Security Assessment - opaque-ke

WhatsApp LLC

December 10, 2021 – Version 1.3

Prepared for

Kevin Lewi

Lindsay Hegy

Prepared by

Ava Howell

Kevin Henry

Synopsis

In June 2021, WhatsApp engaged NCC Group to conduct a security assessment of the `opaque-ke` library, an open source Rust implementation of the OPAQUE password authenticated key exchange protocol. The protocol is designed to allow password-based authentication in such a way that a server does not actually learn the plaintext value of the client's password, only a blinded version of the password computed using a verifiable oblivious pseudorandom function.

Two consultants spent a total of 15 days over 2 weeks focused on a detailed review of the `opaque-ke` source code and the associated version of the OPAQUE specification. The library is open source, with code available in GitHub, and the specification is available as an IETF draft RFC. The WhatsApp team provided support throughout the engagement, and the NCC Group project team achieved good coverage of the provided source code and the associated draft of the OPAQUE specification.

Scope

The primary target is `opaque-ke`, an open source Rust library intended to be a reference implementation of OPAQUE. The review focused on the tagged release `v0.5.0` located at: github.com/novifinancial/opaque-ke/tree/v0.5.0. The reviewed library implements Draft 03 of the OPAQUE RFC: www.ietf.org/archive/id/draft-irtf-cfrg-opaque-03.html. A formal review of the OPAQUE specification was not in scope, but compliance to the specification was assessed, and some comments on the RFC itself are included where relevant.

Following the initial review, a re-test was conducted across two updated releases:.

- Release `v1.2.0`, located at github.com/novifinancial/opaque-ke/tree/v1.2.0, is a direct update to `v0.5.0` and targets the same version of the OPAQUE RFC. This release contains several patches that directly address findings in this report, and was the primary target of the re-test.
- Release `v0.6.0`, located at github.com/novifinancial/opaque-ke/tree/v0.6.0, targets a newer version of the OPAQUE RFC (Draft 05), and was still under development at the time of the initial review. Fixes for some findings identified in this report were already present in this release, and are summarized alongside the relevant findings.

Limitations

During the review, the WhatsApp team was actively working on the `v0.6.0` release of `opaque-ke`, targeting a newer version of OPAQUE. Some of the findings identified in this report had been identified previously, and have been corrected in either the RFC itself and/or in patches to the `opaque-ke` library. While these patches were reviewed in isolation where applicable, an in-depth review of `opaque-ke v0.6.0` or newer versions of the RFC **was not** in scope.

Key Findings

The assessment uncovered several issues, which were promptly addressed by the WhatsApp team:

- **Insufficient Input Validation During OPRF Group Element Deserialization**: Incoming messages containing a group element were not checked for the identity element. Deserializing an identity point could have caused all subsequent point operations to 'zero out' which may have forced the `export_key` to a known value.
- **Server Can Reflect OPRF Value And Force Non-Randomized Password**: A malicious server could have forced the client's randomized password to a non-randomized password value, potentially leading to additional unexpected exposure of the client's password such as through usage of the export key.
- **Missing Error Condition in I2OSP Implementation**: A missing length check when serializing variable-length data may have caused the length prefix to be set incorrectly, preventing correct deserialization of the resulting data at the endpoint.
- **Non-Constant-Time Verification of 3DH Transcript MAC**: An attacker that can precisely measure the timing of the OPAQUE implementation's verification of the transcript hash of the 3DH key exchange may have been able to authenticate a false 3DH transcript.

Strategic Recommendations

- The library could benefit from more comprehensive testing in general. Several tests are implemented with known test vectors and to validate basic behavior, but more robust tests that explicitly validate requirements and negative test cases would be beneficial. Fuzz testing or randomized input testing may help identify potential serialization errors, and could potentially detect missing error conditions and edge cases.
- Additional guidance to a user of the library may be beneficial, particularly in instances where the client or server may need to implement OPAQUE-specific requirements not included in the library itself. For example, a server may mitigate client enumeration during the registration phase by limiting the rate at which a client can initiate the process.
- While this report was under preparation, `opaque-ke v0.6.0` was released, which updates the library to implement Draft `05` of the OPAQUE RFC, including several security-related patches, some of which address issues identified in this report. A formal review of this recent release may reveal additional findings not present in `v0.5.0`.

Additional Content

In addition to a set of formal findings, this report includes several appendices:

- a requirements summary and review;
- an overview of client enumeration attack mitigations;
- a summary of security-related patches in `opaque-ke v0.6.0`;
- a summary of security-related patches in `opaque-ke v1.2.0`;
- comments on the OPAQUE RFC.

Table of Findings

For each finding, NCC Group uses a composite risk score that takes into account the severity of the risk, application's exposure and user population, technical difficulty of exploitation, and other factors. For an explanation of NCC Group's risk rating and finding categorization, see Appendix A on page 21.

