Workgroup: Network Working Group
Internet-Draft: draft-irtf-cfrg-opaque-06
Published: 12 July 2021
Intended Status: Informational
Expires: 13 January 2022
Authors: H. Krawczyk
Algorand Foundation

D. Bourdrez
Novi Research

K. Lewi
Novi Research

C.A. Wood
Cloudflare

The OPAQUE Asymmetric PAKE Protocol

Abstract

This document describes the OPAQUE protocol, a secure asymmetric password-authenticated key exchange (aPAKE) that supports mutual authentication in a client-server setting without reliance on PKI and with security against pre-computation attacks upon server compromise. In addition, the protocol provides forward secrecy and the ability to hide the password from the server, even during password registration. This document specifies the core OPAQUE protocol and one instantiation based on 3DH.

Discussion Venues

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

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

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

Password authentication is ubiquitous in many applications. In a common implementation, a client authenticates to a server by sending its client ID and password to the server over a secure connection. This makes the password vulnerable to server mishandling, including accidentally logging the password or storing it in plaintext in a database. Server compromise resulting in access to these plaintext passwords is not an uncommon security incident, even among security-conscious companies. Moreover, plaintext password authentication over secure channels like TLS is also vulnerable to cases where TLS may fail, including PKI attacks, certificate mishandling, termination outside the security perimeter, visibility to middleboxes, and more.

Asymmetric (or Augmented) Password Authenticated Key Exchange (aPAKE) protocols are designed to provide password authentication and mutually authenticated key exchange in a client-server setting without relying on PKI (except during client/password registration) and without disclosing passwords to servers or other entities other than the client machine. A secure aPAKE should provide the best possible security for a password protocol. Namely, it should only be open to inevitable attacks, such as online impersonation attempts with guessed client passwords and offline dictionary attacks upon the compromise of a server and leakage of its credential file. In the latter case, the attacker learns a mapping of a client's password under a one-way function and uses such a mapping to validate potential guesses for the password. Crucially important is for the password protocol to use an unpredictable one-way mapping. Otherwise, the attacker can pre-compute a deterministic list of mapped passwords leading to almost instantaneous leakage of passwords upon server compromise.

Despite the existence of multiple designs for (PKI-free) aPAKE protocols, none of these protocols are secure against pre-computation attacks. In particular, none of these protocols can use the standard technique against pre-computation that combines *secret* random values ("salt") into the one-way password mappings. Either these protocols do not use a salt at all or, if they do, they transmit the salt from server to client in the clear, hence losing the secrecy of the salt and its defense against pre-computation. Furthermore, transmitting the salt may require additional protocol messages.

This document describes OPAQUE, a PKI-free secure aPAKE that is secure against pre-computation attacks. OPAQUE provides forward secrecy (essential for protecting past communications in case of password leakage) and the ability to hide the password from the server, even during password registration. Furthermore, OPAQUE enjoys good performance and an array of additional features including the ability to increase the difficulty of offline dictionary attacks via iterated hashing or other hardening schemes, and offloading these operations to the client (that also helps against online guessing attacks); extensibility of the protocol to support storage and retrieval of client secrets solely based on a password; being amenable to a multi-server distributed implementation where offline dictionary attacks are not possible without breaking into a threshold of servers (such a distributed solution requires no change or awareness on the client-side relative to a single-server implementation).

OPAQUE is defined and proven as the composition of two functionalities: an oblivious pseudorandom function (OPRF) and an authenticated key exchange (AKE) protocol. It can be seen as a

"compiler" for transforming any suitable AKE protocol into a secure aPAKE protocol. (See Section 10 for requirements of the OPRF and AKE protocols.) This document specifies one OPAQUE instantiation based on 3DH [SIGNAL]. Other instantiations are possible, as discussed in Appendix B, but their details are out of scope for this document. In general, the modularity of OPAQUE's design makes it easy to integrate with additional AKE protocols, e.g., TLS, and with future ones such as those based on post-quantum techniques.

OPAQUE consists of two stages: registration and authenticated key exchange. In the first stage, a client registers its password with the server and stores information used to recover authentication credentials on the server. Recovering these credentials can only be done with knowledge of the client password. In the second stage, a client uses its password to recover those credentials and subsequently uses them as input to an AKE protocol.

Currently, the most widely deployed PKI-free aPAKE is SRP [RFC2945], which is vulnerable to pre-computation attacks, lacks proof of security, and is less efficient relative to OPAQUE. Moreover, SRP requires a ring as it mixes addition and multiplication operations, and thus does not work over plain elliptic curves. OPAQUE is therefore a suitable replacement for applications that use SRP.

This draft complies with the requirements for PAKE protocols set forth in [RFC8125].

1.1. Requirements Notation

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.

1.2. Notation

The following functions are used throughout this document:

* I2OSP and OS2IP: Convert a byte string to and from a non-negative integer as described in Section 4 of [RFC8017]. Note that these functions operate on byte strings in big-endian byte order.

* concat(x0, ..., xN): Concatenate byte strings. For example, concat(0x01, 0x0203, 0x040506) = 0x010203040506.

* random(n): Generate a cryptographically secure pseudorandom byte string of length n bytes.

- *xor(a,b): Apply XOR to byte strings. For example, xor(0xF0F0, 0x1234) = 0xE2C4. It is an error to call this function with two arguments of unequal length.

- *ct_equal(a, b): Return true if a is equal to b, and false otherwise. This function is constant-time in the length of a and b, which are assumed to be of equal length, irrespective of the values a or b.

Except if said otherwise, random choices in this specification refer to drawing with uniform distribution from a given set (i.e., "random" is short for "uniformly random"). Random choices can be replaced with fresh outputs from a cryptographically strong pseudorandom generator, according to the requirements in [RFC4086], or pseudorandom function. For convenience, we define nil as a lack of value.

All protocol messages and structures defined in this document use the syntax from [RFC8446], Section 3.

The name OPAQUE is a homonym of O-PAKE where O is for Oblivious. The name OPAKE was taken.

2. Cryptographic Dependencies

OPAQUE depends on the following cryptographic protocol and primitives:

- *Oblivious Pseudorandom Function (OPRF); Section 2.1

- *Key Derivation Function (KDF); Section 2.2

- *Message Authenticate Code (MAC); Section 2.2

- *Cryptographic Hash Function; Section 2.3

- *Memory-Hard Function (MHF); Section 2.3

- *Authenticated Key Exchange (AKE) protocol; Section 2.4

This section describes these protocols and primitives in more detail. Unless said otherwise, all random nonces used in these dependencies and the rest of the OPAQUE protocol are of length Nn = 32 bytes.

2.1. Oblivious Pseudorandom Function

An Oblivious Pseudorandom Function (OPRF) is a two-party protocol between client and server for computing a PRF such that the client learns the PRF output and neither party learns the input of the

other. This specification uses the the OPRF defined in [I-D.irtf-cfrg-voprf], Version -07, with the following API and parameters:

- *Blind(x): Convert input x into an element of the OPRF group, randomize it by some scalar r, producing M, and output (r, M).

- *Evaluate(k, M): Evaluate input element M using private key k, yielding output element Z.

- *Finalize(x, r, Z): Finalize the OPRF evaluation using input x, random scalar r, and evaluation output Z, yielding output y.

- *DeriveKeyPair(seed): Derive a private and public key pair deterministically from a seed.

- *Noe: The size of a serialized OPRF group element.

- *Nok: The size of an OPRF private key.

Note that we only need the base mode variant (as opposed to the verifiable mode variant) of the OPRF described in [I-D.irtf-cfrg-voprf]. The implementation of DeriveKeyPair based on [I-D.irtf-cfrg-voprf] is below:

```
DeriveKeyPair(seed)

Input:
- seed, pseudo-random byte sequence used as a seed.

Output:
- private_key, a private key.
- public_key, the associated public key.

Steps:
1. private_key = HashToScalar(seed, dst="OPAQUE-DeriveKeyPair")
2. public_key = ScalarBaseMult(private_key)
3. Output (private_key, public_key)
```

HashToScalar(msg, dst) is as specified in [I-D.irtf-cfrg-voprf], Section 2.1.

2.2. Key Derivation Function and Message Authentication Code

A Key Derivation Function (KDF) is a cryptographic function that takes some source of intiial keying material and uses it to derive

one or more cryptographically strong keys. This specification uses a KDF with the following API and parameters:

- *Extract(salt, ikm): Extract a pseudorandom key of fixed length Nx bytes from input keying material ikm and an optional byte string salt.

- *Expand(prk, info, L): Expand a pseudorandom key prk using optional string info into L bytes of output keying material.

- *Nx: The output size of the Extract() function in bytes.

This specification also makes use of a Message Authentication Code (MAC) with the following API and parameters:

- *MAC(key, msg): Compute a message authentication code over input msg with key key, producing a fixed-length output of Nm bytes.

- *Nm: The output size of the MAC() function in bytes.

2.3. Hash Functions

This specification makes use of a cryptographic hash function with the following API and parameters:

- *Hash(msg): Apply a cryptographic hash function to input msg, producing a fixed-length digest of size Nh bytes.

- *Nh: The output size of the Hash() function in bytes.

A Memory Hard Function (MHF) is a slow and expensive cryptographic hash function with the following API:

- *Harden(msg, params): Repeatedly apply a memory-hard function with parameters params to strengthen the input msg against offline dictionary attacks. This function also needs to satisfy collision resistance.

2.4. Authenticated Key Exchange (AKE) Protocol

OPAQUE additionally depends on an Authenticated Key Exchange (AKE) protocol. This specification defines one particular AKE based on 3DH; see Section 6.2. 3DH assumes a prime-order group as described in [I-D.irtf-cfrg-voprf], Section 2.1. We let Npk and Nsk denote the size of public and private keys, respectively, used in the AKE. The AKE protocol must provide the following functions:

- *RecoverPublicKey(private_key): Recover the public key related to the input private_key.

*GenerateAuthKeyPair(): Return a randomly generated private and public key pair. This can be implemented by generating a random private key sk, then computing pk = RecoverPublicKey(sk).

*DeriveAuthKeyPair(seed): Derive a private and public authentication key pair deterministically from the input seed.

The implementation of DeriveAuthKeyPair is as follows:

```
DeriveAuthKeyPair(seed)

Input:
- seed, pseudo-random byte sequence used as a seed.

Output:
- private_key, a private key.
- public_key, the associated public key.

Steps:
1. private_key = HashToScalar(seed, dst="OPAQUE-HashToScalar")
2. public_key = ScalarBaseMult(private_key)
3. Output (private_key, public_key)
```

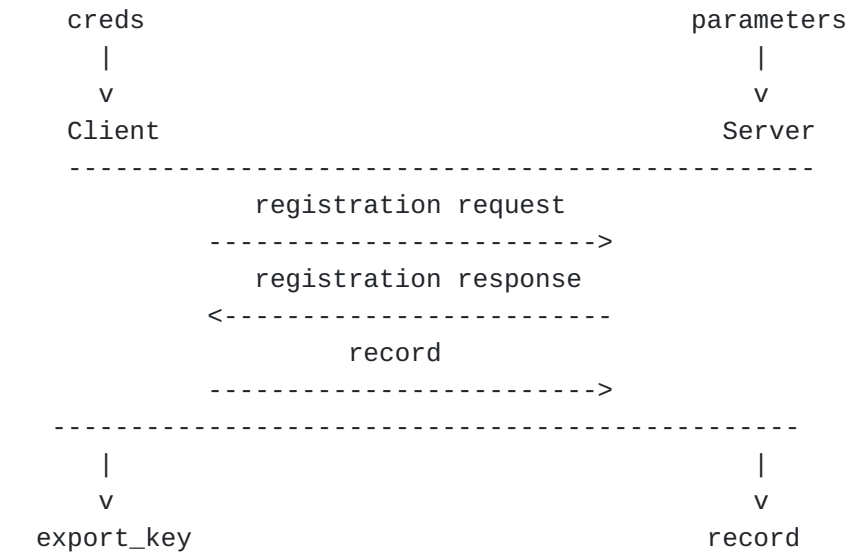
HashToScalar(msg, dst) is as specified in [I-D.irtf-cfrg-voprf], Section 2.1.

3. Protocol Overview

OPAQUE consists of two stages: registration and authenticated key exchange. In the first stage, a client registers its password with the server and stores its encrypted credentials on the server. The client inputs its credentials, which includes its password and user identifier, and the server inputs its parameters, which includes its private key and other information. The client output of this stage is a single value export_key that the client may use for application-specific purposes, e.g., to encrypt additional information to the server. The server output of this stage is a record corresponding to the client's registration that it stores in a credential file alongside other client registrations as needed.

Registration is the only part in OPAQUE that requires an authenticated and confidential channel, either physical, out-of-band, PKI-based, etc.

The registration flow is shown below:



In the second stage, a client obtains credentials previously registered with the server, recovers private key material using the password, and subsequently uses them as input to an AKE protocol. As in the registration phase, the client inputs its credentials, including its password and user identifier, and the server inputs its parameters and the credential file record corresponding to the client. The client outputs two values, an export_key (matching that from registration) and a session_key, the latter of which is the primary AKE output. The server outputs a single value session_key that matches that of the client. Upon completion, clients and servers can use these values as needed.

The authenticated key exchange flow is shown below:

The rest of this document describes the details of these stages in detail. Section 4 describes how client credential information is generated, encoded, encrypted, and stored on the server. Section 5

describes the first registration stage of the protocol, and Section 6 describes the second authentication stage of the protocol. Section 7 describes how to instantiate OPAQUE using different cryptographic dependencies and parameters.

4. Client Credential Storage

OPAQUE makes use of a structure Envelope to manage client credentials. This envelope holds information about its format and content for the client to obtain its authentication material.

OPAQUE allows applications to either provide custom client private and public keys for authentication, or to generate them internally. Each public and private key value is encoded as a byte string, specific to the AKE protocol in which OPAQUE is instantiated. These two options are defined as the external and internal modes, respectively. See Section 4.3 for their specifications.

Applications may pin key material to identities if desired. If no identity is given for a party, its value MUST default to its public key. The following types of application credential information are considered:

- *client_private_key: The encoded client private key for the AKE protocol.

- *client_public_key: The encoded client public key for the AKE protocol.

- *server_public_key: The encoded server public key for the AKE protocol.

- *client_identity: The client identity. This is an application-specific value, e.g., an e-mail address or an account name. If not specified, it defaults to the client's public key.

- *server_identity: The server identity. This is typically a domain name, e.g., example.com. If not specified, it defaults to the server's public key. See Section 10.2 for information about this identity.

These credential values are used in the CleartextCredentials structure as follows:

```
struct {
  uint8 server_public_key[Npk];
  uint8 server_identity<1..2^16-1>;
  uint8 client_identity<1..2^16-1>;
} CleartextCredentials;
```

The function CreateCleartextCredentials constructs a CleartextCredentials structure given application credential information.

```
CreateCleartextCredentials(server_public_key, client_public_key,
                           server_identity, client_identity)

Input:
- server_public_key, The encoded server public key for the AKE protocol.
- client_public_key, The encoded client public key for the AKE protocol.
- server_identity, The optional encoded server identity.
- client_identity, The optional encoded client identity.

Output:
- cleartext_credentials, a CleartextCredentials structure

Steps:
1. if server_identity == nil
2.    server_identity = server_public_key
3. if client_identity == nil
4.    client_identity = client_public_key
5. Create CleartextCredentials cleartext_credentials
   with (server_public_key, server_identity, client_identity)
6. Output cleartext_credentials
```

4.1. Envelope Structure

A client Envelope is constructed based on the EnvelopeMode, consisting of an InnerEnvelope entry whose structure is determined by the mode. Future modes MAY introduce alternate InnerEnvelope contents. Envelope is defined as follows:

```
struct {
  uint8 nonce[Nn];
  InnerEnvelope inner_env;
  uint8 auth_tag[Nm];
} Envelope;
```

nonce: A unique nonce of length Nn used to protect this Envelope.

inner_env: A mode dependent InnerEnvelope structure. See Section 4.3 for its specifications.

auth_tag: Authentication tag protecting the contents of the envelope, covering the envelope nonce, InnerEnvelope, and CleartextCredentials.

The size of the serialized envelope is denoted Ne and varies based on the mode. The exact value for Ne is specified in Section 4.3.1 and Section 4.3.2.

4.2. Envelope Creation and Recovery

Clients create an Envelope at registration with the function CreateEnvelope defined below.

