Workgroup: WG Working Group
Internet-Draft:
draft-reitzenstein-kitten-opaque-00
Published: 6 October 2022
Intended Status: Informational
Expires: 9 April 2023
Authors: N. von Reitzenstein Čerpnjak

A SASL and GSS-API Mechanism family using the asymmetric password-authenticated key agreement OPAQUE

Abstract

This specification describes a family of Simple Authentication and Security Layer (SASL, RFC4422) authentication mechanisms based on the OPAQUE asymmetric password-authenticated key agreement (PAKE) algorithm.

They offer two distinct advantages over the SCRAM family of mechanisms. The underlying OPAQUE algorithm provides the ability for clients to register without the servers getting access to the clear text password of an user, preventing password exfiltration at registration. Secondly a successful authentication produces a long-term secret key only known to the client that can be used to access encrypted server-side data without needing to share keys between clients via a side-band mechanism.

When used in combination with TLS or an equivalent security layer these mechanisms allow for secure channel binding.

Discussion Venues

This note is to be removed before publishing as an RFC.

Discussion of this document takes place on the Common Authentication Technology Next Generation Working Group mailing list (kitten@ietf.org), which is archived at https://mailarchive.ietf.org/arch/browse/kitten/.

Source for this draft and an issue tracker can be found at https://github.com/dequbed/draft-reitzenstein-auth-opaque.

Status of This Memo

Copyright Notice

Table of Contents

1. 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.

2. Introduction

This specification describes a family of authentication mechanisms called OPAQUE, based on the asymmetric PAKE of the same name. The mechanisms provide strong mutual authentication and allow binding the authentication to an pre-existing underlying encrypted transport.

OPAQUE as specified in this document is a Simple Authentication and Security Layer (SASL) mechanism compatible to the bridge between SASL and the Generic Security Services Application Programming Interface (GSS-API) called "GS2" [RFC5801]. This means that the mechanism can be used as either a SASL mechanism or a GSS-API mechanism.

The OPAQUE algorithm provides the following features which this mechanism makes use of: * The authentication information stored in an authentication database on the server is not sufficient to impersonate the client. It is additionally salted and bound to a private key of the server, making pre-stored dictionary attack impossible. * Successfull authentication does not grant the server enough information to impersonate the client. * Mutual authentication is implicit and required. A successfull authentication always strongly authenticates both sides of the exchange. * A successfull authentication provides both parties with an emphemeral shared secret. This secret has high entropy and can be used to establish a trusted encrypted channel without deriving trust from a 3rd party. * A successfull authentication additionally provides the client with a constant secret. This secret is only known to the client and the same for every authentication. It can be used to e.g. store encrypted data on the server without having to manage keys locally.

3. OPAQUE Algorithm Overview

The Authenticated Key Exchange defined by OPAQUE consists of three messages -- KE1, KE2 and KE3 -- send by the client (KE1, KE3) and server (KE2) respectively. A client knows the outcome of the authentication after receiving KE2, the server after receiving KE3.

The following is a description of a full SASL OPAQUE authentication exchange. Nothing in OPAQUE prevents sending the first client response with the SASL authentication request as defined by an application protocol ("initial client response"). See [RFC4422] for more details.

The OPAQUE client starts by being in posession of an username and password. It uses the password to generate a KE1 structure as per OPAQUE [I-D.irtf-cfrg-opaque], and sends it and the username to the server. The server retrieves the corresponding authentication information, i.e. registration record, OPRF seed, server private key, and the key-stretching function (KSF) parameters used at registration. It uses the first three to generate a KE2 structure as per OPAQUE [I-D.irtf-cfrg-opaque] and sends that, channel binding data (if any) and the KSF parameters to the client.

The client authenticates the server using KE2 and the KSF parameters, also showing the integrity of the channel binding data in the process, and generates a final KE3 it can return to the server.

The three messages KE1, KE2 and KE3 are generated using the following functions specified in OPAQUE [I-D.irtf-cfrg-opaque]:

```
KE1 := ClientInit(password)

KE2 := ServerInit(server_identity, server_private_key, server_public_key

KE3 := ClientFinish(client_identity, server_identity, KE2)
```

The values of client_identity and server_identity are set to:

```
client_identity := client-first-message + "," + client_public_key

server_identity := server-message-bare + "," + server_public_key
```

With the values and encodings of the remaining parameters per the OPAQUE specification, and + indicating concatenation.