Title	Status	ID	Risk
Insufficient Input Validation During OPRF Group Element Deserialization	Fixed	004	High
Server Can Reflect OPRF Value And Force Non-Randomized Password	Fixed	010	High
Potential Constraint Violation in Public-Private Key Pairs	Fixed	002	Low
Non-Constant-Time Verification of 3DH Transcript MAC	Fixed	006	Low
Potentially Unsafe Type Conversion of usize	Fixed	007	Low
Missing Error Condition in I2OSP Implementation	Fixed	008	Low
Outdated and Unmaintained Dependencies	Fixed	001	Informational
OPRF Blinding Scalar Can Be Chosen At Random To Be The Zero Element In GF(p)	Fixed	005	Informational

Finding	**Insufficient Input Validation During OPRF Group Element Deserialization**
Risk	**High** Impact: High, Exploitability: Medium
Identifier	NCC-E001000K-004
Status	Fixed
Category	Cryptography
Component	opaque-ke
Location	novifinancial/opaque-ke/blob/master/src/messages.rs
Impact	Deserializing an identity point could have caused all subsequent point operations to 'zero out' which may have forced the `export_key` to a known value.

Description

The VOPRF[1] specification defines the `RandomScalar()` function as returning a randomly chosen **non-zero** element in GF(p). If a zero were to be chosen, then subsequent scalar-by-point multiplications in `Blind()`, `Evaluate()` and `Unblind()` (which is essentially the entire VOPRF flow) would result in the predictable neutral element. As the specification does not consider trust boundaries, the effects of a chosen zero scalar are not articulated as part of the OPRF group element deserialization process.

Given that OPAQUE messages cross trust boundaries, it is imperative to validate received input and reject the neutral point. For example, the OPAQUE[2] specification describes the following registration message types containing serialized group elements that cross trust boundaries.

```
struct {
    SerializedElement data;
} RegistrationRequest;

data  A serialized OPRF group element.

struct {
    SerializedElement data;
    opaque server_public_key[Npk];
} RegistrationResponse;

data  A serialized OPRF group element.
```

The code in *messages.rs* contains a number of messages and associated deserialization functions that do not test for the neutral point. An example for the `RegistrationRequest` is shown below. The `from_element_slice()` function on the highlighted line does not (internally) check for the neutral point. As such, the neutral point will be deserialized and returned to the calling function.

```
impl<CS: CipherSuite> RegistrationRequest<CS> {
...
    /// Deserialization from bytes
    pub fn deserialize(input: &[u8]) -> Result<Self, ProtocolError> {
        let elem_len = <CS::Group as Group>::ElemLen::to_usize();
        let checked_slice = check_slice_size(&input, elem_len,
         ➜ "first_message_bytes")?;
        // Check that the message is actually containing an element of the
```

[1] https://datatracker.ietf.org/doc/html/draft-irtf-cfrg-voprf-06#section-2.1
[2] https://datatracker.ietf.org/doc/html/draft-irtf-cfrg-opaque-03#section-3.2

```
        // correct subgroup
        let arr = GenericArray::from_slice(checked_slice);
        let alpha = CS::Group::from_element_slice(arr)?;
        Ok(Self { alpha })
    }
}
```

In the OPAQUE flow, the received registration request is provided to the `CreateRegistrationResponse()` function.[3] This function calls the OPRF `Evaluate()` function[4] which multiplies the received group element by the server's private key (scalar). The result will always be the neutral point. This same flow exists in other messages and may ultimately reach the `export_key`.

Reproduction Steps

In *opaque-ke/src/serialization/tests.rs*, modify the `random_ristretto_point()` function implementation, as shown below. Note the highlighted line introduces a single comment which results in all returned points being the neutral point. After modification, all tests continue to pass.

```
fn random_ristretto_point() -> RistrettoPoint {
    let mut rng = OsRng;
    let mut random_bits = [0u8; 64];
    rng.fill_bytes(&mut random_bits);

    // This is because RistrettoPoint is on an obsolete sha2 version
    let mut bits = [0u8; 64];
    let mut hasher = sha2::Sha512::new();
    hasher.update(&random_bits[..]);
    //bits.copy_from_slice(&hasher.finalize());

    RistrettoPoint::from_uniform_bytes(&bits)
}
```

Recommendation

Validate that the deserialized group element in all received messages is not the neutral point.

Retest Results

A commit was added to address this issue which verifies that all group elements are not the identity element on deserialization, and that all scalars are non-zero when created: https://github.com/novifinancial/opaque-ke/commit/a69ad9473aa046c03854ce23534dbfaac24ea2c1

This effectively fixes this issue in `v1.2.0`.

Client Response

Issue was fixed. Ensured that upon deserialization, any of the messages (`RegistrationRequest`, `RegistrationResponse`, `CredentialRequest`, `CredentialResponse`) will be rejected if the identity element is detected. Also added tests for *serialization/tests.rs* to ensure that this is the case. Also ensured that scalars output by the library (both client and server) are always non-zero.

[3] https://datatracker.ietf.org/doc/html/draft-irtf-cfrg-opaque-05#section-5.1.1.2
[4] https://datatracker.ietf.org/doc/html/draft-irtf-cfrg-voprf-06#section-3.4.1.1

Finding	**Server Can Reflect OPRF Value And Force Non-Randomized Password**
Risk	**High** Impact: High, Exploitability: Medium
Identifier	NCC-E001000K-010
Status	Fixed
Category	Cryptography
Component	opaque-ke
Impact	A malicious server could have forced the client's randomized password to a non-randomized password value, potentially leading to additional unexpected exposure of the client's password such as through usage of the export key.