For the internal mode, implementations can choose to leave out the client_private_key parameter, as it is not used. For the external mode, implementations are free to additionally provide client_public_key to this function. With this, the public key does not need to be recovered by BuildInnerEnvelope() and that function should be adapted accordingly.

```
CreateEnvelope(randomized_pwd, server_public_key, client_private_key,
               server_identity, client_identity)

Parameter:
- mode, the EnvelopeMode mode

Input:
- randomized_pwd, randomized password.
- server_public_key, The encoded server public key for
  the AKE protocol.
- client_private_key, The encoded client private key for
  the AKE protocol. This is nil in the internal key mode.
- server_identity, The optional encoded server identity.
- client_identity, The optional encoded client identity.

Output:
- envelope, the client's `Envelope` structure.
- client_public_key, the client's AKE public key.
- masking_key, a key used by the server to encrypt the
  envelope during login.
- export_key, an additional client key.

Steps:
1. envelope_nonce = random(Nn)
2. auth_key = Expand(randomized_pwd, concat(envelope_nonce, "AuthKey"),
3. export_key = Expand(randomized_pwd, concat(envelope_nonce, "ExportKey
4. masking_key = Expand(randomized_pwd, "MaskingKey", Nh)
5. inner_env, client_public_key = BuildInnerEnvelope(randomized_pwd, env
6. cleartext_creds = CreateCleartextCredentials(server_public_key, clien
7. auth_tag = MAC(auth_key, concat(envelope_nonce, inner_env, cleartext_
8. Create Envelope envelope with (envelope_nonce, inner_env, auth_tag)
9. Output (envelope, client_public_key, masking_key, export_key)
```

Clients recover their Envelope during authentication with the RecoverEnvelope function defined below.

```
RecoverEnvelope(randomized_pwd, server_public_key, envelope,
                server_identity, client_identity)

Input:
- randomized_pwd, randomized password.
- server_public_key, The encoded server public key for the AKE protocol.
- envelope, the client's `Envelope` structure.
- server_identity, The optional encoded server identity.
- client_identity, The optional encoded client identity.

Output:
- client_private_key, The encoded client private key for the AKE protoco
- export_key, an additional client key.

Exceptions:
- EnvelopeRecoveryError, when the envelope fails to be recovered

Steps:
1. auth_key = Expand(randomized_pwd, concat(envelope.nonce, "AuthKey"),
2. export_key = Expand(randomized_pwd, concat(envelope.nonce, "ExportKey
3. (client_private_key, client_public_key) =
    RecoverKeys(randomized_pwd, envelope.nonce, envelope.inner_env)
4. cleartext_creds = CreateCleartextCredentials(server_public_key,
                      client_public_key, server_identity, client_identit
5. expected_tag = MAC(auth_key, concat(envelope.nonce, inner_env, cleart
6. If !ct_equal(envelope.auth_tag, expected_tag),
     raise EnvelopeRecoveryError
7. Output (client_private_key, export_key)
```

4.3. Envelope Modes

The EnvelopeMode specifies the structure and encoding of the corresponding InnerEnvelope. This document specifies the values of the two aforementioned modes:

```
enum {
  internal(1),
  external(2),
  (255)
} EnvelopeMode;
```

Each EnvelopeMode defines its own InnerEnvelope structure and must implement the following interface:

*inner_env, client_public_key = BuildInnerEnvelope(randomized_pwd, nonce, client_private_key): Build and return the mode's InnerEnvelope structure and the client's public key.

*client_private_key, client_public_key =
RecoverKeys(randomized_pwd, nonce, inner_env): Recover and return the client's private and public keys for the AKE protocol.

The implementations of this interface for both internal and external modes are in Section 4.3.1 and Section 4.3.2, respectively.

The size of the envelope may vary between modes. If applications implement Section 10.8, they MUST use the same envelope mode throughout their lifecycle in order to avoid activity leaks due to mode switching.

4.3.1. Internal Mode

In this mode, the client's private and public keys are deterministically derived from the OPRF output.

With the internal key mode the EnvelopeMode value MUST be internal and the InnerEnvelope is empty, and the size Ne of the serialized Envelope is Nn + Nm.

```
BuildInnerEnvelope(randomized_pwd, nonce, client_private_key)

Input:
- randomized_pwd, randomized password.
- nonce, a unique nonce of length `Nn`.
- client_private_key, empty value. Not used in this function,
  it only serves to comply with the API.

Output:
- inner_env, nil value (serves to comply with the API).
- client_public_key, the client's AKE public key.

Steps:
1. seed = Expand(randomized_pwd, concat(nonce, "PrivateKey"), Nsk)
2. _, client_public_key = DeriveAuthKeyPair(seed)
3. Output (nil, client_public_key)
```

Note that implementations are free to leave out the client_private_key parameter, as it is not used.

```
RecoverKeys(randomized_pwd, nonce, inner_env)

Input:
- randomized_pwd, randomized password.
- nonce, a unique nonce of length `Nn`.
- inner_env, an InnerEnvelope structure. Not used in this
  function, it only serves to comply with the API.

Output:
- client_private_key, The encoded client private key for the AKE protoco
- client_public_key, The encoded client public key for the AKE protocol.

Steps:
1. seed = Expand(randomized_pwd, concat(nonce, "PrivateKey"), Nsk)
2. client_private_key, client_public_key = DeriveAuthKeyPair(seed)
4. Output (client_private_key, client_public_key)
```

Note that implementations are free to leave out the inner_env parameter, as it is not used.

4.3.2. External Mode

This mode allows applications to import or generate keys for the client. This specification only imports the client's private key and internally recovers the corresponding public key. Implementations are free to import both, in which case the functions FinalizeRequest(), CreateEnvelope(), and BuildInnerEnvelope() must be adapted accordingly.

With the external key mode the EnvelopeMode value MUST be external, and the size Ne of the serialized Envelope is Nn + Nm + Nsk.

An encryption key is generated from the hardened OPRF output and used to encrypt the client's private key, which is then stored encrypted in the InnerEnvelope. On key recovery, the client's public key is recovered using the private key.

```
struct {
  uint8 encrypted_creds[Nsk];
} InnerEnvelope;
```

encrypted_creds : Encrypted client_private_key. Authentication of this field is ensured with the auth_tag field in the envelope that covers this InnerEnvelope.

If the implementation provides the client_public_key, then BuildInnerEnvelope() can skip the RecoverPublicKey() call.

```
BuildInnerEnvelope(randomized_pwd, nonce, client_private_key)

Input:
- randomized_pwd, randomized password.
- nonce, a unique nonce of length `Nn`.
- client_private_key, the encoded client private key for the AKE protoco

Output:
- inner_env, an InnerEnvelope structure.
- client_public_key, The encoded client public key for the AKE protocol.

Steps:
1. pseudorandom_pad = Expand(randomized_pwd, concat(nonce, "Pad"), len(c
2. encrypted_creds = xor(client_private_key, pseudorandom_pad)
3. Create InnerEnvelope inner_env with encrypted_creds
4. client_public_key = RecoverPublicKey(client_private_key)
5. Output (inner_env, client_public_key)

RecoverKeys(randomized_pwd, nonce, inner_env)

Input:
- randomized_pwd, randomized password.
- nonce, a unique nonce of length `Nn`.
- inner_env, an InnerEnvelope structure.

Output:
- client_private_key, the encoded client private key for the AKE protoco
- client_public_key, the client's AKE public key.

Steps:
1. encrypted_creds = inner_env.encrypted_creds
2. pseudorandom_pad = Expand(randomized_pwd, concat(nonce, "Pad"), len(e
3. client_private_key = xor(encrypted_creds, pseudorandom_pad)
4. client_public_key = RecoverPublicKey(client_private_key)
5. Output (client_private_key, client_public_key)
```

5. Offline Registration

This section describes the registration flow, message encoding, and helper functions. In a setup phase, the client chooses its password, and the server chooses its own pair of private-public AKE keys (server_private_key, server_public_key) for use with the AKE, along with a Nh-byte oprf_seed. The server can use the same pair of keys with multiple clients and can opt to use multiple seeds (so long as they are kept consistent for each client). These steps can happen offline, i.e., before the registration phase.

If using external mode, the client provides a key pair (client_private_key, client_public_key) for an AKE protocol which is suitable for use with OPAQUE; See Section 6. The private-public keys

(client_private_key, client_public_key) may be randomly generated (using a cryptographically secure pseudorandom number generator) for the account or provided by the calling client. Clients MUST NOT use the same key pair (client_private_key, client_public_key) for two different accounts.

Once complete, the registration process proceeds as follows. The client inputs the following values:

- password: client password.

- creds: client credentials, as described in Section 4.

The server inputs the following values:

- server_private_key: server private key for the AKE protocol.

- server_public_key: server public key for the AKE protocol.

- credential_identifier: client credential identifier.

- oprf_seed: seed used to derive per-client OPRF keys.

The registration protocol then runs as shown below:

```
 Client                                              Server
-------------------------------------------------------
(request, blind) = CreateRegistrationRequest(password)

                        request
             ------------------------->

response = CreateRegistrationResponse(request,
                                      server_public_key,
                                      credential_identifier,
                                      oprf_seed)

                        response
             <-------------------------

(record, export_key) = FinalizeRequest(client_private_key,
                                       password,
                                       blind,
                                       response,
                                       server_identity,
                                       client_identity)

                        record
             ------------------------->
```

Section 5.1.1 describes details of the functions and the corresponding parameters referenced above.

Both client and server MUST validate the other party's public key before use. See Section 10.6 for more details. Upon completion, the server stores the client's credentials for later use. Moreover, the client MAY use the output export_key for further application-specific purposes; see Section 10.4.

5.1. Registration Messages

```
struct {
  uint8 data[Noe];
} RegistrationRequest;
```

data A serialized OPRF group element.

```
struct {
  uint8 data[Noe];
  uint8 server_public_key[Npk];
} RegistrationResponse;
```

data A serialized OPRF group element.

server_public_key The server's encoded public key that will be used for the online authenticated key exchange stage.

```
struct {
  uint8 client_public_key[Npk];
  uint8 masking_key[Nh];
  Envelope envelope;
} RegistrationUpload;
```

client_public_key The client's encoded public key, corresponding to the private key client_private_key.

masking_key A key used by the server to preserve confidentiality of the envelope during login.

envelope The client's Envelope structure.

5.1.1. Registration Functions

5.1.1.1. CreateRegistrationRequest

```
CreateRegistrationRequest(password)

Input:
- password, an opaque byte string containing the client's password.

Output:
- request, a RegistrationRequest structure.
- blind, an OPRF scalar value.

Steps:
1. (blind, M) = Blind(password)
2. Create RegistrationRequest request with M
3. Output (request, blind)
```

5.1.1.2. CreateRegistrationResponse

```
CreateRegistrationResponse(request, server_public_key, credential_identi

Input:
- request, a RegistrationRequest structure.
- server_public_key, the server's public key.
- credential_identifier, an identifier that uniquely represents the cred
  registered.
- oprf_seed, the seed of Nh bytes used by the server to generate an oprf

Output:
- response, a RegistrationResponse structure.

Steps:
1. ikm = Expand(oprf_seed, concat(credential_identifier, "OprfKey"), Nok
2. (oprf_key, _) = DeriveKeyPair(ikm)
3. Z = Evaluate(oprf_key, request.data)
4. Create RegistrationResponse response with (Z, server_public_key)
5. Output response
```

5.1.1.3. FinalizeRequest

To create the user record used for further authentication, the client executes the following function. In the internal key mode, the client_private_key is nil.

Depending on the mode, implementations are free to leave out the client_private_key parameter (internal mode), or to additionally include client_public_key (external mode). See Section 4.2 for more details.

```
FinalizeRequest(client_private_key, password, blind, response,
                server_identity, client_identity)

Input:
- client_private_key, the client's private key. In internal mode, this i
- password, an opaque byte string containing the client's password.
- blind, the OPRF scalar value used for blinding.
- response, a RegistrationResponse structure.
- server_identity, the optional encoded server identity.
- client_identity, the optional encoded client identity.

Output:
- record, a RegistrationUpload structure.
- export_key, an additional client key.

Steps:
1. y = Finalize(password, blind, response.data)
2. randomized_pwd = Extract("", Harden(y, params))
3. (envelope, client_public_key, masking_key, export_key) =
    CreateEnvelope(randomized_pwd, response.server_public_key, client_pr
                   server_identity, client_identity)
4. Create RegistrationUpload record with (client_public_key, masking_key
5. Output (record, export_key)
```

See Section 6 for details about the output export_key usage.

Upon completion of this function, the client MUST send record to the server.

5.1.1.4. Finalize Registration

The server stores the record object as the credential file for each client along with the associated credential_identifier and client_identity (if different). Note that the values oprf_seed and server_private_key from the server's setup phase must also be persisted.

6. Online Authenticated Key Exchange

The generic outline of OPAQUE with a 3-message AKE protocol includes three messages ke1, ke2, and ke3, where ke1 and ke2 include key exchange shares, e.g., DH values, sent by the client and server, respectively, and ke3 provides explicit client authentication and full forward security (without it, forward secrecy is only achieved against eavesdroppers, which is insufficient for OPAQUE security).

This section describes the online authenticated key exchange protocol flow, message encoding, and helper functions. This stage is composed of a concurrent OPRF and key exchange flow. The key exchange protocol is authenticated using the client and server

credentials established during registration; see Section 5. In the end, the client proves its knowledge of the password, and both client and server agree on (1) a mutually authenticated shared secret key and (2) any optional application information exchange during the handshake.

In this stage, the client inputs the following values:

- *password: client password.

- *client_identity: client identity, as described in Section 4.

The server inputs the following values:

- *server_private_key: server private for the AKE protocol.

- *server_public_key: server public for the AKE protocol.

- *server_identity: server identity, as described in Section 4.

- *record: RegistrationUpload corresponding to the client's registration.

- *credential_identifier: client credential identifier.

- *oprf_seed: seed used to derive per-client OPRF keys.

The client receives two outputs: a session secret and an export key. The export key is only available to the client, and may be used for additional application-specific purposes, as outlined in Section 10.4. The output export_key MUST NOT be used in any way before the protocol completes successfully. See Section 10.3 for more details about this requirement. The server receives a single output: a session secret matching that of the client's.

The protocol runs as shown below:

```
  Client                                              Server
 -------------------------------------------------------
  ke1 = ClientInit(client_identity, password)

                           ke1
              ------------------------->

  ke2 = ServerInit(server_identity, server_private_key,
                   server_public_key, record,
                   credential_identifier, oprf_seed, ke1)

                           ke2
              <-------------------------

    (ke3,
    session_key,
    export_key) = ClientFinish(client_identity, password,
                               server_identity, ke2)

                           ke3
              ------------------------->

                       session_key = ServerFinish(ke3)
```

The rest of this section describes these authenticated key exchange messages and their parameters in more detail. Section 6.1 discusses internal functions used for retrieving client credentials, and Section 6.2 discusses how these functions are used to execute the authenticated key exchange protocol.

6.1. Credential Retrieval

6.1.1. Credential Retrieval Messages

```
struct {
  uint8 data[Noe];
} CredentialRequest;
```

data A serialized OPRF group element.

```
struct {
  uint8 data[Noe];
  uint8 masking_nonce[Nn];
  uint8 masked_response[Npk + Ne];
} CredentialResponse;
```

data A serialized OPRF group element.

masking_nonce A nonce used for the confidentiality of the masked_response field.

masked_response
An encrypted form of the server's public key and the client's Envelope structure.

6.1.2. Credential Retrieval Functions

6.1.2.1. CreateCredentialRequest

```
CreateCredentialRequest(password)

Input:
- password, an opaque byte string containing the client's password.

Output:
- request, a CredentialRequest structure.
- blind, an OPRF scalar value.

Steps:
1. (blind, M) = Blind(password)
2. Create CredentialRequest request with M
3. Output (request, blind)
```

6.1.2.2. CreateCredentialResponse

There are two scenarios to handle for the construction of a CredentialResponse object: either the record for the client exists (corresponding to a properly registered client), or it was never created (corresponding to a client that has yet to register).

In the case of an existing record with the corresponding identifier credential_identifier, the server invokes the following function to produce a CredentialResponse:

```
CreateCredentialResponse(request, server_public_key, record,
                         credential_identifier, oprf_seed)

Input:
- request, a CredentialRequest structure.
- server_public_key, the public key of the server.
- record, an instance of RegistrationUpload which is the server's
  output from registration.
- credential_identifier, an identifier that uniquely represents the cred
  being registered.
- oprf_seed, the server-side seed of Nh bytes used to generate an oprf_k

Output:
- response, a CredentialResponse structure.

Steps:
1. ikm = Expand(oprf_seed, concat(credential_identifier, "OprfKey"), Nok
2. (oprf_key, _) = DeriveKeyPair(ikm)
3. Z = Evaluate(oprf_key, request.data)
4. masking_nonce = random(Nn)
5. credential_response_pad = Expand(record.masking_key,
     concat(masking_nonce, "CredentialResponsePad"), Npk + Ne)
6. masked_response = xor(credential_response_pad,
                         concat(server_public_key, record.envelope))
7. Create CredentialResponse response with (Z, masking_nonce, masked_res
8. Output response
```

In the case of a record that does not exist and if client enumeration prevention is desired, the server MUST respond to the credential request to fake the existence of the record. The server SHOULD invoke the CreateCredentialResponse function with a fake client record argument that is configured so that:

- record.client_public_key is set to a randomly generated public key of length Npk
- record.masking_key is set to a random byte string of length Nh
- record.envelope is set to the byte string consisting only of zeros of length Ne

It is RECOMMENDED that a fake client record is created once (e.g. as the first user record of the application) and stored alongside legitimate client records. This allows servers to locate the record in time comparable to that of a legitimate client record.