Upon receipt of KE3 the server can validate the authentication exchange including integrity of the channel binding data it sent previously, and extract a session key that strongly authenticates the client to the server.

4. OPAQUE Mechanism Names

An OPAQUE mechanism name is the string "OPAQUE-" followed by an uppercase identifier for the cryptographic primitives being used. The identifier is limited to 7 octets (20 - len("OPAQUE-") - len("-PLUS")) which is too short to contain the full names of all cryptographic primitives used. Thus OPAQUE mechanisms using new groups of primitives **SHOULD** be registered with IANA to allow implementers to identify all required primitives.

The PLUS suffix is only used when the authenticating parties support channel binding. If the server supports channel binding it **SHOULD** advertise both the "bare" and "plus" version of whichever OPAQUE variant it support. If the server does not it will only advertise the "bare" version.

5. OPAQUE Authentication Exchange

```
C: n,,n=user,r=<ke1>
S: c=<cbdata>,i=<params>,v=<ke2>
C: p=<ke3>
```

First, the client sends the "client-first-message" containing:

* A GS2 header consisting of a flag indicating channel binding support and usage, and an optional SASL authorization identity.

* The authentication ID (AuthID) of the user.

* OPAQUE KE1, containing the OPRF credential request, a nonce, and an ephemeral public key.

In response the server sends the "server-message" containing:

* An encoding of requested channel binding data

* Parameters for the KSF that needs to be used by the client

* OPAQUE KE2, containting the OPRF credential response, a nonce, and an ephemeral public key.

* A MAC proving the integrity of the exchange so far and cryptographically authenticating the server to the client (also contained in KE2)

The client then recovers a long term private key and client-only export key from the OPRF response using the defined KSF with the user-provided password and parameters sent by the server.

To finalize the authentication a client sends a "client-final-message" containing itself a MAC over the exchange (in KE3), thus cryptographically authenticating the client to the server.

5.1. OPAQUE Attributes

This section details all attributes permissible in messages, their use and their value format. All Attributes a single US-ASCII letters and case-sensitive. The selection of letters used for attributes is based on SCRAM [RFC5802] to make it easier to adapt extensions defined for SCRAM to this mechanism.

Note that similar to SCRAM the order of attributes is fixed for all messages, except for extension attributes which are limited to designated positions but may appear in any order. Implementations **MUST NOT** assume a specific ordering of extensions.

* a: This is an optional attribute and is part of the GS2 [RFC5801] bridge between GSS-API and SASL. Its specification and usage is the same as defined in [RFC5802], Section 5.1.

* n: This attribute specifies the name of the user whose password is used for authentication (aka "authentication identity" [RFC4422]). Its encoding, preparation, and usage is the same as defined in [RFC5802], Section 5.1.

* m: This attribute is reserved for future extensibility. In this version of OPAQUE its presence in a client or server message **MUST** cause authentication failure when the attribute is parsed by the other end.

* r: This attribute specifies a base64-encoded serialization of the KE1 message as specified by [OPAQUE].

* c: This **REQUIRED** attribute specifies the base64-encoded GS2 header and channel binding data. Its specification is the same as defined in [RFC5802], Section 5.1, however it is sent by the server to the client instead of the other way around as in SCRAM.

* i: This attribute specifies base64-encoded parameters for the KSF to be used. The format of the parameters is specific to the KSF in use.

* v: This attribute specifies a base64-encoded serialization of the KE2 message as specified by [OPAQUE].

* p: This attribute specifies a base64-encoded serialization of the KE3 message as specified by [OPAQUE].

* Further as of now unspecified mandatory and optional extensions. Mandatory extensions are encoded using the "m" attribute, optional attributes may use any unassigned attribute name. Unknown optional attributes **MUST** be ignored upon receipt.

5.2. SASL Mechanism Requirements

This section describes the required information for SASL mechanisms as laid out in [RFC4422], Section 5.

1) "OPAQUE-A255SHA" and "OPAQUE-A255SHA-PLUS"

2a) OPAQUE is a client-first mechanism

2b) OPAQUE does not send any additional data to indicate a successful outcome. All authentication exchanges take 3 messages regardless of success.

3) OPAQUE can transfer authorization identities from the client to the server.