Description

The OPAQUE protocol uses an oblivious pseudo-random function, OPRF, to allow the client and server to cooperatively compute the output of a pseudo-random function (PRF). This OPRF, specified in an IETF draft,[5] provides key security properties for the OPAQUE protocol. Per the OPAQUE draft RFC, the OPRF provides the following critical functionality:

```
  *  Oblivious Pseudorandom Function (OPRF, [I-D.irtf-cfrg-voprf],
     version -06):

     -  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".
```

The instantiation of the OPRF in OPAQUE is through a prime-order group, Ristretto. Written multiplicatively:

- the `Blind` step takes $m_1 = password^r$,
- `Evaluate` takes $m_2 = m_1^k = password^{r*k}$,
- and `Finalize` takes $m_3 = m_2^{1/r} = password^k$.

In OPAQUE, the client performs the `Blind` step during `CreateRegistrationRequest` (for registration) and `CreateCredentialRequest` (for login), the server performs the `Evaluate` step during `CreateRegistrationResponse` and `CreateCredentialResponse`, and the client performs the `Finalize` step during the `ClientFinalize` part of OPAQUE. The input to `Blind` is given as `H'(password)`, where `H'` is a hashing function that uniformly maps password strings to OPRF group elements under the random oracle assumption. During finalization, subsequent keys are derived from the output of `Finalize` by keying HKDF with the output of a slow hash keyed with the finalized value:

```
1. y = Finalize(password, blind, response.data)
2. envelope_nonce = random(32)
3. prk = HKDF-Extract(envelope_nonce, Harden(y, params))
4. Create SecretCredentials secret_creds with creds.client_private_key
```

[5] https://datatracker.ietf.org/doc/draft-irtf-cfrg-voprf/

```
5. Create CleartextCredentials cleartext_creds with response.server_public_key
   and custom identifiers creds.client_identity and creds.server_identity if
   mode is custom_identifier
6. pseudorandom_pad =
     HKDF-Expand(prk, "Pad", len(secret_creds))
7. auth_key = HKDF-Expand(prk, "AuthKey", Nh)
8. export_key = HKDF-Expand(prk, "ExportKey", Nh)
```

As such, an attacker who knows the output `y` of Finalize can compute the `auth_key`, `export_key`, and any other derived keys.

In the current design of the specification and the implementation of `opaque-ke`, the following active attack is possible (during registration *or* login):

1. The client sends its blinded password, in the traditional manner, by mapping the password to a group element and randomizing it with the scalar `r`.
2. The server receives the blinded value. Instead of randomizing the blinded password, it *reflects* the same group element sent by the client, $H'(password)^r$.
3. The client computes $y = Finalize(password, blind, response.data)$, computing $y = Z^{1/r} = H'(password)^{r*1/r} = H'(password)$. The randomized password has effectively been de-randomized. The client then generates keys from the `prk` instantiated with the de-randomized password.
4. Later, if any authentication tags or ciphertexts based on those keys are exposed, an attacker can mount a brute-force attack against the user's password by repeatedly creating trial `prk` candidates. Note that this requires access to the nominally public `envelope_nonce`, and is limited by the speed of the `Harden` slow-hash function. If the `export_key`, `auth_key`, or any other keys derived from the prk are directly exposed, brute-force attacks are possible even without `envelope_nonce`.

In practice, the exposure of ciphertexts and tags based on `export_key` or `export_key` itself may be more likely, given that its recommended usage is broad:

```
6.4.  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.
```

This property is inherent to the design of OPAQUE itself, not the `opaque-ke` implementation. The attack described is a specific case of the attack wherein a server generates an insecure OPRF key, in this case they have effectively chosen their OPRF key to be the identity element of the group by reflecting the client's blinded value. One could also imagine a malicious server which generates a non-identity low entropy OPRF key, or later leaks their OPRF key to the world, and a similar situation would result where exposure of the `export_key` or data encrypted or authenticated by any derived keys leads to additional exposure of the user's password.

Recommendation Clients should reject the server's output of `Evaluate` if it reflects the client's `Blind` output, halting the protocol with an error if reflection is detected.

Retest Results A commit was added that updates the information stored by the client such that a (constant time) check for a reflected value can be performed, and a suitable error returned if reflection occurs: https://github.com/novifinancial/opaque-ke/commit/9c28c8597a6ab33d167f3711dab1a297b1be7d6d

This issue is considered fixed in `v1.2.0`.

Client Response

- Issue was fixed. Added `ReflectedValueError` which is thrown upon checking for the equality of the "beta" value from `RegistrationResponse` and `CredentialResponse` against the "alpha" value that is stored from `RegistrationRequest` and `CredentialRequest`.
- Added a dependency on `constant_time_eq` to ensure this equality check is done in constant time.
- Also added tests to exercise this case.

Finding **Potential Constraint Violation in Public-Private Key Pairs**

Risk **Low** Impact: Undetermined, Exploitability: Undetermined

Identifier NCC-E001000K-002

Status Fixed

Category Data Validation

Component opaque-ke

Location https://github.com/novifinancial/opaque-ke/blob/v0.5.0/src/keypair.rs#L53

Impact A public method for initializing a `KeyPair` did not enforce the expected constraint that the public key corresponds to the private key, potentially violating assumptions made by other functions in the library.