Note that the responses output by either scenario are indistinguishable to an adversary that is unable to guess the registered password for the client corresponding to credential_identifier.

6.1.2.3. RecoverCredentials

```
RecoverCredentials(password, blind, response,
                   server_identity, client_identity)

Input:
- password, an opaque byte string containing the client's password.
- blind, an OPRF scalar value.
- response, a CredentialResponse structure.
- server_identity, The optional encoded server identity.
- client_identity, The optional encoded client identity.

Output:
- client_private_key, the client's private key for the AKE protocol.
- server_public_key, the public key of the server.
- export_key, an additional client key.

Steps:
1. y = Finalize(password, blind, response.data)
2. randomized_pwd = Extract("", Harden(y, params))
3. masking_key = Expand(randomized_pwd, "MaskingKey", Nh)
4. credential_response_pad = Expand(masking_key,
     concat(response.masking_nonce, "CredentialResponsePad"), Npk + Ne)
5. concat(server_public_key, envelope) = xor(credential_response_pad,
                                             response.masked_response)
6. (client_private_key, export_key) =
    RecoverEnvelope(randomized_pwd, server_public_key, envelope,
                    server_identity, client_identity)
7. Output (client_private_key, response.server_public_key, export_key)
```

6.2. AKE Protocol

This section describes the authenticated key exchange protocol for OPAQUE using 3DH, a 3-message AKE which satisfies the forward secrecy and KCI properties discussed in Section 10. The protocol consists of three messages sent between client and server, each computed using the following application APIs:

```
  *ke1 = ClientInit(client_identity, password)

  *ke2 = ServerInit(server_identity, server_private_key,
   server_public_key, record, credential_identifier, oprf_seed, ke1)

  *ke3, session_key, export_key = ClientFinish(password,
   client_identity, server_identity, ke2)

  *session_key = ServerFinish(ke3)
```

Outputs ke1, ke2, and ke3 are the three protocol messages sent between client and server. session_key and export_key are outputs to

be consumed by applications. Applications can use session_key to derive additional keying material as needed.

Both ClientFinish and ServerFinish return an error if authentication failed. In this case, clients and servers MUST NOT use any outputs from the protocol, such as session_key or export_key. ClientInit and ServerInit both implicitly return internal state objects client_state and server_state, respectively, with the following named fields:

```
struct {
  uint8 blind[Nok];
  uint8 client_secret[Nsk];
  KE1 ke1;
} ClientState;

struct {
  uint8 expected_client_mac[Nm];
  uint8 session_key[Nx];
} ServerState;
```

Section 6.2.3 and Section 6.2.4 specify the inner working of these functions and their parameters for clients and servers, respectively.

Prior to the execution of these functions, both the client and the server MUST agree on a configuration; see Section 7 for details.

6.2.1. Protocol Messages

```
struct {
  CredentialRequest request;
  uint8 client_nonce[Nn];
  uint8 client_keyshare[Npk];
} KE1;
```

request A CredentialRequest generated according to Section 6.1.2.1.

client_nonce A fresh randomly generated nonce of length Nn.

client_keyshare Client ephemeral key share of fixed size Npk, where Npk depends on the corresponding prime order group.

```
struct {
  struct {
    CredentialResponse response;
    uint8 server_nonce[Nn];
    uint8 server_keyshare[Npk];
  } inner_ke2;
  uint8 server_mac[Nm];
} KE2;
```

response A CredentialResponse generated according to Section 6.1.2.2.

server_nonce A fresh randomly generated nonce of length Nn.

server_keyshare Server ephemeral key share of fixed size Npk, where Npk depends on the corresponding prime order group.

server_mac An authentication tag computed over the handshake transcript computed using Km2, defined below.

```
struct {
  uint8 client_mac[Nm];
} KE3;
```

client_mac An authentication tag computed over the handshake transcript computed using Km2, defined below.

6.2.2. Key Schedule Functions

6.2.2.1. Transcript Functions

The OPAQUE-3DH key derivation procedures make use of the functions below, re-purposed from TLS 1.3 [RFC8446].

```
Expand-Label(Secret, Label, Context, Length) =
    Expand(Secret, CustomLabel, Length)
```

Where CustomLabel is specified as:

```
struct {
  uint16 length = Length;
  opaque label<8..255> = "OPAQUE-" + Label;
  uint8 context<0..255> = Context;
} CustomLabel;
```

```
Derive-Secret(Secret, Label, Transcript-Hash) =
    Expand-Label(Secret, Label, Transcript-Hash, Nx)
```

Note that the Label parameter is not a NULL-terminated string.

OPAQUE-3DH can optionally include shared context information in the transcript, such as configuration parameters or application-specific info, e.g. "appXYZ-v1.2.3".

The OPAQUE-3DH key schedule requires a preamble, which is computed as follows.

```
Preamble(client_identity, ke1, server_identity, inner_ke2)

Parameters:
- context, optional shared context information.

Input:
- client_identity, the optional encoded client identity, which is set
  to client_public_key if not specified.
- ke1, a KE1 message structure.
- server_identity, the optional encoded server identity, which is set
  to server_public_key if not specified.
- inner_ke2, an inner_ke2 structure as defined in KE2.

Output:
- preamble, the protocol transcript with identities and messages.

Steps:
1. preamble = concat("RFCXXXX",
                     I2OSP(len(context), 2), context,
                     I2OSP(len(client_identity), 2), client_identity,
                     ke1,
                     I2OSP(len(server_identity), 2), server_identity,
                     inner_ke2)
2. Output preamble
```

6.2.2.2. Shared Secret Derivation

The OPAQUE-3DH shared secret derived during the key exchange protocol is computed using the following function.

```
TripleDHIKM(sk1, pk1, sk2, pk2, sk3, pk3)

Input:
- skx, scalar to be multiplied with their corresponding pkx.
- pkx, element to be multiplied with their corresponding skx.

Output:
- ikm, input key material.

Steps:
1. dh1 = SerializeElement(sk1 * pk1)
2. dh2 = SerializeElement(sk2 * pk2)
3. dh3 = SerializeElement(sk3 * pk3)
4. Output concat(dh1, dh2, dh3)
```

Using this shared secret, further keys used for encryption and authentication are computed using the following function.

```
DeriveKeys(ikm, preamble)

Input:
- ikm, input key material.
- preamble, the transcript as defined by Preamble().

Output:
- Km2, a MAC authentication key.
- Km3, a MAC authentication key.
- session_key, the shared session secret.

Steps:
1. prk = Extract("", ikm)
2. handshake_secret = Derive-Secret(prk, "HandshakeSecret", Hash(preambl
3. session_key = Derive-Secret(prk, "SessionKey", Hash(preamble))
4. Km2 = Derive-Secret(handshake_secret, "ServerMAC", "")
5. Km3 = Derive-Secret(handshake_secret, "ClientMAC", "")
6. Output (Km2, Km3, session_key)
```

6.2.3. External Client API

```
ClientInit(client_identity, password)

State:
- state, a ClientState structure.

Input:
- client_identity, the optional encoded client identity, which is nil
  if not specified.
- password, an opaque byte string containing the client's password.

Output:
- ke1, a KE1 message structure.
- blind, the OPRF blinding scalar.
- client_secret, the client's Diffie-Hellman secret share for the sessio

Steps:
1. request, blind = CreateCredentialRequest(password)
2. state.blind = blind
3. ke1 = Start(request)
4. Output ke1

ClientFinish(client_identity, password, server_identity, ke1, ke2)

State:
- state, a ClientState structure

Input:
- client_identity, the optional encoded client identity, which is set
  to client_public_key if not specified.
- password, an opaque byte string containing the client's password.
- server_identity, the optional encoded server identity, which is set
  to server_public_key if not specified.
- ke1, a KE1 message structure.
- ke2, a KE2 message structure.

Output:
- ke3, a KE3 message structure.
- session_key, the session's shared secret.

Steps:
1. (client_private_key, server_public_key, export_key) =
    RecoverCredentials(password, state.blind, ke2.CredentialResponse,
                       server_identity, client_identity)
2. (ke3, session_key) =
    ClientFinalize(client_identity, client_private_key, server_identity,
                   server_public_key, ke1, ke2)
3. Output (ke3, session_key)
```

6.2.3.1. Internal Client Functions

```
Start(credential_request)

Parameters:
- Nn, the nonce length.

State:
- state, a ClientState structure.

Input:
- credential_request, a CredentialRequest structure.

Output:
- ke1, a KE1 structure.

Steps:
1. client_nonce = random(Nn)
2. client_secret, client_keyshare = GenerateAuthKeyPair()
3. Create KE1 ke1 with (credential_request, client_nonce, client_keyshar
4. state.client_secret = client_secret
5. Output (ke1, client_secret)
```

```
ClientFinalize(client_identity, client_private_key, server_identity,
               server_public_key, ke1, ke2)

State:
- state, a ClientState structure.

Input:
- client_identity, the optional encoded client identity, which is
  set to client_public_key if not specified.
- client_private_key, the client's private key.
- server_identity, the optional encoded server identity, which is
  set to server_public_key if not specified.
- server_public_key, the server's public key.
- ke2, a KE2 message structure.

Output:
- ke3, a KE3 structure.
- session_key, the shared session secret.

Exceptions:
- HandshakeError, when the handshake fails

Steps:
1. ikm = TripleDHIKM(state.client_secret, ke2.server_keyshare,
    state.client_secret, server_public_key, client_private_key, ke2.serv
2. preamble = Preamble(client_identity, state.ke1, server_identity, ke2.
3. Km2, Km3, session_key = DeriveKeys(ikm, preamble)
4. expected_server_mac = MAC(Km2, Hash(preamble))
5. If !ct_equal(ke2.server_mac, expected_server_mac),
     raise HandshakeError
6. client_mac = MAC(Km3, Hash(concat(preamble, expected_server_mac))
7. Create KE3 ke3 with client_mac
8. Output (ke3, session_key)
```

6.2.4. External Server API

```
ServerInit(server_identity, server_private_key, server_public_key,
           record, credential_identifier, oprf_seed, ke1)

Input:
- server_identity, the optional encoded server identity, which is set to
  server_public_key if nil.
- server_private_key, the server's private key.
- server_public_key, the server's public key.
- record, the client's RegistrationUpload structure.
- credential_identifier, an identifier that uniquely represents the cred
  being registered.
- oprf_seed, the server-side seed of Nh bytes used to generate an oprf_k
- ke1, a KE1 message structure.

Output:
- ke2, a KE2 structure.

Steps:
1. response = CreateCredentialResponse(ke1.request, server_public_key, r
    credential_identifier, oprf_seed)
2. ke2 = Response(server_identity, server_private_key,
    client_identity, record.client_public_key, ke1, response)
3. Output ke2

ServerFinish(ke3)

State:
- state, a ServerState structure.

Input:
- ke3, a KE3 structure.

Output:
- session_key, the shared session secret if and only if KE3 is valid.

Exceptions:
- HandshakeError, when the handshake fails

Steps:
1. if !ct_equal(ke3.client_mac, state.expected_client_mac):
2.    raise HandshakeError
3. Output state.session_key
```

6.2.4.1. Internal Server Functions

```
Response(server_identity, server_private_key, client_identity,
         client_public_key, ke1, credential_response)

Parameters:
- Nn, the nonce length.

State:
- state, a ServerState structure.

Input:
- server_identity, the optional encoded server identity, which is set to
  server_public_key if not specified.
- server_private_key, the server's private key.
- client_identity, the optional encoded client identity, which is set to
  client_public_key if not specified.
- client_public_key, the client's public key.
- ke1, a KE1 message structure.
- credential_response, a CredentialResponse structure.

Output:
- ke2, A KE2 structure.

Steps:
1. server_nonce = random(Nn)
2. server_secret, server_keyshare = GenerateAuthKeyPair()
3. Create inner_ke2 ike2 with (credential_response, server_nonce, server
4. preamble = Preamble(client_identity, ke1, server_identity, ike2)
5. ikm = TripleDHIKM(server_secret, ke1.client_keyshare, server_private_
6. Km2, Km3, session_key = DeriveKeys(ikm, preamble)
7. server_mac = MAC(Km2, Hash(preamble))
8. expected_client_mac = MAC(Km3, Hash(concat(preamble, server_mac))
9. Populate state with ServerState(expected_client_mac, session_key)
10. Create KE2 ke2 with (ike2, server_mac)
11. Output ke2
```

7. Configurations

An OPAQUE-3DH configuration is a tuple (OPRF, KDF, MAC, Hash, MHF, EnvelopeMode, Group, Context) such that the following conditions are met:

- The OPRF protocol uses the "base mode" variant of [I-D.irtf-cfrg-voprf] and implements the interface in Section 2. Examples include OPRF(ristretto255, SHA-512) and OPRF(P-256, SHA-256).

- The KDF, MAC, and Hash functions implement the interfaces in Section 2. Examples include HKDF [RFC5869] for the KDF, HMAC [RFC2104] for the MAC, and SHA-256 and SHA-512 for the Hash

functions. If an extensible output function such as SHAKE128 [FIPS202] is used then the output length Nh MUST be chosen to align with the target security level of the OPAQUE configuration. For example, if the target security parameter for the configuration is 128-bits, then Nh SHOULD be at least 32 bytes.

- The MHF has fixed parameters, chosen by the application, and implements the interface in Section 2. Examples include Argon2 [I-D.irtf-cfrg-argon2], scrypt [RFC7914], and PBKDF2 [RFC2898] with fixed parameter choices.

- EnvelopeMode value is as defined in Section 4, and is one of internal or external.

- The Group mode identifies the group used in the OPAQUE-3DH AKE. This SHOULD match that of the OPRF. For example, if the OPRF is OPRF(ristretto255, SHA-512), then Group SHOULD be ristretto255.

Context is the shared parameter used to construct the preamble in Section 6.2.2.1. This parameter SHOULD include any application-specific configuration information or parameters that are needed to prevent cross-protocol or downgrade attacks.

Absent an application-specific profile, the following configurations are RECOMMENDED:

- OPRF(ristretto255, SHA-512), HKDF-SHA-512, HMAC-SHA-512, SHA-512, Scrypt(32768,8,1), internal, ristretto255

- OPRF(P-256, SHA-256), HKDF-SHA-256, HMAC-SHA-256, SHA-256, Scrypt(32768,8,1), internal, P-256

Future configurations may specify different combinations of dependent algorithms, with the following considerations:

1. The size of AKE public and private keys -- Npk and Nsk, respectively -- must adhere to the output length limitations of the KDF Expand function. If HKDF is used, this means Npk, Nsk <= 255 * Nx, where Nx is the output size of the underlying hash function. See [RFC5869] for details.

2. The output size of the Hash function SHOULD be long enough to produce a key for MAC of suitable length. For example, if MAC is HMAC-SHA256, then Nh could be the 32 bytes.

8. Application Considerations

Beyond choosing an appropriate configuration, there are several parameters which applications can use to control OPAQUE:

- Client credential identifier: As described in Section 5, this is a unique handle to the client credential being stored. In applications where there are alternate client identifiers that accompany an account, such as a username or email address, this identifier can be set to those alternate values. Applications SHOULD set the credential identifier to the client identifier. Applications MUST NOT use the same credential identifier for multiple clients.

- Context information: As described in Section 7, applications may include a shared context string that is authenticated as part of the handshake. This parameter SHOULD include any configuration information or parameters that are needed to prevent cross-protocol or downgrade attacks. This context information is not sent over the wire in any key exchange messages. However, applications may choose to send it alongside key exchange messages if needed for their use case.

- Client and server identifier: As described in Section 4, clients and servers are identified with their public keys by default. However, applications may choose alternate identifiers that are pinned to these public keys. For example, servers may use a domain name instead of a public key as their identifier. Absent alternate notions of an identity, applications SHOULD set these identifiers to nil and rely solely on public key information.

- Enumeration prevention: As described in Section 6.1.2.2, if servers receive a credential request for a non-existent client, they SHOULD respond with a "fake" response in order to prevent active client enumeration attacks. Servers that implement this mitigation SHOULD use the same configuration information (such as the oprf_seed) for all clients; see Section 10.8. In settings where this attack is not a concern, servers may choose to not support this functionality.

9. Implementation Considerations

Implementations of OPAQUE should consider addressing the following:

- Clearing secrets out of memory: All private key material and intermediate values, including the outputs of the key exchange phase, should not be retained in memory after deallocation.

- Constant-time operations: All operations, particularly the cryptographic and group arithmetic operations, should be

constant-time and independent of the bits of any secrets. This includes any conditional branching during the creation of the credential response, to support implementations which provide mitigations against client enumeration attacks.

* Deserialization checks: When parsing messages that have crossed trust boundaries (e.g. a network wire), implementations should properly handle all error conditions covered in [I-D.irtf-cfrg-voprf] and abort accordingly.