4) OPAQUE does not offer security layers but allows channel binding.

5) OPAQUE uses a MAC to protect the integrity of the entire authentication exchange including the authzid.

6. Channel Binding

OPAQUE supports binding the authentication to an underlying secure transport. Support for channel binding is optional, therefore the usage of channel binding is negotiable.

The negotiation of channel binding is performed as defined in [RFC5802], Section 6 with the following differences:

* The non-PLUS and PLUS variants of the mechanism are instead named OPAQUE-<variant> and OPAQUE-<variant>-PLUS respectively.

* As it is the server who sends the channel binding data the client is responsible to verify this data by constructing the expected value of the "c=" attribute and comparing it to the received one. This comparison **SHOULD** be implemented to be constant-time.

6.1. Default Channel Binding

'tls-exporter' is the default channel binding type for any application that do not specify one.

Servers **MUST** implement the 'tls-exporter' [RFC9266] channel binding type if they implement any channel binding and use TLS. Clients **SHOULD** implement the 'tls-exporter' [RFC9266] channel binding type if they implement any and use TLS.

Servers **MUST** use the channel binding type indicated by the client, or fail authentication if they do not support it.

7. OPAQUE-A255SHA(-PLUS)

TODO: Define one set of primitives; probably OPAQUE-A255SHA(-PLUS), using HKDF, HMAC, ristretto255, SHA-512 and Argon2i(d?)

8. Formal Syntax

The following syntax specification is written in Augmented Backus-Naur Form (ABNF) notation as specified in [RFC5234]. The non-terminals "UTF8-2", "UTF8-3" and "UTF8-4" are defined in [RFC3629].

The syntax is based in large parts on [RFC5802], Section 7, which may be referenced for clarification. If this specification and [RFC5802] are in conflict, this speification takes priority.

Used definitions from [RFC5802] are reproduced here for convenience:

```
ALPHA = <as defined in RFC 5234 appendix B.1>
DIGIT = <as defined in RFC 5234 appendix B.1>
UTF8-2 = <as defined in RFC 3629 (STD 63)>
UTF8-3 = <as defined in RFC 3629 (STD 63)>
UTF8-4 = <as defined in RFC 3629 (STD 63)>

attr-val        = ALPHA "=" value
                 ;; Generic syntax of any attribute sent
                 ;; by server or client

value           = 1*value-char

value-safe-char = %x01-2B / %x2D-3C / %x3E-7F /
                 UTF8-2 / UTF8-3 / UTF8-4
                 ;; UTF8-char except NUL, "=", and ",".

value-char      = value-safe-char / "="

printable       = %x21-2B / %x2D-7E
                 ;; Printable ASCII except ",".
                 ;; Note that any "printable" is also
                 ;; a valid "value".

base64-char     = ALPHA / DIGIT / "/" / "+"

base64-4        = 4base64-char

base64-3        = 3base64-char "="

base64-2        = 2base64-char "=="

base64          = *base64-4 [base64-3 / base64-2]

posit-number = %x31-39 *DIGIT
                 ;; A positive number.

saslname        = 1*(value-safe-char / "=2C" / "=3D")
                 ;; Conforms to <value>.

authzid         = "a=" saslname
                 ;; Protocol specific.

cb-name         = 1*(ALPHA / DIGIT / "." / "-")
                  ;; See RFC 5056, Section 7.
                  ;; E.g., "tls-server-end-point" or
                  ;; "tls-unique".

gs2-cbind-flag  = ("p=" cb-name) / "n" / "y"
                  ;; "n" -> client doesn't support channel binding.
                  ;; "y" -> client does support channel binding
```

```
                  ;;        but thinks the server does not.
                  ;; "p" -> client requires channel binding.
                  ;; The selected channel binding follows "p=".

gs2-header      = gs2-cbind-flag "," [ authzid ] ","
                  ;; GS2 header for OPAQUE

username        = "n=" saslname
                  ;; Usernames are prepared using SASLprep.

reserved-mext  = "m=" 1*(value-char)
                  ;; Reserved for signaling mandatory extensions.
                  ;; The exact syntax will be defined in
                  ;; the future.

channel-binding = "c=" base64
                  ;; base64 encoding of cbind-input.

cbind-data    = 1*OCTET

cbind-input   = gs2-header [ cbind-data ]
                  ;; cbind-data MUST be present for
                  ;; gs2-cbind-flag of "p" and MUST be absent
                  ;; for "y" or "n".
```