Description The `KeyPair` struct is defined in *keypair.rs* and provides a container for a public key and private key. A comment on lines 73–74 of *keypair.rs* specifies:

> At all times, we should have `&public_from_private(self.private()) == self.public()`

The referenced function can be used to construct a `KeyPair` from a private key such that this constraint holds. A generic `new()` function can also be used to construct a `KeyPair` from raw bytes:

```
/// A constructor that receives public and private key independently as
/// bytes
pub fn new(public: Key, private: Key) -> Result<Self, InternalPakeError> {
    Ok(Self {
        pk: public,
        sk: private,
        _g: PhantomData,
    })
}
```

This function is used elsewhere within the project in conjunction with `public_from_private()` to construct a valid `KeyPair`. Note that the visibility of `new()` is public, and that it does not enforce any validity constraints on the constructed `KeyPair`. Therefore, the function could potentially be used unsafely by a caller.

Recommendation If it is assumed that all `KeyPair` structs represent a valid key pair over their associated group, then additional validity checks are needed within the `new()` function. The `new()` function could be also be changed to only allow initialization from a private key, with the corresponding public key being derived. Alternatively, the function's visibility could be reduced such that no calls from outside the crate are possible. In the latter case, future uses of the function must continue to use the `new()` function safely.

Retest Results The following patch removes the `new()` constructor, preventing the unsafe initialization of a `KeyPair` in `v1.2.0`: https://github.com/novifinancial/opaque-ke/commit/27f6975136d10adc9a270283461ed2e5f385ec8a. Therefore, this finding is considered addressed in `v1.2.0`.

Client Response Issue was fixed. Deleted the `new()` constructor entirely, since it was only used in a single place (`from_private_key_slice`).

Finding **Non-Constant-Time Verification of 3DH Transcript MAC**

Risk **Low** Impact: Medium, Exploitability: Low

Identifier NCC-E001000K-006

Status Fixed

Category Data Exposure

Component opaque-ke

Location https://github.com/novifinancial/opaque-ke/blob/v0.5.0/src/key_exchange/tripledh.rs#L184
https://github.com/novifinancial/opaque-ke/blob/v0.5.0/src/key_exchange/tripledh.rs#L225

Impact An attacker that can precisely measure the timing of the OPAQUE implementation's verification of the transcript hash of the 3DH key exchange may have been able to authenticate a false 3DH transcript.

Description The `opaque-ke` implementation uses triple Diffie-Hellman (3DH) to establish a session key as a result of a successful OPAQUE protocol exchange. As part of the 3DH key agreement protocol, a symmetric transcript hash is used to authenticate the 3DH key agreement messages exchanged between client and server. When the client generates the `ke3` message, it verifies the transcript hash in *tripledh.rs* as follows:

```
184         if ke2_message.mac != server_mac.finalize().into_bytes() {
185             return Err(ProtocolError::VerificationError(
186                 PakeError::KeyExchangeMacValidationError,
187             ));
```

When the server reaches the final step, processing the client's `ke3` message, it similarly verifies the transcript hash as follows:

```
225         if ke3_message.mac != client_mac.finalize().into_bytes() {
226             return Err(ProtocolError::VerificationError(
227                 PakeError::KeyExchangeMacValidationError,
228             ));
```

`ke3_message.mac` and `ke2_message.mac` are both of type `GenericArray`. Thus, the direct equality comparison will use a generic Rust array equality operation, which runs in time dependent on the `mac` and the array being compared against. Such an implementation may leak information about the correct symmetric authentication tag through timing side channels. If an attacker can precisely measure the timing of these comparisons, they may be able to forge correct symmetric authentication tags for their own transcript.

Recommendation Change these comparisons to use a constant-time comparison operation, for example by using the `hmac.verify` primitive which is in use for envelope authentication.

Retest Results A later commit was added between `opaque-ke` versions `v0.5.0` and `v0.6.0` which changes the direct comparison to the constant-time implementation provided by `hmac.verify`. This change has been applied to `v1.2.0`: https://github.com/novifinancial/opaque-ke/commit/3c4b7ce4825031736a04542103ac490d34f9dae2. Therefore, this issue is considered fixed in `v1.2.0`.

Client Response Issue was fixed. All MAC checks now rely on the `verify()` function provided by `hmac` instead of doing equality checks.

Finding **Potentially Unsafe Type Conversion of usize**

Risk **Low** Impact: Undetermined, Exploitability: Undetermined

Identifier NCC-E001000K-007

Status Fixed

Category Data Validation

Component opaque-ke

Location https://github.com/novifinancial/opaque-ke/blob/v0.5.0/src/key_exchange/tripledh.rs#L506

Impact Conversion from platform-specific `usize` to a 2 byte slice may have caused a panic on out of bound access if `usize` is too small.