10. Security Considerations

OPAQUE is defined and proven as the composition of two functionalities: an OPRF and an AKE protocol. It can be seen as a "compiler" for transforming any AKE protocol (with KCI security and forward secrecy - see below) into a secure aPAKE protocol. In OPAQUE, the client stores a secret private key at the server during password registration and retrieves this key each time it needs to authenticate to the server. The OPRF security properties ensure that only the correct password can unlock the private key while at the same time avoiding potential offline guessing attacks. This general composability property provides great flexibility and enables a variety of OPAQUE instantiations, from optimized performance to integration with TLS. The latter aspect is of prime importance as the use of OPAQUE with TLS constitutes a major security improvement relative to the standard password-over-TLS practice. At the same time, the combination with TLS builds OPAQUE as a fully functional secure communications protocol and can help provide privacy to account information sent by the client to the server prior to authentication.

The KCI property required from AKE protocols for use with OPAQUE states that knowledge of a party's private key does not allow an attacker to impersonate others to that party. This is an important security property achieved by most public-key based AKE protocols, including protocols that use signatures or public key encryption for authentication. It is also a property of many implicitly authenticated protocols, e.g., HMQV, but not all of them. We also note that key exchange protocols based on shared keys do not satisfy the KCI requirement, hence they are not considered in the OPAQUE setting. We note that KCI is needed to ensure a crucial property of OPAQUE: even upon compromise of the server, the attacker cannot impersonate the client to the server without first running an exhaustive dictionary attack. Another essential requirement from AKE protocols for use in OPAQUE is to provide forward secrecy (against active attackers).

10.1. Related Analysis

Jarecki et al. [OPAQUE] proved the security of OPAQUE in a strong aPAKE model that ensures security against pre-computation attacks and is formulated in the Universal Composability (UC) framework [Canetti01] under the random oracle model. This assumes security of the OPRF function and the underlying key exchange protocol. In turn, the security of the OPRF protocol from [I-D.irtf-cfrg-voprf] is proven in the random oracle model under the One-More Diffie-Hellman assumption [JKKX16].

Very few aPAKE protocols have been proven formally, and those proven were analyzed in a weak security model that allows for pre-computation attacks (e.g., [GMR06]). This is not just a formal issue: these protocols are actually vulnerable to such attacks. This includes protocols that have recent analyses in the UC model such as AuCPace [AuCPace] and SPAKE2+ [SPAKE2plus]. We note that as shown in [OPAQUE], these protocols, and any aPAKE in the model from [GMR06], can be converted into an aPAKE secure against pre-computation attacks at the expense of an additional OPRF execution.

OPAQUE's design builds on a line of work initiated in the seminal paper of Ford and Kaliski [FK00] and is based on the HPAKE protocol of Xavier Boyen [Boyen09] and the (1,1)-PPSS protocol from Jarecki et al. [JKKX16]. None of these papers considered security against pre-computation attacks or presented a proof of aPAKE security (not even in a weak model).

10.2. Identities

AKE protocols generate keys that need to be uniquely and verifiably bound to a pair of identities. In the case of OPAQUE, those identities correspond to client_identity and server_identity. Thus, it is essential for the parties to agree on such identities, including an agreed bit representation of these identities as needed.

Applications may have different policies about how and when identities are determined. A natural approach is to tie client_identity to the identity the server uses to fetch envelope (hence determined during password registration) and to tie server_identity to the server identity used by the client to initiate an offline password registration or online authenticated key exchange session. server_identity and client_identity can also be part of the envelope or be tied to the parties' public keys. In principle, identities may change across different sessions as long as there is a policy that can establish if the identity is acceptable or not to the peer. However, we note that the public keys

of both the server and the client must always be those defined at the time of password registration.

The client identity (client_identity) and server identity (server_identity) are optional parameters that are left to the application to designate as monikers for the client and server. If the application layer does not supply values for these parameters, then they will be omitted from the creation of the envelope during the registration stage. Furthermore, they will be substituted with client_identity = client_public_key and server_identity = server_public_key during the authenticated key exchange stage.

The advantage to supplying a custom client_identity and server_identity (instead of simply relying on a fallback to client_public_key and server_public_key) is that the client can then ensure that any mappings between client_identity and client_public_key (and server_identity and server_public_key) are protected by the authentication from the envelope. Then, the client can verify that the client_identity and server_identity contained in its envelope match the client_identity and server_identity supplied by the server.

However, if this extra layer of verification is unnecessary for the application, then simply leaving client_identity and server_identity unspecified (and using client_public_key and server_public_key instead) is acceptable.

10.3. Envelope Encryption

The analysis of OPAQUE from [OPAQUE] requires the authenticated encryption scheme used to produce the envelope in the external mode to have a special property called random key-robustness (or key-committing). This specification enforces this property by utilizing encrypt-then-MAC in the construction of the envelope. There is no option to use another authenticated encryption scheme with this specification. (Deviating from the key-robustness requirement may open the protocol to attacks, e.g., [LGR20].) We remark that export_key for authentication or encryption requires no special properties from the authentication or encryption schemes as long as export_key is used only after the envelope is validated, i.e., after the MAC in RecoverCredentials passes verification.

10.4. Export Key Usage

The export key can be used (separately from the OPAQUE protocol) to provide confidentiality and integrity to other data which only the client should be able to process. For instance, if the server is expected to maintain any client-side secrets which require a

password to access, then this export key can be used to encrypt these secrets so that they remain hidden from the server.

10.5. Static Diffie-Hellman Oracles

While one can expect the practical security of the OPRF function (namely, the hardness of computing the function without knowing the key) to be in the order of computing discrete logarithms or solving Diffie-Hellman, Brown and Gallant [BG04] and Cheon [Cheon06] show an attack that slightly improves on generic attacks. For typical curves, the attack requires an infeasible number of calls to the OPRF or results in insignificant security loss; see [I-D.irtf-cfrg-voprf] for more information. For OPAQUE, these attacks are particularly impractical as they translate into an infeasible number of failed authentication attempts directed at individual users.

10.6. Input Validation

Both client and server MUST validate the other party's public key(s) used for the execution of OPAQUE. This includes the keys shared during the offline registration phase, as well as any keys shared during the online key agreement phase. The validation procedure varies depending on the type of key. For example, for OPAQUE instantiations using 3DH with P-256, P-384, or P-521 as the underlying group, validation is as specified in Section 5.6.2.3.4 of [keyagreement]. This includes checking that the coordinates are in the correct range, that the point is on the curve, and that the point is not the point at infinity. Additionally, validation MUST ensure the Diffie-Hellman shared secret is not the point at infinity.

10.7. OPRF Hardening

Hardening the output of the OPRF greatly increases the cost of an offline attack upon the compromise of the credential file at the server. Applications SHOULD select parameters that balance cost and complexity.

10.8. Client Enumeration

Client enumeration refers to attacks where the attacker tries to learn extra information about the behavior of clients that have registered with the server. There are two types of attacks we consider:

1) An attacker tries to learn whether a given client identity is registered with a server, and 2) An attacker tries to learn whether a given client identity has recently completed registration, re-registered (e.g. after a password change), or changed its identity.

OPAQUE prevents these attacks during the authentication flow. The first is done by requiring servers to act with unregistered client identities in a way that is indistinguishable from its behavior with existing registered clients. Servers do this for an unregistered client by simulating a fake CredentialResponse as specified in Section 6.1.2.2. Implementations must also take care to avoid side-channel leakage (e.g., timing attacks) from helping differentiate these operations from a regular server response. Note that server implementations may choose to forego the construction of a simulated credential response message for an unregistered client if these client enumeration attacks can be mitigated through other application-specific means or are otherwise not applicable for their threat model.

Preventing the second type of attack requires the server to supply a credential_identifier value for a given client identity, consistently between the registration response and credential response; see Section 5.1.1.2 and Section 6.1.2.2. Note that credential_identifier can be set to client_identity for simplicity.

In the event of a server compromise that results in a re-registration of credentials for all compromised clients, the oprf_seed value MUST be resampled, resulting in a change in the oprf_key value for each client. Although this change can be detected by an adversary, it is only leaked upon password rotation after the exposure of the credential files, and equally affects all registered clients.

Finally, applications must use the same envelope mode when using this prevention throughout their lifecycle. The envelope size varies between modes, so a switch in mode could then be detected.

OPAQUE does not prevent either type of attack during the registration flow. Servers necessarily react differently during the registration flow between registered and unregistered clients. This allows an attacker to use the server's response during registration as an oracle for whether a given client identity is registered. Applications should mitigate against this type of attack by rate limiting or otherwise restricting the registration flow.

10.9. Password Salt and Storage Implications

In OPAQUE, the OPRF key acts as the secret salt value that ensures the infeasibility of pre-computation attacks. No extra salt value is needed. Also, clients never disclose their passwords to the server, even during registration. Note that a corrupted server can run an exhaustive offline dictionary attack to validate guesses for the client's password; this is inevitable in any aPAKE protocol. (OPAQUE enables defense against such offline dictionary attacks by

distributing the server so that an offline attack is only possible if all - or a minimal number of - servers are compromised [OPAQUE].) Furthermore, if the server does not sample this OPRF key with sufficiently high entropy, or if it is not kept hidden from an adversary, then any derivatives from the client's password may also be susceptible to an offline dictionary attack to recover the original password.

Some applications may require learning the client's password for enforcing password rules. Doing so invalidates this important security property of OPAQUE and is NOT RECOMMENDED. Applications should move such checks to the client. Note that limited checks at the server are possible to implement, e.g., detecting repeated passwords.

11. IANA Considerations

This document makes no IANA requests.

12. References

12.1. Normative References

[I-D.irtf-cfrg-voprf] Davidson, A., Faz-Hernandez, A., Sullivan, N., and C. A. Wood, "Oblivious Pseudorandom Functions (OPRFs) using Prime-Order Groups", Work in Progress, Internet-Draft, draft-irtf-cfrg-voprf-07, 6 July 2021, <https://datatracker.ietf.org/doc/html/draft-irtf-cfrg-voprf-07>.

[RFC2104] Krawczyk, H., Bellare, M., and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication", RFC 2104, DOI 10.17487/RFC2104, February 1997, <https://www.rfc-editor.org/rfc/rfc2104>.

[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>.

[RFC4086] Eastlake 3rd, D., Schiller, J., and S. Crocker, "Randomness Requirements for Security", BCP 106, RFC 4086, DOI 10.17487/RFC4086, June 2005, <https://www.rfc-editor.org/rfc/rfc4086>.

[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>.

12.2. Informative References

[AuCPace] Haase, B. and B. Labrique, "AuCPace: Efficient verifier-based PAKE protocol tailored for the IIoT", http://eprint.iacr.org/2018/286 , 2018.

[BG04] Brown, D. and R. Galant, "The static Diffie-Hellman problem", http://eprint.iacr.org/2004/306 , 2004.

[Boyen09] Boyen, X., "HPAKE: Password Authentication Secure against Cross-Site User Impersonation", Cryptology and Network Security (CANS) , 2009.

[Canetti01] Canetti, R., "Universally composable security: A new paradigm for cryptographic protocols", IEEE Symposium on Foundations of Computer Science (FOCS) , 2001.

[Cheon06] Cheon, J.H., "Security analysis of the strong Diffie-Hellman problem", Euroctypt 2006 , 2006.

[FIPS202] National Institute of Standards and Technology (NIST), "SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions", August 2015, <https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.202.pdf>.

[FK00] Ford, W. and B.S. Kaliski, Jr, "Server-assisted generation of a strong secret from a password", WETICE , 2000.

[GMR06] Gentry, C., MacKenzie, P., and . Z, Ramzan, "A method for making password-based key exchange resilient to server compromise", CRYPTO , 2006.

[HMQV] Krawczyk, H., "HMQV: A high-performance secure Diffie-Hellman protocol", CRYPTO , 2005.

[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", Work in Progress, Internet-Draft, draft-irtf-cfrg-argon2-13, 11 March 2021, <https://datatracker.ietf.org/doc/html/draft-irtf-cfrg-argon2-13>.

[JKKX16] Jarecki, S., Kiayias, A., Krawczyk, H., and J. Xu, "Highly-efficient and composable password-protected secret sharing (or: how to protect your bitcoin wallet online)", IEEE European Symposium on Security and Privacy , 2016.

[keyagreement] Barker, E., Chen, L., Roginsky, A., Vassilev, A., and R. Davis, "Recommendation for pair-wise key-

establishment schemes using discrete logarithm cryptography", National Institute of Standards and Technology report, DOI 10.6028/nist.sp.800-56ar3, April 2018, <https://doi.org/10.6028/nist.sp.800-56ar3>.

[LGR20] Len, J., Grubbs, P., and T. Ristenpart, "Partitioning Oracle Attacks", n.d., <https://eprint.iacr.org/2020/1491.pdf>.

[OPAQUE] Jarecki, S., Krawczyk, H., and J. Xu, "OPAQUE: An Asymmetric PAKE Protocol Secure Against Pre-Computation Attacks", Eurocrypt , 2018.

[RFC2898] Kaliski, B., "PKCS #5: Password-Based Cryptography Specification Version 2.0", RFC 2898, DOI 10.17487/RFC2898, September 2000, <https://www.rfc-editor.org/rfc/rfc2898>.

[RFC2945] Wu, T., "The SRP Authentication and Key Exchange System", RFC 2945, DOI 10.17487/RFC2945, September 2000, <https://www.rfc-editor.org/rfc/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/rfc/rfc5869>.

[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/rfc/rfc7914>.

[RFC8017] Moriarty, K., Ed., Kaliski, B., Jonsson, J., and A. Rusch, "PKCS #1: RSA Cryptography Specifications Version 2.2", RFC 8017, DOI 10.17487/RFC8017, November 2016, <https://www.rfc-editor.org/rfc/rfc8017>.

[RFC8125] Schmidt, J., "Requirements for Password-Authenticated Key Agreement (PAKE) Schemes", RFC 8125, DOI 10.17487/RFC8125, April 2017, <https://www.rfc-editor.org/rfc/rfc8125>.

[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>.

[SIGNAL] "Simplifying OTR deniability", https://signal.org/blog/simplifying-otr-deniability , 2016.

[SPAKE2plus] Shoup, V., "Security Analysis of SPAKE2+", http://eprint.iacr.org/2020/313 , 2020.

Appendix A. Acknowledgments

The OPAQUE protocol and its analysis is joint work of the author with Stas Jarecki and Jiayu Xu. We are indebted to the OPAQUE reviewers during CFRG's aPAKE selection process, particularly Julia Hesse and Bjorn Tackmann. This draft has benefited from comments by multiple people. Special thanks to Richard Barnes, Dan Brown, Eric Crockett, Paul Grubbs, Fredrik Kuivinen, Payman Mohassel, Jason Resch, Greg Rubin, and Nick Sullivan.

Appendix B. Alternate AKE Instantiations

It is possible to instantiate OPAQUE with other AKEs, such as HMQV [HMQV] and SIGMA-I. HMQV is similar to 3DH but varies in its key schedule. SIGMA-I uses digital signatures rather than static DH keys for authentication. Specification of these instantiations is left to future documents. A sketch of how these instantiations might change is included in the next subsection for posterity.

OPAQUE may also be instantiated with any post-quantum (PQ) AKE protocol that has the message flow above and security properties (KCI resistance and forward secrecy) outlined in Section 10. Note that such an instantiation is not quantum-safe unless the OPRF is quantum-safe. However, an OPAQUE instantiation where the AKE is quantum-safe, but the OPRF is not, would still ensure the confidentiality of application data encrypted under session_key (or a key derived from it) with a quantum-safe encryption function.

B.1. HMQV Instantiation Sketch

An HMQV instantiation would work similar to OPAQUE-3DH, differing primarily in the key schedule [HMQV]. First, the key schedule preamble value would use a different constant prefix -- "HMQV" instead of "3DH" -- as shown below.

```
preamble = concat("HMQV",
                  I2OSP(len(client_identity), 2), client_identity,
                  KE1,
                  I2OSP(len(server_identity), 2), server_identity,
                  KE2.inner_ke2)
```

Second, the IKM derivation would change. Assuming HMQV is instantiated with a cyclic group of prime order p with bit length L, clients would compute IKM as follows:

```
u' = (eskU + u \* skU) mod p
IKM = (epkS \* pkS^s)^u'
```

Likewise, servers would compute IKM as follows:

```
s' = (eskS + s \* skS) mod p
IKM = (epkU \* pkU^u)^s'
```

In both cases, u would be computed as follows:

```
hashInput = concat(I2OSP(len(epkU), 2), epkU,
                   I2OSP(len(info), 2), info,
                   I2OSP(len("client"), 2), "client")
u = Hash(hashInput) mod L
```

Likewise, s would be computed as follows:

```
hashInput = concat(I2OSP(len(epkS), 2), epkS,
                   I2OSP(len(info), 2), info,
                   I2OSP(len("server"), 2), "server")
s = Hash(hashInput) mod L
```

Hash is the same hash function used in the main OPAQUE protocol for key derivation. Its output length (in bits) must be at least L.

B.2. SIGMA-I Instantiation Sketch

A SIGMA-I instantiation differs more drastically from OPAQUE-3DH since authentication uses digital signatures instead of Diffie Hellman. In particular, both KE2 and KE3 would carry a digital signature, computed using the server and client private keys established during registration, respectively, as well as a MAC, where the MAC is computed as in OPAQUE-3DH.

