>.

[I-D.barnes-tls-pake]
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://www.ietf.org/archive/id/draft-barnes-tls-pake-04.txt>.

[I-D.irtf-cfrg-opaque]
Krawczyk, H., Lewi, K., and C. A. Wood, "The OPAQUE Asymmetric PAKE Protocol", Work in Progress, Internet-Draft, draft-irtf-cfrg-opaque-03, 21 February 2021, <https://www.ietf.org/archive/id/draft-irtf-cfrg-opaque-03.txt>.

[I-D.irtf-cfrg-spake2]
Ladd, W. and B. Kaduk, "SPAKE2, a PAKE", Work in Progress, Internet-Draft, draft-irtf-cfrg-spake2-18, 17 January 2021, <https://www.ietf.org/archive/id/draft-irtf-cfrg-spake2-18.txt>.

[opaque-paper]
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>.

[RFC5869] Krawczyk, H. and P. Eronen, "HMAC-based Extract-and-Expand Key Derivation Function (HKDF)", RFC 5869, DOI 10.17487/RFC5869, May 2010, <https://www.rfc-editor.org/info/rfc5869>.

Appendix A. Acknowledgments

Authors' Addresses

Nick Sullivan
Cloudflare

Email: nick@cloudflare.com

Hugo Krawczyk
IBM Research

Email: hugo@ee.technion.ac.il

Owen Friel
Cisco

Email: ofriel@cisco.com

Richard Barnes
Cisco

Email: rlb@ipv.sx