Description From the section 4.2.1 of the OPAQUE specification:[6]

> The key derivation procedures for OPAQUE-3DH makes use of the functions below, re-purposed from TLS 1.3 [RFC8446]. HKDF-Expand-Label(Secret, Label, Context, Length) = HKDF-Expand(Secret, HkdfLabel, Length) Where HkdfLabel is specified as:

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

The corresponding implementation is found in *tripledh.rs*, where the first step involves writing the 16-bit length value:

```rust
fn hkdf_expand_label_extracted<D: Hash>(
    hkdf: &Hkdf<D>,
    label: &[u8],
    context: &[u8],
    length: usize,
) -> Result<Vec<u8>, ProtocolError> {
    let mut okm = vec![0u8; length];

    let mut hkdf_label: Vec<u8> = Vec::new();
    hkdf_label.extend_from_slice(&length.to_be_bytes()[std::mem::size_of::<usize>() - 2..]);
```

The highlighted line of code sets the 16-bit length by converting the big endian byte representation of the length as a `usize`, and grabbing the last two bytes as a slice. This code implicitly assumes that the length of a `usize` is at least 16 bits, otherwise the result will be out of bounds. The overwhelming majority of Rust's target platforms are either 32-bit or 64-bit, but support for some 16-bit microcontrollers does exist. While `usize` will almost certainly be large enough on any target platform, there is currently no policy which mandates this. See https://github.com/rust-lang/rfcs/issues/1748 for additional details.

The reviewers noted that logic had been fixed from a previous commit where a 64-bit target was assumed: `hkdf_label.extend_from_slice(&length.to_be_bytes()[6..]);`

[6] https://datatracker.ietf.org/doc/html/draft-irtf-cfrg-opaque-03#section-4.2.1

While the likelihood of this logic ever triggering a panic is negligible, Rust provides several safer ways to achieve the same result.

Recommendation Because the minimum length of `usize` is technically undefined, a checked conversion to a u16 should be performed with proper error checking. Functions such as `TryFrom<usize>` or `TryInto<u16>` could be considered as an alternative.

Retest Results The following patch in `v1.2.0` addresses this finding by using an explicit conversion to a `u16` as recommended: https://github.com/novifinancial/opaque-ke/commit/27f6975136d10adc9a270283461ed2e5f385ec8a

This issue is considered fixed in `v1.2.0`.

Client Response Issue was fixed. Replaced the offending code with:

```
let length_u16: u16 = u16::try_from(length).map_err(|_|
 → PakeError::SerializationError)?;
hkdf_label.extend_from_slice(&length_u16.to_be_bytes());
```

Finding **Missing Error Condition in I2OSP Implementation**

Risk **Low** Impact: Low, Exploitability: Low

Identifier NCC-E001000K-008

Status Fixed

Category Error Reporting

Component opaque-ke

Location https://github.com/novifinancial/opaque-ke/blob/v0.5.0/src/serialization/mod.rs#L9

Impact A missing length check when serializing variable-length data may have caused the length prefix to be set incorrectly, preventing correct deserialization of the resulting data at the endpoint.

Description The values `client_identity` and `server_identity` represent either client and server public keys respectively, or server-specified custom identifiers. These values are used as part of the 3DH protocol, and are contained in the `Envelope` when the type is “custom_identifier”, as defined in section 3.1 of the OPAQUE specification:

```
struct {
 opaque server_public_key[Npk];
 opaque client_identity<0..2^16-1>;
 opaque server_identity<0..2^16-1>;
 } CleartextCredentials;
```

These values are serialized by using a 2-byte I2OSP-encoded length prefix followed by the raw bytes, as specified in RFC8017:[7]

```
1.  If x >= 256^xLen, output "integer too large" and stop.

2.  Write the integer x in its unique xLen-digit representation in base 256:

       x = x_(xLen-1) 256^(xLen-1) + x_(xLen-2) 256^(xLen-2) + ...
       + x_1 256 + x_0,

    where 0 <= x_i < 256 (note that one or more leading digits
    will be zero if x is less than 256^(xLen-1)).

3.  Let the octet X_i have the integer value x_(xLen-i) for 1 <= i
       <= xLen.  Output the octet string

             X = X_1 X_2 ... X_xLen.
```

Note that step 1 in the above function is an explicit check to ensure that the provided length can be represented in the specified number of bytes.

In `opaque-ke`, I2OSP is implemented in the following function in *serialization/mod.rs*:

```rust
// Corresponds to the I2OSP() function from RFC8017
pub(crate) fn i2osp(input: usize, length: usize) -> Vec<u8> {
    if length <= std::mem::size_of::<usize>() {
```

[7] https://datatracker.ietf.org/doc/html/rfc8017#section-4.1

```
        return (&input.to_be_bytes()[std::mem::size_of::<usize>() -
         ➜  length..]).to_vec();
    }

    let mut output = vec![0u8; length];
    output.splice(
        length - std::mem::size_of::<usize>()..length,
        input.to_be_bytes().iter().cloned(),
    );
    output
}
```

The required length check is not performed, and this function never returns an error. Therefore, if provided with a value too large to fit in the specified length, an incorrect length value will be returned.

Recommendation The `i2osp()` function should be updated to perform an explicit length check as specified in the RFC, and proper error reporting should be added to detect this condition.

Retest Results The missing check has been added in the following patch on `v1.2.0`: https://github.com/novifinancial/opaque-ke/commit/f15b37fda4a61ef97025e91f2fcff25ec4d23362

This issue is considered fixed in `v1.2.0`.