The key schedule would also change. Specifically, the key schedule preamble value would use a different constant prefix -- "SIGMA-I" instead of "3DH" -- and the IKM computation would use only the ephemeral key shares exchanged between client and server.

Appendix C. Test Vectors

This section contains real and fake test vectors for the OPAQUE-3DH specification. Each real test vector in Appendix C.1 specifies the configuration information, protocol inputs, intermediate values computed during registration and authentication, and protocol outputs.

Similarly, each fake test vector in Appendix C.2 specifies the configuration information, protocol inputs, and protocol outputs computed during authentication of an unknown or unregistered user. Note that masking_key, client_private_key, and client_public_key are used as additional inputs as described in Section 6.1.2.2. client_public_key is used as the fake record's public key, and masking_key for the fake record's masking key parameter.

All values are encoded in hexadecimal strings. The configuration information includes the (OPRF, Hash, MHF, EnvelopeMode, Group) tuple, where the Group matches that which is used in the OPRF. These test vectors were generated using draft-06 of [I-D.irtf-cfrg-voprf].

C.1. Real Test Vectors

C.1.1. OPAQUE-3DH Real Test Vector 1

C.1.1.1. Configuration

```
OPRF: 0001
Hash: SHA512
MHF: Identity
KDF: HKDF-SHA512
MAC: HMAC-SHA512
EnvelopeMode: 01
Group: ristretto255
Context: 4f50415155452d504f43
Nh: 64
Npk: 32
Nsk: 32
Nm: 64
Nx: 64
Nok: 32
```

C.1.1.2. Input Values

```
oprf_seed: 5c4f99877d253be5817b4b03f37b6da680b0d5671d1ec5351fa61c5d82
eab28b9de4c4e170f27e433ba377c71c49aa62ad26391ee1cac17011d8a7e9406657c
8
credential_identifier: 31323334
password: 436f7272656374486f72736542617474657279537461706c65
envelope_nonce: 71b8f14b7a1059cdadc414c409064a22cf9e970b0ffc6f1fc6fdd
539c4676775
masking_nonce: 54f9341ca183700f6b6acf28dbfe4a86afad788805de49f2d680ab
86ff39ed7f
server_private_key: 16eb9dc74a3df2033cd738bf2cfb7a3670c569d7749f284b2
b241cb237e7d10f
server_public_key: 18d5035fd0a9c1d6412226df037125901a43f4dff660c0549d
402f672bcc0933
server_nonce: f9c5ec75a8cd571370add249e99cb8a8c43f6ef05610ac6e354642b
f4fedbf69
client_nonce: 804133133e7ee6836c8515752e24bb44d323fef4ead34cde967798f
2e9784f69
server_keyshare: 6e77d4749eb304c4d74be9457c597546bc22aed699225499910f
c913b3e90712
client_keyshare: f67926bd036c5dc4971816b9376e9f64737f361ef8269c18f69f
1ab555e96d4a
server_private_keyshare: f8e3e31543dd6fc86833296726773d51158291ab9afd
666bb55dce83474c1101
client_private_keyshare: 4230d62ea740b13e178185fc517cf2c313e6908c4cd9
fb42154870ff3490c608
blind_registration: c62937d17dc9aa213c9038f84fe8c5bf3d953356db01c4d48
acb7cae48e6a504
blind_login: b5f458822ea11c900ad776e38e29d7be361f75b4d79b55ad74923299
bf8d6503
```

C.1.1.3. Intermediate Values

```
client_public_key: 100f6f945b57e4a316e46ed7169b6e9e533c35b29128368a9a
40534b09227428
auth_key: 0ee186ac3a0fe0ec45d36c7cc9786934918a58d6a1abce6842a2b7bd0ec
1c0626e64d887622e8937e987bfbe042f904728966e121b01c739c8dbe66beb6241eb
randomized_pwd: 22f5e31fbbbf4649f77ebfc92a2ef555fc30a09edc903123d978d
e3ca356b85ce2120b0d2735bd772011ecb573e614cd7b1aeeb86ca0ac6b8732c33cdf
7a6816
envelope: 71b8f14b7a1059cdadc414c409064a22cf9e970b0ffc6f1fc6fdd539c46
767754ed20356eabd714970fc0058aad92414cc26131c7c3f9a0f5baa76c8fc410984
0921d83e26786e9c5b6a1cbc4458925fd65415547ffd61d2afbcb593a3a82fb1
handshake_secret: 9dc2e984200002626dac10f89b9d2efe967f68c8eb19612dd6c
2592d531ca5bd443c0548b8cef946f6e99998d810838c0ee99219fe13052beae3dc2c
b157c991
server_mac_key: 66739820f04e1cddc7090b05d510baf0de4273ebd9e26f6da961b
687dfdc05c6a68a93f81fd6ed7ff19c1c7f95cc76ceb2c680ed4ab2d6a5eb53088abd
3f7a36
client_mac_key: e0d836ea8e158a50a7cc610ca27eee2fb92187aa7273cb46ee21d
230be82e7efddf742fe127bc95eb6e034a01efe38ae3c4240b0d0f7fa395dfa52032a
0c1cb5
oprf_key: 3f76113135e6ca7e51ac5bb3e8774eb84709ad36b8907ec8f7bc3537828
71906
```

C.1.1.4. Output Values

```
registration_request: 76cc85628d5ac0e01de4ede72479d607490e7f58b94578d
b7a0606d74bc58b03
registration_response: 865f3305ff73be7388313e7a74b5fc277a165ff2895f92
60391057b84c7bc72718d5035fd0a9c1d6412226df037125901a43f4dff660c0549d4
02f672bcc0933
registration_upload: 100f6f945b57e4a316e46ed7169b6e9e533c35b29128368a
9a40534b09227428dc3b0057603d1c23df7e6f239984604c4b0dfa111528ab0ba3c7f
6ab1ceb11d10aa85433f63bbf30b9b0ae8951653bcd3beb12aa61cf942e6e5b442282
0d810871b8f14b7a1059cdadc414c409064a22cf9e970b0ffc6f1fc6fdd539c467677
54ed20356eabd714970fc0058aad92414cc26131c7c3f9a0f5baa76c8fc4109840921
d83e26786e9c5b6a1cbc4458925fd65415547ffd61d2afbcb593a3a82fb1
KE1: e47c1c5e5eed1910a1cbb6420c5edf26ea3c099aaaedcb03599fc311a724d84f
804133133e7ee6836c8515752e24bb44d323fef4ead34cde967798f2e9784f69f6792
6bd036c5dc4971816b9376e9f64737f361ef8269c18f69f1ab555e96d4a
KE2: 9692d473e0bde7a1fbb6d2c0e4001ccc58902102857d0e67e5fa44f4b902b17f
54f9341ca183700f6b6acf28dbfe4a86afad788805de49f2d680ab86ff39ed7f7b11e
e2cb784efa8e6cbbb9cc6b52b16290e3906235d71b773534c3da1575a00708219fa81
05b3d2a1292d58d6ea6b0e464c752df6f957a9e34a66de7e5d44db0e48648d414f7b1
54e52d5f664c1b88dd42a8117b048fb26428a3b86885d2157a7136708494fd92f50c1
c3f63bd60d0b458254ac54c6e64841be63f1c4459d3bf9c5ec75a8cd571370add249e
99cb8a8c43f6ef05610ac6e354642bf4fedbf696e77d4749eb304c4d74be9457c5975
46bc22aed699225499910fc913b3e907129491571e3137ff950f905e6c83db79e5103
bd8b7c27799d1eac8d8c57fd2f0b913e81411c02bc722b30b4eb4ee3a53fc1b21232b
4218eb0ed00996dc8c841a96
KE3: 5a4e48d857196ce709b054f7be4e3973f0892abfafc030762bd4be5dbeb342af
b5bf1fd24a7b355f556f9d53f146ebb3729ae693f69fc1f7862a0c11c8ee7c9d
export_key: 47d742be256471ec7a7b0ebc022d6ca016b022a7dcbdd41fa1b6dbfcd
6f88285aee60db87e7c5e5aff87b55904b07137b3d85648bb62d70a18954dd1c66cdd
c2
session_key: e132b5c83951919bc0f22aa4e3b12b4af81831ccf770f318d11c8c0a
854fddf6bbeee72e24114402ff9012cf117bac95dda241b24d055f7ce2750e6975fa2
22d
```

C.1.2. OPAQUE-3DH Real Test Vector 2

C.1.2.1. Configuration

```
OPRF: 0001
Hash: SHA512
MHF: Identity
KDF: HKDF-SHA512
MAC: HMAC-SHA512
EnvelopeMode: 01
Group: ristretto255
Context: 4f50415155452d504f43
Nh: 64
Npk: 32
Nsk: 32
Nm: 64
Nx: 64
Nok: 32
```

C.1.2.2. Input Values

```
client_identity: 616c696365
server_identity: 626f62
oprf_seed: db5c1c16e264b8933d5da56439e7cfed23ab7287b474fe3cdcd58df089
a365a426ea849258d9f4bc13573601f2e727c90ecc19d448cf3145a662e0065f157ba
5
credential_identifier: 31323334
password: 436f7272656374486f72736542617474657279537461706c65
envelope_nonce: d0c7b0f0047682bd87a87e0c3553b9bcdce7e1ae3348570df20bf
2747829b2d2
masking_nonce: 30635396b708ddb7fc10fb73c4e3a9258cd9c3f6f761b2c227853b
5def228c85
server_private_key: eeb2fcc794f98501b16139771720a0713a2750b9e528adfd3
662ad56a7e19b04
server_public_key: 8aa90cb321a38759fc253c444f317782962ca18d33101eab2c
8cda04405a181f
server_nonce: 3fa57f7ef652185f89114109f5a61cc8c9216fdd7398246bb7a0c20
e2fbca2d8
client_nonce: a6bcd29b5aecc3507fc1f8f7631af3d2f5105155222e48099e5e608
5d8c1187a
server_keyshare: ae070cdffe5bb4b1c373e71be8e7d8f356ee5de37881533f1039
7bcd84d35445
client_keyshare: 642e7eecf19b804a62817486663d6c6c239396f709b663a4350c
da67d025687a
server_private_keyshare: 0974010a8528b813f5b33ae0d791df88516c8839c152
b030697637878b2d8b0a
client_private_keyshare: 03b52f066898929f4aca48014b2b97365205ce691ee3
444b0a7cecec3c7efb01
blind_registration: a66ffb41ccf1194a8d7dda900f8b6b0652e4c7fac4610066f
e0489a804d3bb05
blind_login: e6f161ac189e6873a19a54efca4baa0719e801e336d929d35ca28b5b
4f60560e
```

C.1.2.3. Intermediate Values

client_public_key: c8c9f3a9adab66971bbdb230bd44512741a489e333624186ac
a1b0e967011e06
auth_key: 494f1326c65c057e301f15e619b9e3de553c77132987828ba20026062da
a1f18d516ac2e37b1dfc296e21137623856fb3ccba48cc511f143110944848764dfb7
randomized_pwd: a4853e726d14efb03c35686ee2bc67665d02bdccb0c4c02523bf4
e1398e78b1094195a082b5ebb1b62ac75d06711643e9990c2be0071a42bc21a2b766b
787eac
envelope: d0c7b0f0047682bd87a87e0c3553b9bcdce7e1ae3348570df20bf274782
9b2d21a6d25b3a1b8a541f47fc4bbe5783c2cc61c77ed389280a05ed2ef8c2c2d03fe
2066bd22ea959c7bb14b5259ee136e86f4d956fbaa2af2f6a326785f21262909
handshake_secret: aacad16f4fb6ff90e3a6afb4e430a550000f0351cd30aab930c
3118c8aebbf8bcaa5252809b065d19a26ab475214a1c4dd60c28e91ce310b5a79968d
bb87938c
server_mac_key: 58edbb9aa5489765c102470a5287b5036893b8cea58ecc836bd0c
df4a818f41d6cd58c46d64db3548774f2cc651d10f8e152a7fdd47493c90a28b1106c
8d329b
client_mac_key: 57be496bd6fa74ecfed7a8c30a9ee085f3e56dbf0d5c18c9ad8c3
8799f0ededd9a8af85e59276935da7bc3ddd2a8cf4fa9ea723ea70b6dba1db2872815
27944a
oprf_key: 531b0c7b0a3f90060c28d3d96ef5fecf56e25b8e4bf71c14bc770804c3f
b4507

C.1.2.4. Output Values

```
registration_request: ec2927a03ced1220168b6d5a54f0372f813ced8ad3673d5
1dee92d2cbfee500c
registration_response: f6e244e131f8cd14bc37a856a933c91128b2498c06540d
2dba3a197ed7d8bd778aa90cb321a38759fc253c444f317782962ca18d33101eab2c8
cda04405a181f
registration_upload: c8c9f3a9adab66971bbdb230bd44512741a489e333624186
aca1b0e967011e069fabb8544108ec64de2b992935dd5fd9a98441412ccf724bf4853
c28749d9fd33fb1824b964f616a7fce654e05bb15133bd4a69441dcbfe6a02b8a546e
1b32dbd0c7b0f0047682bd87a87e0c3553b9bcdce7e1ae3348570df20bf2747829b2d
21a6d25b3a1b8a541f47fc4bbe5783c2cc61c77ed389280a05ed2ef8c2c2d03fe2066
bd22ea959c7bb14b5259ee136e86f4d956fbaa2af2f6a326785f21262909
KE1: d0a498e621d3ff7a011b37166a63ef40fe268f93c7d75a467eea42a98c0a490d
a6bcd29b5aecc3507fc1f8f7631af3d2f5105155222e48099e5e6085d8c1187a642e7
eecf19b804a62817486663d6c6c239396f709b663a4350cda67d025687a
KE2: e6d0c01cdc14f7bfdb38effbd63394f6304b47c2dc26fd510ecdbf471486b972
30635396b708ddb7fc10fb73c4e3a9258cd9c3f6f761b2c227853b5def228c8563625
e7cfa2857bdc95991bfccc09b69ed53fd5f173389f4d8786b261b6dfd2fc7c18968dd
1be8cdb52b1ca691d8d27ad655e6c78a6ef67c2ad43899259b060706f7d4f4946f97b
d283c009a736227e9c1913b394d0d88419c6463970c0c2887bd5890eac805e8657903
f7f8887f5eab9e700414af99bbabe3b6594418e2a3723fa57f7ef652185f89114109f
5a61cc8c9216fdd7398246bb7a0c20e2fbca2d8ae070cdffe5bb4b1c373e71be8e7d8
f356ee5de37881533f10397bcd84d354450f1409ae0d487a21921d7edbfe34f9e16a8
ac94f6f83b0d4a78c1fe567ea4bda594d375917e22fe0fc203773bf728eff0309f259
4ac3f4dc2cf066f4febcca0e
KE3: c3dc494d69ef860b348714fd67a3e6490e5afd493d7212729a96d61e9502f7a8
202a18be10b513f236b3e7e2e2f62ff25a61525991c81c030836cb1933b7a13d
export_key: 7f2e5b749ec5f6ab34663655184f3653275aafd5db070b6aac6afd80a
78309a8ec0f97a2f2cfcc7a971983a914ead081a8a642b65d298c579d3526d2219381
8d
session_key: a07bb20c5ad960af8caf83b112a008a61e10e2475456fd0620a11ea5
a75706750b72c4943c3985cae5f31969dea9c74192c8bf601e2d062fcb141c89234ff
7be
```

C.1.3. OPAQUE-3DH Real Test Vector 3

C.1.3.1. Configuration

```
OPRF: 0003
Hash: SHA256
MHF: Identity
KDF: HKDF-SHA256
MAC: HMAC-SHA256
EnvelopeMode: 01
Group: P256_XMD:SHA-256_SSWU_RO_
Context: 4f50415155452d504f43
Nh: 32
Npk: 33
Nsk: 32
Nm: 32
Nx: 32
Nok: 32
```

C.1.3.2. Input Values

```
oprf_seed: 77bfc065218c9a5593c952161b93193f025b3474102519e6984fa64831
0dd1bf
credential_identifier: 31323334
password: 436f7272656374486f72736542617474657279537461706c65
envelope_nonce: 2527e48c983deeb54c9c6337fdd9e120de85343dc7887f00248f1
acacc4a8319
masking_nonce: cb792f3657240ce5296dd5633e7333531009c11ee6ab46b6111f15
6d96a160b2
server_private_key: 87ef09986545b295e8f5bbbaa7ad3dce15eb299eb2a5b3487
5ff421b1d63d7a3
server_public_key: 025b95a6add1f2f3d038811b5ad3494bed73b1e2500d8dadec
592d88406e25c2f2
server_nonce: 8018e88ecfc53891529278c47239f8fe6f1be88972721898ef81cc0
a76a0b550
client_nonce: 967fcded96ed46986e60fcbdf985232639f537377ca3fcf07ad4899
56b2e9019
server_keyshare: 0242bc29993976185dacf6be815cbfa923aac80fad8b7f020c9d
4f18e0b6867a17
client_keyshare: 03358b4eae039953116889466bfddeb40168e39ed83809fd5f0d
5f2de9c5234398
server_private_keyshare: b1c0063e442238bdd89cd62b4c3ad31f016b68085d25
f85613f5838cd7c6b16a
client_private_keyshare: 10256ab078bc1edbaf79bee4cd28dd9db89179dcc921
9bc8f388b533f5439099
blind_registration: d50e29b581d716c3c05c4a0d6110b510cb5c9959bee817fde
b1eabd7ccd74fee
blind_login: 503d8495c6d04efaee8370c45fa1dfad70201edd140cec8ed6c73b5f
cd15c478
```