The following definitions are specific to OPAQUE:

```
client-first-message-bare =
            [reserved-mext ","] username "," auth-request
            ["," extensions]

client-first-message = gs2-header client-first-message-bare

validator = "v=" base64

server-message-bare =
            [reserved-mext ","] channel-binding "," ksf-params ","
            credentials-response ["," extensions]

server-message = server-message-bare "," validator

client-final-message = "p=" base64
```

9. Security Considerations

The KSF parameters and channel bindings aren't authenticated before KSF usage, allowing a DoS of a client by an malicious actor posing as the server.

If not used with a secure channel providing confidentiality this mechanism leaks the authid and authzid of an authenticating user to any passive observer.

The cryptographic security of this mechanism is not increased over the one provided by the underlying OPAQUE algorithm, so all security considerations applying to that specification also apply to this one.

10. Open Issues

* OPAQUE allows for a static context to be bound into generated keys to prevent cross-protocol and downgrade attacks. We should probably set that to the mechanism name (e.g. 'OPAQUE-A255SHA')

* With the current design the KSF parameters can not be MAC-verified until after they have been used. This is bad. The only other option is using the ephemeral keypair to generate a MAC key and use that. This may impact security.

* This mechanism should be extended to also become a GSS-API mechanism like SCRAM is.

11. IANA Considerations

The IANA is requested to add the following family of SASL mechanisms to the SASL Mechanism registry established by [RFC4422]:

To: iana@iana.org Subject: Registration of new SASL family OPAQUE

SASL mechanism name (or prefix for the family): OPAQUE- Security Considerations: See this document Published Specification: See this document For futher information: Contact the authors of this document. Owner/Change controller: the IETF Note: None

12. Normative References

[I-D.irtf-cfrg-opaque] "The OPAQUE Asymmetric PAKE Protocol", n.d., <https://github.com/cfrg/draft-irtf-cfrg-opaque>.

[OPAQUE] Bourdrez, D., Krawczyk, H., Lewi, K., and C. A. Wood, "The OPAQUE Asymmetric PAKE Protocol", Work in Progress, Internet-Draft, draft-irtf-cfrg-opaque-09, 6 July 2022, <https://datatracker.ietf.org/doc/html/draft-irtf-cfrg-opaque-09>.

[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>.

[RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, DOI 10.17487/RFC3629, November 2003, <https://www.rfc-editor.org/rfc/rfc3629>.

[RFC4422] Melnikov, A., Ed. and K. Zeilenga, Ed., "Simple Authentication and Security Layer (SASL)", RFC 4422, DOI 10.17487/RFC4422, June 2006, <https://www.rfc-editor.org/rfc/rfc4422>.

[RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/RFC5234, January 2008, <https://www.rfc-editor.org/rfc/rfc5234>.

[RFC5801] Josefsson, S. and N. Williams, "Using Generic Security Service Application Program Interface (GSS-API) Mechanisms in Simple Authentication and Security Layer (SASL): The GS2 Mechanism Family", RFC 5801, DOI

10.17487/RFC5801, July 2010, <https://www.rfc-editor.org/rfc/rfc5801>.

[RFC5802] Newman, C., Menon-Sen, A., Melnikov, A., and N. Williams, "Salted Challenge Response Authentication Mechanism (SCRAM) SASL and GSS-API Mechanisms", RFC 5802, DOI 10.17487/RFC5802, July 2010, <https://www.rfc-editor.org/rfc/rfc5802>.

[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>.

[RFC9266] Whited, S., "Channel Bindings for TLS 1.3", RFC 9266, DOI 10.17487/RFC9266, July 2022, <https://www.rfc-editor.org/rfc/rfc9266>.

Acknowledgments

Thank you to Daniel Bourdrez, Hugo Krawczyk, Kevin Lewi, and C. A. Wood for their work on the OPAQUE PAKE that this mechanism is based on. Thank you to Abhijit Menon-Sen, Alexey Melnikov, Nicolas Williams, and Chris Newman for their work on the SCRAM RFC, most of which this draft oh so blatanly steals for its own gain.

Author's Address

Nadja von Reitzenstein Čerpnjak

Email: me@dequbed.space