Client Response Issue was fixed. The i2osp function now does this check first:

```
    // Check if input >= 256^length
    if (sizeof_usize as u32 - input.leading_zeros() / 8) > length as u32( {
        return Err(PakeError::SerializationError);
    }
```

Also added tests to exercise this case.

Finding	**Outdated and Unmaintained Dependencies**
Risk	**Informational** Impact: Undetermined, Exploitability: Undetermined
Identifier	NCC-E001000K-001
Status	Fixed
Category	Auditing and Logging
Component	opaque-ke
Location	• https://github.com/novifinancial/opaque-ke/blob/v0.5.0/Cargo.toml • https://github.com/novifinancial/opaque-ke/blob/v0.5.0/deny.toml
Impact	Failure to update dependencies or to detect and respond to security notices may introduce future vulnerabilities or increase attack surface.
Description	

Cargo Audit

Rust's cargo-audit utility automates checking for crates for security advisories. The results of a scan on the opaque-ke repo follow:

```
>cargo audit
    Fetching advisory database from `https://github.com/RustSec/advisory-db.git`
      Loaded 307 security advisories (from .cargo\advisory-db)
    Updating crates.io index
    Scanning Cargo.lock for vulnerabilities (134 crate dependencies)
Crate:         cpuid-bool
Version:       0.1.2
Warning:       unmaintained
Title:         `cpuid-bool` has been renamed to `cpufeatures`
Date:          2021-05-06
ID:            RUSTSEC-2021-0064
URL:           https://rustsec.org/advisories/RUSTSEC-2021-0064
Dependency tree:
cpuid-bool 0.1.2

Crate:         cpuid-bool
Version:       0.2.0
Warning:       unmaintained
Title:         `cpuid-bool` has been renamed to `cpufeatures`
Date:          2021-05-06
ID:            RUSTSEC-2021-0064
URL:           https://rustsec.org/advisories/RUSTSEC-2021-0064
Dependency tree:
cpuid-bool 0.2.0

Crate:         crossbeam-epoch
Version:       0.9.1
Warning:       yanked
Dependency tree:
crossbeam-epoch 0.9.1
└── crossbeam-deque 0.8.0
    ├── rayon-core 1.9.0
    │   └── rayon 1.5.0
    │       └── criterion 0.3.4
    │           └── opaque-ke 0.5.0
```

```
    └── rayon 1.5.0

warning: 3 allowed warnings found
```

Note that these warnings are not necessarily security critical and do not represent vulnerabilities within the project.

The `opaque-ke` repo appears to utilize cargo-deny, a similar tool that includes additional dependency-related checks, and includes the warnings highlighted above. An excerpt from the configuration *deny.toml* follows:

```
# This section is considered when running `cargo deny check advisories`
# More documentation for the advisories section can be found here:
# https://embarkstudios.github.io/cargo-deny/checks/advisories/cfg.html
[advisories]
# The path where the advisory database is cloned/fetched into
db-path = "~/.cargo/advisory-db"
# The url of the advisory database to use
db-urls = ["https://github.com/rustsec/advisory-db"]
# The lint level for security vulnerabilities
vulnerability = "deny"
# The lint level for unmaintained crates
unmaintained = "warn"
# The lint level for crates that have been yanked from their source registry
yanked = "warn"
# The lint level for crates with security notices. Note that as of
# 2019-12-17 there are no security notice advisories in
# https://github.com/rustsec/advisory-db
notice = "warn"
# A list of advisory IDs to ignore. Note that ignored advisories will still
# output a note when they are encountered.
ignore = [
    #"RUSTSEC-0000-0000",
]
```

As highlighted above, crates with disclosed vulnerabilities will trigger a build error, and crates with a security notice will trigger a warning. While there are currently no published notices in the RustSec Database, this field could be switched to "deny" to ensure that any future security notices are highlighted in the future.

Outdated Dependencies

The tool cargo-outdated scans for outdated dependencies. The results of a non-recursive scan on the `v0.5.0` branch follow:

```
>cargo outdated -R
Name                Project  Compat  Latest  Kind         Platform
----                -------  ------  ------  ----         --------
anyhow              1.0.38   1.0.41  1.0.41  Development  ---
chacha20poly1305    0.7.1    ---     0.8.0   Development  ---
curve25519-dalek    3.0.2    3.1.0   3.1.0   Normal       ---
displaydoc          0.1.7    ---     0.2.1   Normal       ---
hex                 0.4.2    0.4.3   0.4.3   Development  ---
hkdf                0.10.0   ---     0.11.0  Normal       ---
hmac                0.10.1   ---     0.11.0  Normal       ---
proptest            0.10.1   ---     1.0.0   Development  ---
rand                0.8.3    0.8.4   0.8.4   Normal       ---
```

```
rustyline          6.3.0    ---     8.2.0   Development  ---
scrypt             0.5.0    ---     0.7.0   Normal       ---
serde_json         1.0.62   1.0.64  1.0.64  Development  ---
sha2               0.9.3    0.9.5   0.9.5   Development  ---
thiserror          1.0.23   1.0.25  1.0.25  Normal       ---
zeroize            1.2.0    1.3.0   1.3.0   Normal       ---
```

Note that at the time of review, the `v0.5.0` branch is several months old, so updated dependencies are expected. While no specific vulnerabilities were identified in the currently used versions, it is nevertheless best practice to keep dependencies up to date.