C.1.3.3. Intermediate Values

```
client_public_key: 02680493263d3bc4c7af455ba1219fd9bbe329fd0c2a0248e8
7321ded8ff17b386
auth_key: 570a8105a7d86679b4c9d009edc9627af6b17e8b2d2f0d50cbd13ea8a00
82cd7
randomized_pwd: 04f1615bc400765f22f7af1277a0814b5665ad1d4ef9bf1829880
2a0f6b4636b
envelope: 2527e48c983deeb54c9c6337fdd9e120de85343dc7887f00248f1acacc4
a8319890e251c4b6397fb35900ff46ae1df1e86eed2d23005c6b9c61caa4e12af8bf5
handshake_secret: 56212ac60eca9f917f6a4ce6aefe762743da701b008ec986cd1
87ff75df5df84
server_mac_key: 8c7f132b6cd9a7e4ce9171cd469d02dc1ab0e8d96f4e1ddca1718
55fc723203f
client_mac_key: 348cb4526417423090d386ce43459d652a6489122e27d3953ed47
5b3ab9cd336
oprf_key: d153d662a1e7dd4383837aa7125685d2be6f8041472ecbfd610e46952a6
a24f1
```

C.1.3.4. Output Values

```
registration_request: 0325768a660df0c15f6f2a1dcbb7efd4f1c92702401edf3
e2f0742c8dce85d5fa8
registration_response: 0244211a4d2a067f7a61ed88dff6764856d347465f330d
0e15502700afd1865911025b95a6add1f2f3d038811b5ad3494bed73b1e2500d8dade
c592d88406e25c2f2
registration_upload: 02680493263d3bc4c7af455ba1219fd9bbe329fd0c2a0248
e87321ded8ff17b3868efb26f2bb390fd23b90c49ae680c4560fbd2b3c4f32891505c
ad7d95b7bc58e2527e48c983deeb54c9c6337fdd9e120de85343dc7887f00248f1aca
cc4a8319890e251c4b6397fb35900ff46ae1df1e86eed2d23005c6b9c61caa4e12af8
bf5
KE1: 03884e56429f1ee53559f2e244392eb8f994fd46c8fd9ffdd24ac5a7af963a66
3b967fcded96ed46986e60fcbdf985232639f537377ca3fcf07ad489956b2e9019033
58b4eae039953116889466bfddeb40168e39ed83809fd5f0d5f2de9c5234398
KE2: 0383fff1b3e8003723dff1b1f90a7934a036bd6691aca0366b07a100bf2bb3dc
2acb792f3657240ce5296dd5633e7333531009c11ee6ab46b6111f156d96a160b23b6
a5ff1ce8035a1dca4776f32f43c7ce626d796da0f27fc9897522fc1fab70d2fb443d8
2a4333770057e929c2f9977d40a64e8b4a5a553d25a8b8392b4adbf0a0a6e87f26165
0d04084823b23b07d351e3b947778a43859be3ba218b22d054edf8018e88ecfc53891
529278c47239f8fe6f1be88972721898ef81cc0a76a0b5500242bc29993976185dacf
6be815cbfa923aac80fad8b7f020c9d4f18e0b6867a171e7fda886b9fd9f3bc9e37c4
04ca07f7a5c9a6f98df20a5cac42371162731faa
KE3: 86a57a3e1a2e537ea667031091c025cb539826dbbb1756683220dd239d4a7bff
export_key: a83a3fe26af0dadb63d15ed808a4dc2edb57f45212554ecc1af5e0273
50651de
session_key: e26d54798ce8a66fb415cb67f4d87647dcd3d8aa79a7ab6a5f701b82
f037b1e3
```

C.1.4. OPAQUE-3DH Real Test Vector 4

C.1.4.1. Configuration

```
OPRF: 0003
Hash: SHA256
MHF: Identity
KDF: HKDF-SHA256
MAC: HMAC-SHA256
EnvelopeMode: 01
Group: P256_XMD:SHA-256_SSWU_RO_
Context: 4f50415155452d504f43
Nh: 32
Npk: 33
Nsk: 32
Nm: 32
Nx: 32
Nok: 32
```

C.1.4.2. Input Values

```
client_identity: 616c696365
server_identity: 626f62
oprf_seed: 482123652ea37c7e4a0f9f1984ff1f2a310fe428d9de5819bf63b3942d
be09f9
credential_identifier: 31323334
password: 436f7272656374486f72736542617474657279537461706c65
envelope_nonce: 75c245690f9669a9af5699e8b23d6d1fa9e697aeb4526267d942b
842e4426e42
masking_nonce: 5947586f69259e0708bdfab794f689eec14c7deb7edde68c816451
56cf278f21
server_private_key: c728ebf47b1c65594d77dab871872dba848bdf20ed725f0fa
3b58e7d8f3eab2b
server_public_key: 029a2c6097fbbcf3457fe3ff7d4ef8e89dab585a67dfed0905
c9f104d909138bae
server_nonce: 581ac468101aee528cc6b69daac7a90de8837d49708e76310767cbe
4af18594d
client_nonce: 46498f95ec7986f0602019b3fbb646db87a2fdbc12176d4f7ab74fa
5fadace60
server_keyshare: 022aa8746ab4329d591296652d44f6dfb04470103311bacd7ad5
1060ef5abac41b
client_keyshare: 02a9f857ad3eabe09047049e8b8cee72feea2acb7fc487777c0b
22d3add6a0e0c0
server_private_keyshare: 48a5baa24274d5acc5e007a44f2147549ac8dd675564
2638f1029631944beed4
client_private_keyshare: 161e3aaa50f50e33344022969d17d9cf4c88b7a9eec4
c36bf64de079abb6dc7b
blind_registration: 9280e203ef27d9ef0d1d189bb3c02a66ef9a72d48cca6c1f9
afc1fedea22567c
blind_login: 4308682dc1bdab92ff91bb1a5fc5bc084223fe4369beddca3f1640a6
645455ad
```

C.1.4.3. Intermediate Values

client_public_key: 02e89507b3a1a946e8096cd7e1e8fbc31e2dd39fecc49580ed
2659262c08ea33eb
auth_key: 76cba5b349c60c5a19ab06b70a3191d3418318b5a203fd298b18a0eda53
efd1a
randomized_pwd: 74649c9c7b0d7436c4873984732fe45e19dabd1a96d7e9175468a
85ed16bea65
envelope: 75c245690f9669a9af5699e8b23d6d1fa9e697aeb4526267d942b842e44
26e423938d818ea53f58fdaab8541765d5171e99b1bdc2c63e8e1eaf62d3a60aacabe
handshake_secret: 22f608959308cb4dff55cf77c006ea8e9bc66df75d7076a927a
3d21d3fce5562
server_mac_key: 76e1415cfcf0ff271533fdd4ce4fffb4110ba1ff4aa9a02a1734d
9ae0e0ce47a
client_mac_key: 9341b0b36ac36875910cd1260cd8dc6d6cd58e0fb6503fece6524
11b6f627bf7
oprf_key: f14e1fc34ba1218bfd3f7373f036889bf4f35a8fbc9e8c9c07ccf2d2388
79d9c

C.1.4.4. Output Values

registration_request: 02792b0f4670aced5970a68b01bb951004ccad962159be4
b6783170c9ad68f6052
registration_response: 03cc3491b4bcb3e4804f3eadbc6a04c8fff18cc9ca5a4f
eeb577fdfebd71f5060f029a2c6097fbbcf3457fe3ff7d4ef8e89dab585a67dfed090
5c9f104d909138bae
registration_upload: 02e89507b3a1a946e8096cd7e1e8fbc31e2dd39fecc49580
ed2659262c08ea33eb260603b2690f3d466fb0b747e256283bed94836ac98c10d4588
1372046d3b1e875c245690f9669a9af5699e8b23d6d1fa9e697aeb4526267d942b842
e4426e423938d818ea53f58fdaab8541765d5171e99b1bdc2c63e8e1eaf62d3a60aac
abe
KE1: 02fe96fc48d9fc921edd8e92ada581cbcc2a65e30962d0002ea5242f5baf627f
f646498f95ec7986f0602019b3fbb646db87a2fdbc12176d4f7ab74fa5fadace6002a
9f857ad3eabe09047049e8b8cee72feea2acb7fc487777c0b22d3add6a0e0c0
KE2: 035115b21dde0992cb812926d65c7dccd5e0f8ffff573da4a7c1e603e0e40827
895947586f69259e0708bdfab794f689eec14c7deb7edde68c81645156cf278f21cef
3adc4e524db33258c5774efaec59750eaf3755a2dfa194ec593ce41a7a17f889978a2
f97ced10bd1592793497e58b5d05a02ebf003f8a8949a2f8a22a09e4d1b8ba19c9e77
4b6f31545ac4c02aba4ad8e26b4f43d65319f8d1c5a5a04668d4b581ac468101aee52
8cc6b69daac7a90de8837d49708e76310767cbe4af18594d022aa8746ab4329d59129
6652d44f6dfb04470103311bacd7ad51060ef5abac41bbe51a3c1deeeab8ded9273ad
681001416cbb6d1f0976548f36d1ddb1d3b1f948
KE3: 5770a1ce912fecf1fa339fe1646b2abfe8dd683767c885a0f1dedba8dfab653e
export_key: 5b92e3454d59062460a87ad2ff6546d862f722c6fbd7678a0997b3c9d
c61e9a0
session_key: 3a04636e2c14b4ef3a01070a2ff129cd2248318d8b85d6c4368f5115
0f0348ff

C.1.5. OPAQUE-3DH Real Test Vector 5

C.1.5.1. Configuration

```
OPRF: 0001
Hash: SHA512
MHF: Identity
KDF: HKDF-SHA512
MAC: HMAC-SHA512
EnvelopeMode: 02
Group: ristretto255
Context: 4f50415155452d504f43
Nh: 64
Npk: 32
Nsk: 32
Nm: 64
Nx: 64
Nok: 32
```

C.1.5.2. Input Values

```
oprf_seed: 98ee70b2c51d3e89d9c08b00889a1fa8f3947a48dac9ad994e946f408a
2c31250ee34f9d04a7d85661bab11c67048ecfb7a68c657a3df87cff3d09c6af9912a
1
credential_identifier: 31323334
password: 436f7272656374486f72736542617474657279537461706c65
envelope_nonce: 13e74aa6263b562b9764bfbcbbe54081383f3057c26fead2a8b3d
5910e25fd29
masking_nonce: 8837b6c0709160251cbebe0d55e4423554c45da7a8952367cf336e
b623379e80
client_private_key: 74f3dec61a0126d02343c90d62c0399789041c71e44d15c05
bc8182bc764e903
server_private_key: 030c7634ae24b9c6d4e2c07176719e7a246850b8e019f1c71
a23af6bdb847b0b
server_public_key: 1ac449e9cdd633788069cca1aaea36ea359d7c2d493b254e5f
fe8d64212dcc59
server_nonce: ff26a6386c0a4077f512138e2203f247d56cbe900310cd43b4a55e4
c54231cc6
client_nonce: 077adba76f768fd0979f8dc006ca297e7954ebf0e81a893021ee24a
cc35e1a3f
server_keyshare: 046a0fce623dc08f253b239bfc96850e7ed02dc87f3e29830ccd
128aaa365840
client_keyshare: f27ab6cb55389234dd3f045713b6fc7c1f3de0e84140ee8b07be
e138ba587d79
server_private_keyshare: 26d7584722349b4670cbddb025f5e3e830b8bff4e482
80bab53db37c34bf0b06
client_private_keyshare: 32f2e562f6ca9ffa8add0c2c5b8b2ffe8c39667010ba
488071c889448f2a2f06
blind_registration: 21c97ffc56be5d93f86441023b7c8a4b629399933b3f845f5
852e8c716a60408
blind_login: f3fbfd52953b95ae7567d2590ecd8b0defe05a0321ec8230137157c1
6cc06b09
```

C.1.5.3. Intermediate Values

client_public_key: cef8fa52ed24697b25418aa4703deb80a1e8f2caca405e2aaf
6758718ac7e33f
auth_key: 760cf1c679e1599d4ac3a8278f01c4494750c4038a713e7854b1d72af8f
9dd754e4e2819456f9d8a38b178d6472d4e893aff415ad93a7546c1dacb9b19fc9d25
randomized_pwd: 6df3b91dd1cf81df3b0b72998374dc446a8ade0fdc7eea49cb7a6
21ae27b90b883d94e7d528378e8f665977b5e03909511ee729c36f16fd6aba83613d8
c25142
envelope: 13e74aa6263b562b9764bfbcbbe54081383f3057c26fead2a8b3d5910e2
5fd293005c607b1e9e20601905d07de4b195e9f3812a14faa21bc7ec6b3aaeb4ff833
e36d0ea6c1c575161a16d3de1052fd1bca6d387a6dd0947d5284f1c27010dbc771b93
e986d552e1c808c4506d430532d01cfa6c577edbe2230f2b1d68899af19
handshake_secret: 5d37e08f982c39e4edf75d23d6c8e412c56eff600a1d2baf02e
3c8d67c41490ee34477e2bb6a58f5c555ac2a1292bdbfbbea9940d958dafa968ca6c4
a70b6933
server_mac_key: fb738205ec3bf99dce71a809f61c68f52d507260abb3ad67c7d7c
c6f6b673affce4b5166c965b0d1ad37aa0429ee6cc09c0063c5eff062e53db4d2adc8
fb7d4f
client_mac_key: f6f3c6826a01c16003ae99fa38a9efd63459e14a57071f3177f44
c3d4cd726db61c0de0297bb5c8ebb9c26cc9474899d92b453e8232e3feab9e8aba119
71ee46
oprf_key: ea5421a9d7d562bce7d5541d716a45e2e28430c33b50e2c54dee998e328
35503

C.1.5.4. Output Values

```
registration_request: f0c2f72afd000afa4cfde5eb4122f42c4082332d8783204
6dc4f7f9691e86c3f
registration_response: f033a257a3f77f2df98a46036738c20706b4d70015b59f
1c74a2606d03725b121ac449e9cdd633788069cca1aaea36ea359d7c2d493b254e5ff
e8d64212dcc59
registration_upload: cef8fa52ed24697b25418aa4703deb80a1e8f2caca405e2a
af6758718ac7e33fb3ab72316071ee2799c26865b8fbdbc87a0d861cc0b9f5bd9cb1e
a97569472896ee0bfc2a25f9d2031f409c323df56a5c2bd9c6317874c5a17935dfd79
26b2f413e74aa6263b562b9764bfbcbbe54081383f3057c26fead2a8b3d5910e25fd2
93005c607b1e9e20601905d07de4b195e9f3812a14faa21bc7ec6b3aaeb4ff833e36d
0ea6c1c575161a16d3de1052fd1bca6d387a6dd0947d5284f1c27010dbc771b93e986
d552e1c808c4506d430532d01cfa6c577edbe2230f2b1d68899af19
KE1: dea559cab899a46d045370d26c2551e74f1d0da2b090ca860c1379241d5f2900
077adba76f768fd0979f8dc006ca297e7954ebf0e81a893021ee24acc35e1a3ff27ab
6cb55389234dd3f045713b6fc7c1f3de0e84140ee8b07bee138ba587d79
KE2: 8433d62cb0db1a06942ce70567ea1b28ce01972d577fcc997b3951deadda5677
8837b6c0709160251cbebe0d55e4423554c45da7a8952367cf336eb623379e80db22c
f683b95f135f5ca1a9c2ee5af15577eb567303c95f7ea18355fb7f8c7cd209f0c156b
232d143d5b7aec3c9a495a5223017d18e4281e41b89456803f8eca42b419daee26218
1d43ad1794b467043bd882f1a19f3155d7d30fd1b5cdc2dd13aece79fd2f8905da3ab
1a98c3a0ef50995b4e30a8cc3e9882384e9576fb85ee4c1dabbbe1dd967f2ec96e6ba
df1a20a2ce6d583747feabd690992a44f5a4798ff26a6386c0a4077f512138e2203f2
47d56cbe900310cd43b4a55e4c54231cc6046a0fce623dc08f253b239bfc96850e7ed
02dc87f3e29830ccd128aaa365840ccaba8d6b2056626d9ac6b8983148f7685ebb5f0
31823e575e3ac75f4bc88086084f6f1a6ae2662a5079f6167ec55f52811a81e6d0a3c
8fafa337d66ac737d55
KE3: ce0d378f6590de86be61608c42a7a1379c58a0a8cf6f1ce6af0164fff7a65162
265f78e1ddbf056af34694b609d59ece78c101a0dcbb5ce2e0ff3f5cc39ea1c9
export_key: a656dd5acbaae4d7149d2edc825bd23b7b3505218c2dd3240f16c2a60
29d9d2837c8beed8467491e936fed5139a750bc979902017156aed40b6b2f8eae6c04
32
session_key: 0940f92334f093f39da5a9f36b76a137c4446796d5cb9f272a3d85e7
01f3b0160301bde0761dc96f8a8da0241c9f88afdc5d614d270c86d8249b0f425fa19
39e
```