Recommendation

- Consider increasing the warning level of security notices to “deny”.
- Use up-to-date dependencies wherever possible, particularly when packages are security-related.

Retest Results

The following patch addressed recommendations by increasing the severity of security notices from “warn” to “deny” on the `v1.0.0` release: https://github.com/novifinancial/opaque-ke/commit/27f6975136d10adc9a270283461ed2e5f385ec8a

The following patch updated several dependencies and bumps the Minimum Supported Rust Version (MSRV) to `1.51` as part of `v.1.2.0`: https://github.com/novifinancial/opaque-ke/commit/e66140b71287e7227de5745dd33ae8321cee45bc

Similarly, the following patch on `v0.6.0` updated several dependencies: https://github.com/novifinancial/opaque-ke/commit/ed086c95284fc06c2dbba015e9aea2bca3c3a1b6

Updated crates include:

- `curve25519-dalek`
- `hkdf`
- `hmac`

These represent the core security libraries utilized by `opaque-ke`.

The general recommendation to ensure that dependencies are kept up to date remains as best practice. The commit history for both the `v1.2.0` (Draft 03) and `v0.6.0` (Draft 05) branches indicate that updates are monitored and applied as part of the release process. Therefore, this finding is considered fixed.

Client Response

Issue was fixed. Changed `notice = “warn”` to `notice = “deny”` in *deny.toml*. Ensured that all dependencies were up to date (at the time of the fix) by running cargo check.

Finding	**OPRF Blinding Scalar Can Be Chosen At Random To Be The Zero Element In GF(p)**
Risk	**Informational** Impact: High, Exploitability: Low
Identifier	NCC-E001000K-005
Status	Fixed
Category	Data Validation
Component	opaque-ke
Impact	The OPRF password-derived key may have been set to a zero-filled byte array, with negligible probability in the absence of other implementation issues. This may have permitted attackers to easily decrypt credentials protected with this key, in transit or at rest.

Description

The `opaque-ke` library includes an implementation of the OPAQUE protocol. OPAQUE relies on the Oblivious Pseudorandom Function (OPRF) protocol for two parties to compute the output of a PRF. In this protocol, the client generates a token and blinding data. The server computes the OPRF evaluation over this blinded token. The client then unblinds the server response and produces the password-derived key. Section "3.4.3.1. Blind" of Draft 06 of the "Oblivious Pseudorandom Functions (OPRFs) using Prime-Order Groups" RFC[8] explains that the blinded data must be derived as follows:

```
def Blind(input):
  blind = GG.RandomScalar()
  P = GG.HashToGroup(input)
  blindedElement = GG.SerializeElement(blind * P)

  return blind, blindedElement
```

In the above pseudo-code, function `RandomScalar()` is a member function of `GG`, a prime-order group, that chooses at random a *non-zero* element in `GF(p)`, a finite field of prime order `p`.

`opaque-ke`'s `random_scalar()` function is used to create blinding factors as follows:

```
fn random_scalar<R: RngCore + CryptoRng>(rng: &mut R) -> Self::Scalar {
        #[cfg(not(test))]
        {
            let mut scalar_bytes = [0u8; 64];
            rng.fill_bytes(&mut scalar_bytes);
            Scalar::from_bytes_mod_order_wide(&scalar_bytes)
        }

        // Tests need an exact conversion from bytes to scalar, sampling only 32
         ➜  bytes from rng
        #[cfg(test)]
        {
            let mut scalar_bytes = [0u8; 32];
            rng.fill_bytes(&mut scalar_bytes);
            Scalar::from_bytes_mod_order(scalar_bytes)
        }
    }
```

Note that there is no branch here which checks that the randomly generated value is *non-zero*.

[8] https://datatracker.ietf.org/doc/html/draft-irtf-cfrg-voprf-06#section-3.4.1

This is a deviation from the OPRF protocol specification, which may have an adverse security impact. Indeed, if the OPRF protocol is fully evaluated by all parties with a blinding value of zero, then the password-derived key will be a byte array filled with zeros. This would in turn permit attackers to decrypt client credentials wrapped in an OPAQUE envelope, without knowledge of the client password, be it in transit between the client and server, or at rest on the server.

However, in the absence of other implementation issues, the probability of generating the zero group element at random is negligible.

Recommendation Update the `random_scalar()` function to return only non-zero values in all places where non-zero scalars are specified by the OPAQUE RFC.

Retest Results It was verified that a new commit changes this function to `random_nonzero_scalar()`, which loops until a non-zero random scalar is found, only ever returning a non-zero scalar: https://github.com/novifinancial/opaque-ke/commit/a69ad9473aa046c03854ce23534dbfaac24ea2c1

This finding is considered addressed in `v1.2.0`.

Client Response Issue was fixed. Changes the `random_scalar()` function to `random_nonzero_scalar()`, which loops until a non-zero random scalar is found, only ever returning a non-zero scalar.

Appendix A: Finding Field Definitions

The following sections describe the risk rating and category assigned to issues NCC Group identified.

Risk Scale

NCC Group uses a composite risk score that takes into account the severity of the risk, application's exposure and user population, technical difficulty of exploitation, and other factors. The risk rating is NCC Group's recommended prioritization for addressing findings. Every organization has a different risk sensitivity, so to some extent these recommendations are more relative than absolute guidelines.