C.1.6. OPAQUE-3DH Real Test Vector 6

C.1.6.1. Configuration

```
OPRF: 0001
Hash: SHA512
MHF: Identity
KDF: HKDF-SHA512
MAC: HMAC-SHA512
EnvelopeMode: 02
Group: ristretto255
Context: 4f50415155452d504f43
Nh: 64
Npk: 32
Nsk: 32
Nm: 64
Nx: 64
Nok: 32
```

C.1.6.2. Input Values

client_identity: 616c696365
server_identity: 626f62
oprf_seed: 41723264b8bb2700cdd47e339d95404519f2fb3da58c93d84cbb4d51de
6757a31919382ba65c10e80cbb7f50a43e32782b08f8bee3ffaba39407660179105ac
5
credential_identifier: 31323334
password: 436f7272656374486f72736542617474657279537461706c65
envelope_nonce: 86e2b0916361fb6a69f9a097e3ef2f83f8fd5f95cc79432eabf3e
01f0020bac6
masking_nonce: f551feeed04ea7967c2d6c8847b6ca8bc09eced6848f0a73208e71
e3f1b688e6
client_private_key: f51ea8d2b529ab39f6e6c883a14c38941c81115856ef6ef92
c0a0aacf1449e05
server_private_key: 5791eacfa2980bc0807bd327239b53f4370eb21d5b306257a
82104eaaab43f01
server_public_key: 846c28da13144e5908f1db4c8ba2c848ac34ae6b9a8855dcdb
08d5ecfb000d73
server_nonce: f35483b457e208972f1cbffbb70249fa31064a8883002d3b8024a02
3b4feac41
client_nonce: e0d04374ad9a276620c681abfca7bdb432e63509e5ec96ed2ec5542
f6fc7db23
server_keyshare: d856f10de218fc035faad7dddd3f3f01e2f27ff589ebbe605fa8
e4659652fa52
client_keyshare: 1c48ce65e44088bb7f7503f7ad7f3a050177b727e9b490aac3d4
e639be06e157
server_private_keyshare: 381fbcf62f24a60451e6ca6a9b8cffdacb40f507334b
4384173539b287a94e0d
client_private_keyshare: d7b1baf64ffc214733c46987fd2a0e91bb1f30f8ac55
70d636293675879c6900
blind_registration: a8fe808852933dec78a535ad1cd3a6e64b82b5302a5a99f16
66c3cce1c4f2408
blind_login: ec58b546968ed476e1a55a9dff9c80cc9c56cc9021a662d5cb7d5d37
109b760b

C.1.6.3. Intermediate Values

client_public_key: 262321548e1fbb06525167181fb3ca34409410120c7feb4144
26b4176f519264
auth_key: b89cd540fb9e954e7a622723120bcb9f82c5006570ca36d44424dcc4805
e4f7b90ce47c31e8a8c809fc666006a377ce68b19431378c68776ddbfa0617490f16a
randomized_pwd: b2828519c684910131dad09652c589fc77a307a7560fe78d2c8c7
02c8eeece38f470e695185e45be762c6e4732980715e535f411f02b916e18bf3d9f0c
249e69
envelope: 86e2b0916361fb6a69f9a097e3ef2f83f8fd5f95cc79432eabf3e01f002
0bac600880f67a9cc7800800677a52179746eb1426db885097a0d0a05f93d64f93ff1
498d75c646f7f836c53ca43e93ddccd0e8225e3d608d01e8620a4a71b1a5ee9be6219
79afc8c0114626fdd78fb1ee4bf7537a1743964b364b02ff7851284ae00
handshake_secret: 3d925e90f1b621e6e6c896c87d565a8ff6f6fc9ea375b46cd97
edc0ea8a576eb1e6bd1d202ff0f767302f21812b4f99186025f237f8f7778e8e27768
9ddfda11
server_mac_key: 40628b867618de054b613cb1d6563dba6dded5ba4dd10f6113b22
2744aa5252b68d5fc07442c30be318ff2c675e3d4cee5c76d66aa468c5b3c2f69ed7e
e2a8ee
client_mac_key: 3d3d749f7406a8e81eb431ab38e1bf1c5a4dfabec6a4e93ad8aa3
6671d81f18c167e8806a6c8ff62e2b263243acb6b4f36d2706e108b2fd7ddb34e0372
2c2ed8
oprf_key: 66234277859d85dc708f57e1f66448303ab5853c26f7f82f67bcfb2deaa
75c03

C.1.6.4. Output Values

```
registration_request: 34b611a08a5f7aff96a7a0f069c96c63db4018508da4589
b741ed538e1416234
registration_response: 1e96d9c134a103f1f961f16079f62a5468cc5b06a59d43
e20d37babea5043714846c28da13144e5908f1db4c8ba2c848ac34ae6b9a8855dcdb0
8d5ecfb000d73
registration_upload: 262321548e1fbb06525167181fb3ca34409410120c7feb41
4426b4176f519264846d97893ee0b033b318b220e7c9b3a6e63c05accae0929e086c7
7ffb9359fe9a17fffa244e400950c86a75ba7c4badbb87db74998c9f5d40967f38914
93738c86e2b0916361fb6a69f9a097e3ef2f83f8fd5f95cc79432eabf3e01f0020bac
600880f67a9cc7800800677a52179746eb1426db885097a0d0a05f93d64f93ff1498d
75c646f7f836c53ca43e93ddccd0e8225e3d608d01e8620a4a71b1a5ee9be621979af
c8c0114626fdd78fb1ee4bf7537a1743964b364b02ff7851284ae00
KE1: 703d6738685624f122e72d457bc966edbbf4a74ff8eeb530becb528762c89f7c
e0d04374ad9a276620c681abfca7bdb432e63509e5ec96ed2ec5542f6fc7db231c48c
e65e44088bb7f7503f7ad7f3a050177b727e9b490aac3d4e639be06e157
KE2: 222a32fe68ee4bc87992ab9eee9bca419aee14e3ca120020758459d85d59e639
f551feeed04ea7967c2d6c8847b6ca8bc09eced6848f0a73208e71e3f1b688e6877df
f33abacd0c5877dac42c7ebfb625a7bc11af54dc1fcc5ee9186a132e0032a49615db5
4038d2d75552e3c71188ce364e5967d6c419724b6b4d94eca4cf625c6289e300d63a7
0483156af8070f4ae27cfb2a2fc7f77405cf2431784c4c5117ceb4cb086f95fe87301
63ce6686a15a5ebd75785c973055a903cd49a90f7d98c350a8b7017c11d8f2b9c8393
241971b51e53a6a5d18f6001b4085f5a2ae62b6f35483b457e208972f1cbffbb70249
fa31064a8883002d3b8024a023b4feac41d856f10de218fc035faad7dddd3f3f01e2f
27ff589ebbe605fa8e4659652fa5286aa76afc3fc0f0d7a282848a8e49279949abb29
216a8a9105f3c419388ada5e0bba5ea621d540acd3ab0d2030cf9ba1b805571a971f9
b7b1e9cd3b422adc26c
KE3: e794a1d0ec961cf944e251270bff61ad8b262c59d7e60d465241ada6bed420bc
f1885458846463b551b9ab4ce687e0592e4f7335f46ad9dd3bb0028c0dc14225
export_key: 50235404f5212db6aa62b419e64607ab27a7070120491891ef290aa1d
d87e221a6cf8a2bb72302dd756776d1fb08da1d90bf1957d3154c76917a955fa41b95
a0
session_key: 9be2659064b28f5353414c37fff5f4f8a8f7e4c951b0637373bab2f5
75d805f02d87d34921ace763e185fa8b5ca47e98b25313195719cbc1602ef6ef61b2d
d06
```

C.1.7. OPAQUE-3DH Real Test Vector 7

C.1.7.1. Configuration

```
OPRF: 0003
Hash: SHA256
MHF: Identity
KDF: HKDF-SHA256
MAC: HMAC-SHA256
EnvelopeMode: 02
Group: P256_XMD:SHA-256_SSWU_RO_
Context: 4f50415155452d504f43
Nh: 32
Npk: 33
Nsk: 32
Nm: 32
Nx: 32
Nok: 32
```

C.1.7.2. Input Values

```
oprf_seed: 26bf796e5e24e879ebb50b60b2569a3cefa279126183be8b798dc8a346
5fab30
credential_identifier: 31323334
password: 436f7272656374486f72736542617474657279537461706c65
envelope_nonce: d761c8b636815f62dad9d7ed319efefe641bb4c3d1cc83a6a6600
bbadfe603ad
masking_nonce: 65ebd00e208f7a1c679eb9edc4b8943b0ffbe09577ecdb625726cb
333292ebc7
client_private_key: 185f462ce1eec52f588a8e392f36915849b6bfcb6bd5b904c
095be332181a28b
server_private_key: df77e28c3c1a1ef8821fb7db7020bdd866ab6f771482b58ca
91c9485d74c9011
server_public_key: 03d264f0e386387704da3d82a32883d2045326b32296ed1028
12dd3ed6d26f3b8b
server_nonce: 6819cd5cca28ea3ed382a155205669141c9bcbdca0a2c66ad0adc66
f43e4e82c
client_nonce: b9e5e27d7077eec3be075d20ac4e145572a45beeeb6066b9533421c
2cd4fee72
server_keyshare: 02e47a5f856e1ad51f3c0e9e34b620c2f7e2d4b70c59db7c1d0a
3a108322242baa
client_keyshare: 02778a3a12ae78faf76c59723a0b72ee134ac3977d297ce65d59
b41a12a385975d
server_private_keyshare: c25d83d1a98a38717e97bfde3b1dd8eb8d3346ca21d8
a6439465aea2b4b54c1e
client_private_keyshare: cc99d1ec838306ac17f68550b99cea1bc58b89723c0d
fada6c3dcb53ee14b2d8
blind_registration: e147f60270fc68fd0990c1fd9b34b9187adfc0b6e485f8fc6
ab6fea7ba043862
blind_login: 417fe2da072d4eccf379adfb08f7f8ec28aadd14a78b68466e51a251
d13299f2
```

C.1.7.3. Intermediate Values

client_public_key: 026be6ae76a078ed03bada87b595104050b65951dbd2f4160b
7fe0153494e9dc9b
auth_key: df9b4d68e993a3ea006d592636c8f855952e467e1cf95eb74c65b6ed8d9
fda9a
randomized_pwd: b884ee3cb8ba6e559fad6c5a4fc90612e753e98096d111724beb9
ef28aa7d8f2
envelope: d761c8b636815f62dad9d7ed319efefe641bb4c3d1cc83a6a6600bbadfe
603adb4a1813f859cc062290962b036202f889e59c281dda3979b474117a4cf86566e
18b577bacfbf0a2d9952ee1a86a49aa05fb7535871c2289480845e6667a389f6
handshake_secret: 293c040ae3ba21f683e0322cfd437d883da62920d613ee14eec
d40f5ba50f95d
server_mac_key: 2f8eaaf0719aa329a2189f1a53ef467e601a2e5fc4536956ecbb1
8a98af74352
client_mac_key: 9c07ba7f8d3639820e4c457abcd0c80884a5d212e6df2a9d55643
e6445bb8c4d
oprf_key: 7d6f3b70621307fe8f1546f736fea87d9de2c3a05a6e0526dab36c04907
8a314

C.1.7.4. Output Values

registration_request: 02aa62e9c8e75283c03ac512eb720cfaabf4e8b880e3b19
2a2aecf44000fdf4556
registration_response: 03e09f760f9f24cee3d66fabf1abdbb2ec87933f8c7fb9
6a4d7be1227b874afd3a03d264f0e386387704da3d82a32883d2045326b32296ed102
812dd3ed6d26f3b8b
registration_upload: 026be6ae76a078ed03bada87b595104050b65951dbd2f416
0b7fe0153494e9dc9b489718a412643a0c432d5263a7f186229856e6369c6ff90b1f6
463481fac03f8d761c8b636815f62dad9d7ed319efefe641bb4c3d1cc83a6a6600bba
dfe603adb4a1813f859cc062290962b036202f889e59c281dda3979b474117a4cf865
66e18b577bacfbf0a2d9952ee1a86a49aa05fb7535871c2289480845e6667a389f6
KE1: 036ddd62a3852518a45d35bfdc610bd8d479817af6d522f753c5fb8d773b1008
8cb9e5e27d7077eec3be075d20ac4e145572a45beeeb6066b9533421c2cd4fee72027
78a3a12ae78faf76c59723a0b72ee134ac3977d297ce65d59b41a12a385975d
KE2: 02a0230a874e88e6361474924de4e674af5231547c5ab8b786aab29d71bda9f0
ab65ebd00e208f7a1c679eb9edc4b8943b0ffbe09577ecdb625726cb333292ebc77d8
17dadce52c16c0543ab1e42c5b7bf8a007facf4084cadb74740071c02bc801c152b54
22b29073ec39a52bf8752814e7525552dd79f4196360362e68fd8ce6a96d6577e6f02
ce6203f2b848a1794027bdeefc7e6b38b29d5b19a4786303546e8bcd0225a254e2840
18922507d5f0a49b29b2b8d7e7dacfbc5b0a527ea752adcb6819cd5cca28ea3ed382a
155205669141c9bcbdca0a2c66ad0adc66f43e4e82c02e47a5f856e1ad51f3c0e9e34
b620c2f7e2d4b70c59db7c1d0a3a108322242baa47ebb98243f0c332def12af18492c
68b133ab65b6ce280f08d9ff56f09b15d64
KE3: 8390976f35482624243d818f663fe572c861b38c11e079768e6513e95f1055a7
export_key: 22e0e8c776040c794313e61ea682c7b0625a13482e1fe2a8f53fafbba
2a66893
session_key: a6fff7dfb587e64683ecb38ef21be99e87c8e2f9a7b9cca92363bef8
a2c69bed

C.1.8. OPAQUE-3DH Real Test Vector 8

C.1.8.1. Configuration

```
OPRF: 0003
Hash: SHA256
MHF: Identity
KDF: HKDF-SHA256
MAC: HMAC-SHA256
EnvelopeMode: 02
Group: P256_XMD:SHA-256_SSWU_RO_
Context: 4f50415155452d504f43
Nh: 32
Npk: 33
Nsk: 32
Nm: 32
Nx: 32
Nok: 32
```

C.1.8.2. Input Values

client_identity: 616c696365
server_identity: 626f62
oprf_seed: 14aa8d5a418b130dafb6513ad917000c83c70199f3202b928f355704e3
a25dc7
credential_identifier: 31323334
password: 436f7272656374486f72736542617474657279537461706c65
envelope_nonce: 6244563994473ca960143fe2de622464f00e607813de02e1784fa
9f7d6e3b7bc
masking_nonce: c68c15bf920702323ae20c552cdaefac7e1d665da7cc705485b75a
3514b967a3
client_private_key: 56c2f4d56bcb3cb24401b182ad275c32713d314a14cd58a64
66cfb980f9a2605
server_private_key: f1a945c34e824e9063e6f2ac4fc8a48af35e4c417ebc43be4
717550e6140dc52
server_public_key: 03512bae40ee42c0fce71f447611b39f2e91d68437f75fc1f1
71e80ae8a09d30ce
server_nonce: dc4ff822a5e5865d2f9e86117a7480a8a64a166e1cb724f59c39f93
a802f7f70
client_nonce: af0023091cb7e3e9c8581d8ca2837e78cd88fb76287d235b919bc75
7a0a70ab8
server_keyshare: 03ba9d8ad954f80f5ee6053c6c51f70a7ef7a65cc9f73c6e809d
0eac701f412acb
client_keyshare: 02ef46b931a4b8a017235597c595090dac6f059f599a00f46e08
c62fc266e28d04
server_private_keyshare: 1af9e9958dbf9198484561119bff18efc1f931f08888
9520195d6a0e47b7ce91
client_private_keyshare: f3f58965653b1dcdbf269eabb52782056318551fd0f9
48617aac77ab942f2817
blind_registration: 677e95d5335dbbd7d468a4bc0750ce4672a5b3e9098e8263f
37dea986a921e99
blind_login: e304d794b21e52c661090c25af0e432e6bdcc891568226adab56bf02
dd0ea391

C.1.8.3. Intermediate Values

```
client_public_key: 030833fa0933e79ed8dafa9cf3d537eec06987fb1c064d74f4
d45a480de9a179c9
auth_key: 635d3868cdfc288fd140f838f441cb7a12c15f4340791f2eea3d84abc14
c7e87
randomized_pwd: 40b63dd7162516849eacc4e9ba4a8a18ec56ccc8420b29b2963a5
29b3d1bf88c
envelope: 6244563994473ca960143fe2de622464f00e607813de02e1784fa9f7d6e
3b7bc7728fba39864da738a24a6d48a3941ebd0106ceea73d9cb14357fb3a11a80533
1745ac05fdedbcabd3d50f6e1064d2140918843fe71964cd0d6b380b783f5443
handshake_secret: c2c9412d7a7f789383e0dab5a3872a22b1485fe92ec5a6ccb70
c0573f0a61621
server_mac_key: 517bf63a14053d17ff76030b0646d7eb2c7cf337f68241e9a49ff
a4f312cbcb7
client_mac_key: d04bf6600a7286ffb4438a5c05ec1d0be439d19833786cf1033dc
a23e1a78b4e
oprf_key: 2ec7a8d9f98c0b936e38b56d802de8a663883588afba3f3e02b152a9e6d
12627
```

C.1.8.4. Output Values

```
registration_request: 033db40e9dfcd60acbb2aa08f5dcabd2e8bb8a0d7cd24a7
39ae669ddb7b6915eae
registration_response: 02d7007b0aee071ddaa1c5a61f52f7426a72daa071548b
597b4bd8fd0e4539330603512bae40ee42c0fce71f447611b39f2e91d68437f75fc1f
171e80ae8a09d30ce
registration_upload: 030833fa0933e79ed8dafa9cf3d537eec06987fb1c064d74
f4d45a480de9a179c9d15ebd9bcb3301043e0487e01bce0b4f7cdf142de83c9522b23
c734d863d86dc6244563994473ca960143fe2de622464f00e607813de02e1784fa9f7
d6e3b7bc7728fba39864da738a24a6d48a3941ebd0106ceea73d9cb14357fb3a11a80
5331745ac05fdedbcabd3d50f6e1064d2140918843fe71964cd0d6b380b783f5443
KE1: 035d2d3a97256fd02c323405c66e25ccf2298998fd3bc8cbbf92664f9ac50b38
16af0023091cb7e3e9c8581d8ca2837e78cd88fb76287d235b919bc757a0a70ab802e
f46b931a4b8a017235597c595090dac6f059f599a00f46e08c62fc266e28d04
KE2: 02249df93cafad8cdb0450827ddd6d35c3c289a8093e00e58e93c8a7f53767f9
78c68c15bf920702323ae20c552cdaefac7e1d665da7cc705485b75a3514b967a3b1c
efc1ba51c2cfaf32cd4e5aca3d5e2bc16ab5906680acc41efef0c7e728ad66c6afcdd
e89bc9a7b7c3cc2938aa97a28edc48761877b31c41d8c8c8904bde90d70b2d0d8cd57
58984f0f1314279ce2bb41f2be7ee04c8a7dc07241b14045509220f0f4e1008eda6ce
fadd21a4017c61e8d5d4082b88e0f189563da374c039ddb2dc4ff822a5e5865d2f9e8
6117a7480a8a64a166e1cb724f59c39f93a802f7f7003ba9d8ad954f80f5ee6053c6c
51f70a7ef7a65cc9f73c6e809d0eac701f412acbff584d94c888c14c84b82142ab60c
ab13ccfdcdfd2d7e07d34a8ea9b60cad807
KE3: 375521e87d669af1f619c5083b05ee12174a054a25c08d7088b30fd3fb3f271d
export_key: de041d0b4eb5b5a67ac994bef8836fb1402acc03fe3e3a21fdc97592e
54f52c1
session_key: a7ee3765471893e0983f373e3a49899a9212318e35adb1fde922e657
2254566b
```

C.2. Fake Test Vectors

C.2.1. OPAQUE-3DH Fake Test Vector 1

C.2.1.1. Configuration

```
OPRF: 0001
Hash: SHA512
MHF: Identity
KDF: HKDF-SHA512
MAC: HMAC-SHA512
EnvelopeMode: 01
Group: ristretto255
Context: 4f50415155452d504f43
Nh: 64
Npk: 32
Nsk: 32
Nm: 64
Nx: 64
Nok: 32
```

C.2.1.2. Input Values

```
client_identity: 616c696365
server_identity: 626f62
oprf_seed: d3cb00535339fe4063c7ba5506a990c243a2b5c77b06848a0be9a0568c
252fb0d7425382babd267deeed669e56d1d5654c036211f49b42f4489f96f37100779
f
credential_identifier: 31323334
masking_nonce: 3058799f42516228746821dc8c8530d0e8273ebde81941591d69ca
5aea773090
client_private_key: 83c9bcc31a9da0ffa4489900d3d1f85bb65c27f26e9ae4e3b
66f6e02e098c503
client_public_key: 56717b74a5e1770edb14c65f22cee0487046bd96e122ba97da
ffed06c4bf4052
server_private_key: 8d3a9355f9757e7071b3f836e3fb1461a6436e92971625b17
cd7e580dd27c009
server_public_key: 7a464761cb19c8b6e832fdfcfd18779b0edc246fe808f5de6c
e7bdb54df41b67
server_nonce: 4e2a8098173efa2968036f1762f2e5df41ab976fb1bfb91dae29950
f8526de4c
server_keyshare: 0e247410004d83d7cbe3af89c62ff03f942127aec4b0084c9eb5
88e74ce6dd06
server_private_keyshare: 326345820acc8aacf4948fce775a1fd265e4e93fd579
cec8177d6389ee379b0a
masking_key: e968bfe56ad934c3e1088115bcbf1af8b405fd0de94cdf301f9192cc
2781de00617e568b14b7235cc1189265811ea354031ea39b62e31a104f181c01d3dae
4b8
KE1: b61bfe5997b644e9654b7796203831ea9b9e86499c17db3331a40673832c9729
05603c1acb64ea417c0dabaab858a5f9da046d4a0cdbf092034c00451ccdc6e1ee835
5c91d5ed7aa5ea75b8a730ba8dc45f6b41ae9713e6aa7126211346e8754
```

C.2.1.3. Output Values

```
KE2: 0826f0581be79672ccf51276e4b4079bf05aa94530591b24acbf4106cf2fa34e
3058799f42516228746821dc8c8530d0e8273ebde81941591d69ca5aea77309078577
13efdc95f69166737cd7a80ead60e1a1f805c1da9cccbc0d29120f34be291518798c7
00793f232374e66182495b76b388d9e11f479580cc2297da02fecee88a99cea6bc411
b9467e8bfa9a4006aba7f21b74b4ce3bccd686785878b0ec9b3fc4200228014d5d073
69d42d1d1b1669ecd2ad8905734ca0a641d8f16667ca4e2a8098173efa2968036f176
2f2e5df41ab976fb1bfb91dae29950f8526de4c0e247410004d83d7cbe3af89c62ff0
3f942127aec4b0084c9eb588e74ce6dd0689ffd826511fa128dc90837369bb9ed14d0
794aa3a6e45d2ef533cf6b7e3b47d963eed736c71c8ca933af078af45f573bd3fb790
336c9b47cc40f3d7c091a552
```

C.2.2. OPAQUE-3DH Fake Test Vector 2

C.2.2.1. Configuration

```
OPRF: 0003
Hash: SHA256
MHF: Identity
KDF: HKDF-SHA256
MAC: HMAC-SHA256
EnvelopeMode: 01
Group: P256_XMD:SHA-256_SSWU_RO_
Context: 4f50415155452d504f43
Nh: 32
Npk: 33
Nsk: 32
Nm: 32
Nx: 32
Nok: 32
```

C.2.2.2. Input Values

```
client_identity: 616c696365
server_identity: 626f62
oprf_seed: 42cd4f606841ca8f403920a8ecf2d60399962f49d83f857ca86676b272
1c4366
credential_identifier: 31323334
masking_nonce: d3974af728aeafc9e5af4b4cab57d7e7dfbe0ef6b08df28fae5269
229cac2332
client_private_key: 0e6b97ef90ea8cedbada0e1295233ba417790ed8e99676903
71d527ddad59a64
client_public_key: 033043e30c3dd5fb22d0b3d167acc28878ea7c3ac49cf82b2e
b4b60a8299a67f7a
server_private_key: b08b686382820021a7d32ad3cb8ff60f15437b5cb00c53f21
f3fa17ac31d2bc0
server_public_key: 03983ac5783e6a460a526066f1398cdc648518a985cc26a66f
c7573a71ce36dbe5
server_nonce: 1a60a3e31bb007db74b7114aab2f196ef6bec942a9b4fe6c61143fa
c34d42143
server_keyshare: 03eefd21dd74c665064ebcbf63ac5ebce9a45097d47dfc08d845
52a105419b44aa
server_private_keyshare: 751e5012ba0c535e008b2389bea166a5d59a49353f12
20f5e345f0546463ccdf
masking_key: 5b8caab90accd4f239e85ec978f6a6346edc0019c5671e81034ead61
5ce096fc
KE1: 028bc054fff79a9e0f0315e31cc035384aedd9d50ea8ee36630d39876ca4e592
93d797d24fe5ad528130825016bfdc2eeaeef19914c366a615bcdbefd1f04b7208023
843b78440c0e79d828ac4c2658d1cedf7e9795f2242527a4c1a254501d2ca1a
```

C.2.2.3. Output Values

```
KE2: 0353685a152940706b1ed877b2da12f3c9f417d38fab56f3228c60f72429f602
d9d3974af728aeafc9e5af4b4cab57d7e7dfbe0ef6b08df28fae5269229cac23329a9
93151e43ac41ce18939444cea5d012b8a8316ed439d6fccf06b064f7564722f555750
61897fbb6051f37e3247d08804437259fb9b022cc12715caca4ac12ef7a8b2f101269
37619ce4725e6b821de5f44ddb71a8582883aa9b5aaefa9e3d0231a60a3e31bb007db
74b7114aab2f196ef6bec942a9b4fe6c61143fac34d4214303eefd21dd74c665064eb
cbf63ac5ebce9a45097d47dfc08d84552a105419b44aadb37380855acdd939b7eb300
708d78b17ff0f99cee4ca4777c7628fb8ff591d1
```

C.2.3. OPAQUE-3DH Fake Test Vector 3

C.2.3.1. Configuration

```
OPRF: 0001
Hash: SHA512
MHF: Identity
KDF: HKDF-SHA512
MAC: HMAC-SHA512
EnvelopeMode: 02
Group: ristretto255
Context: 4f50415155452d504f43
Nh: 64
Npk: 32
Nsk: 32
Nm: 64
Nx: 64
Nok: 32
```

C.2.3.2. Input Values

```
client_identity: 616c696365
server_identity: 626f62
oprf_seed: f25de0448fe47b0e58b656838ca98ab353a8ad48d9d2594ab586bb07b2
aa57e125b63f13483a9999240c192f735821167786be727255a4dae656c5a01059cf3
a
credential_identifier: 31323334
masking_nonce: af63872227d860610466d7e3772e2092c5f770bc56d3b4961919c4
ad92059537
client_private_key: 6c1ce31656eb5fee3ead7db7565a3618a0c38fd7b4fdcfdff
48868c519ff2a0f
client_public_key: 7894b12117db10b45476e45c8d3de597c3a410287524d7b368
961afa0a556c28
server_private_key: 3777b07d6e40562f0789806a6244d71f68a66fe4eca45ab41
38c1933d7065d02
server_public_key: ce0f4be418e606efb13cc01415b35c775e546a75e539762d63
10a268bb64bb1e
server_nonce: 244c13e9e741425fef935fbfb85c70d69b4154e77bf116bdfc92cbe
93d7598f0
server_keyshare: e21b0c1506869330ada34bfcec71862762853fc95476543a0abd
89a0a3ee556d
server_private_keyshare: eedab29b34c52d3908e3c54a79d6a561182034fff5e2
99e9fcf9317fda782106
masking_key: 463d334336a31a7b1aeeed9c60ab5a3950eb508ef1a159a931abc976
049df4afb6cc6fea0f95ae3e34c802bdcc4da208c4dd68402b708c0ebb6ae0a72d59c
b90
KE1: 986d70a47ec6b689e2deafdf5f799be248e244e976f102b5ff4c9d1164351b6e
1854ee25966eeee2c830ac3f6e212b97809eced53f0d503d4c96c7e27257f4132ce34
8e52135164b0e7c16fd09b304686c39b90cb04091a1048c399b2e9d4270
```

C.2.3.3. Output Values

```
KE2: 28ae606956cc5c4bdcb854814993ef611eeaae6a8200a6ce484de35ff40f2073
af63872227d860610466d7e3772e2092c5f770bc56d3b4961919c4ad92059537e2df9
d67a4caab9b89d824a40a41d8c1e594b6c404e548abb83e47ce48b91b7905c575892c
985d33173f2c23cb1431b7ed0696f43853fde720a012a87a818b10e77787cd9a747ad
972537f6849795409a4515665783230bfc41c657f8fac5bbe78ab842cbedc3ef2df79
90cfa9d01dcacc177b9a779b5b1a453464eea16a294552c214f34c85ce3a9961b80d9
1ec028ec0f1bdf01b047989a5ee575ec7983736244c13e9e741425fef935fbfb85c70
d69b4154e77bf116bdfc92cbe93d7598f0e21b0c1506869330ada34bfcec718627628
53fc95476543a0abd89a0a3ee556d697cee3e6d2ac62405776abe1bc3ac76c71da021
9fc91ecae4806a444a0c3a8a6311671764d4d507f99cc4b57523ff5d1a8af7628ff29
dd4dfc8d223749290e1
```

C.2.4. OPAQUE-3DH Fake Test Vector 4

C.2.4.1. Configuration

```
OPRF: 0003
Hash: SHA256
MHF: Identity
KDF: HKDF-SHA256
MAC: HMAC-SHA256
EnvelopeMode: 02
Group: P256_XMD:SHA-256_SSWU_RO_
Context: 4f50415155452d504f43
Nh: 32
Npk: 33
Nsk: 32
Nm: 32
Nx: 32
Nok: 32
```

C.2.4.2. Input Values

```
client_identity: 616c696365
server_identity: 626f62
oprf_seed: 8f200b8f641a5697233757853f5f9d5184433cfa367aa55e8491fcd37b
efaf95
credential_identifier: 31323334
masking_nonce: 5cd3c47180f435d1387c5541d2cbaca3ff3818b0e1875a9b006ac7
c78ae01b6c
client_private_key: c46a5d117c6ffdebb3b958328884c080eb922b9d8ecae5d6b
f001b9670f2851a
client_public_key: 0244fe5b332845cca6b44101ea04d4c7d52108b5b7a2afd22a
ed2c2d9e555035f4
server_private_key: 046e8097602a7b16b1ed184c65c208d14792a661a7a99c495
049b803e18da601
server_public_key: 02071a35268c772719bc31f9533f9d3665d4ed96a6780894bd
99f6910059a62807
server_nonce: 68498d0a3bcbf95d086073de4572bcc707c8fe7fc297e4e9a8600f4
eb8f7730f
server_keyshare: 02bf50c918a085c33c8fdfb3d346c4f4959401cdc1c5870567f8
947040af079d2d
server_private_keyshare: c0a9b2285356d0b6ceb9c29a7932ea8039d3170a4963
85b65eee17df9e7119cf
masking_key: c533fc7175763b1f43bc46fc6fd1145e2c24964fd3fd3c88454d0390
ca876610
KE1: 03bb16bcad5d9b7bce9f9e157f732c3d78e74cf1b1eace86a32d92400ce25b7e
e7b30e7bfbc1054696ddbc54accfea2cf8a46ab37ba489e28504f3ca7a3267d8d7036
3c36c15a1593205f26ca2d31c7a61c83a138943a5754f85d249da210bb71406
```

C.2.4.3. Output Values

```
KE2: 02f0e6fbda156524e87f9639fffbf98e75cd78b9c756c67bf1bea24054688ba8
485cd3c47180f435d1387c5541d2cbaca3ff3818b0e1875a9b006ac7c78ae01b6c95c
700af3a8ad87e16b2b20f23cf596305f21d23ba619d53d4b19d1604f9347bf944f84b
cab9373990c8b3ba517cc6418a834c946acd543079256413dd25db9ebf9f009b7da2f
4cd6b73c450d7ec779405a3198e516d11afc103e9cd782a937ba813d4627d5c76ef7b
6afe4273abe2bc76b489a0a2d2d31050f004434771de9f5068498d0a3bcbf95d08607
3de4572bcc707c8fe7fc297e4e9a8600f4eb8f7730f02bf50c918a085c33c8fdfb3d3
46c4f4959401cdc1c5870567f8947040af079d2d96e9d294c33a0dd233e902982fe82
4106b9424f8df5ced750012489bbfb6b805
```

Authors' Addresses

Hugo Krawczyk
Algorand Foundation

Email: hugokraw@gmail.com

Daniel Bourdrez

Email: dan@bytema.re

Kevin Lewi
Novi Research

Email: lewi.kevin.k@gmail.com

Christopher A. Wood
Cloudflare

Email: caw@heapingbits.net