Overall Risk

Overall risk reflects NCC Group's estimation of the risk that a finding poses to the target system or systems. It takes into account the impact of the finding, the difficulty of exploitation, and any other relevant factors.

Critical Implies an immediate, easily accessible threat of total compromise.

High Implies an immediate threat of system compromise, or an easily accessible threat of large-scale breach.

Medium A difficult to exploit threat of large-scale breach, or easy compromise of a small portion of the application.

Low Implies a relatively minor threat to the application.

Informational No immediate threat to the application. May provide suggestions for application improvement, functional issues with the application, or conditions that could later lead to an exploitable finding.

Impact

Impact reflects the effects that successful exploitation has upon the target system or systems. It takes into account potential losses of confidentiality, integrity and availability, as well as potential reputational losses.

High Attackers can read or modify all data in a system, execute arbitrary code on the system, or escalate their privileges to superuser level.

Medium Attackers can read or modify some unauthorized data on a system, deny access to that system, or gain significant internal technical information.

Low Attackers can gain small amounts of unauthorized information or slightly degrade system performance. May have a negative public perception of security.

Exploitability

Exploitability reflects the ease with which attackers may exploit a finding. It takes into account the level of access required, availability of exploitation information, requirements relating to social engineering, race conditions, brute forcing, etc, and other impediments to exploitation.

High Attackers can unilaterally exploit the finding without special permissions or significant roadblocks.

Medium Attackers would need to leverage a third party, gain non-public information, exploit a race condition, already have privileged access, or otherwise overcome moderate hurdles in order to exploit the finding.

Low Exploitation requires implausible social engineering, a difficult race condition, guessing difficult-to-guess data, or is otherwise unlikely.

Category

NCC Group categorizes findings based on the security area to which those findings belong. This can help organizations identify gaps in secure development, deployment, patching, etc.

Access Controls	Related to authorization of users, and assessment of rights.
Auditing and Logging	Related to auditing of actions, or logging of problems.
Authentication	Related to the identification of users.
Configuration	Related to security configurations of servers, devices, or software.
Cryptography	Related to mathematical protections for data.
Data Exposure	Related to unintended exposure of sensitive information.
Data Validation	Related to improper reliance on the structure or values of data.
Denial of Service	Related to causing system failure.
Error Reporting	Related to the reporting of error conditions in a secure fashion.
Patching	Related to keeping software up to date.
Session Management	Related to the identification of authenticated users.
Timing	Related to race conditions, locking, or order of operations.

Appendix B: OPAQUE Draft 03 Requirements Review

The reviewed version of `opaque-ke` (`v0.5.0`) implements Draft 03 of the OPAQUE specification: https://www.ietf.org/archive/id/draft-irtf-cfrg-opaque-03.html. This section surveys formal requirements (e.g., SHALL, MUST, SHOULD) from the RFC, as well as other technical requirements identified in the same document, and explains how `opaque-ke` meets these requirements.

Requirement:

> Clients MUST NOT use the same key pair (client_private_key, client_public_key) for two different accounts.

The current implementation generates a fresh keypair internally as part of `ClientRegistration::finish()` in *opaque.rs* line 187: https://github.com/novifinancial/opaque-ke/blob/v0.5.0/src/opaque.rs#L187

Requirement:

> Both client and server MUST validate the other party's public key(s) used for the execution of OPAQUE.

The provided `KeyPair` struct defines a method `check_public_key()` for this purpose. This function is used to validate a public key on deserialization in the following situations:

- Deserializing a `ServerRegistration` message containing the client's public key. https://github.com/novifinancial/opaque-ke/blob/v0.5.0/src/opaque.rs#L287
- Deserializing a `RegistrationUpload` message containing the client's public key. https://github.com/novifinancial/opaque-ke/blob/v0.5.0/src/messages.rs#L118
- Deserializing a `CredentialResponse` message containing the server's public key. https://github.com/novifinancial/opaque-ke/blob/v0.5.0/src/messages.rs#L210

The OPAQUE RFC provides additional explicit guidance on validating certain types of keys:

> 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.

The `opaque-ke` implementation is defined over the Ristretto Group, so length and point validation are implicit; however, **there is no check for the point at infinity**, as noted in finding NCC-E001000K-005 on page 19.

Requirement:

> The "EnvelopeMode" value. This MUST be one of "base" or "custom_identifier".

The enum `ClientRegistrationFinishParameters` encapsulates these two options, with "base" being the default case. Rust's exhaustive matching ensures both cases are correctly handled during serialization.

Requirement:

> Upon completion of this [client registration] function, the client MUST send "record" to the server.

The actual sending of messages is out of scope of the library. The provided examples acknowledge this step; e.g. https://github.com/novifinancial/opaque-ke/blob/v0.5.0/examples/simple_login.rs#L80

Requirement:

> The type of keys MUST be suitable for the key exchange protocol. For example, if the key exchange protocol is 3DH, as described in Section 4.2.2, then the private and public keys must be Diffie-Hellman keys.

The `opaque-ke` library provides a `CipherSuite` trait to allow support of new algorithms, which contains a `Group` trait encapsulating this requirement. The only supported `Group` by default is the Ristretto Group, which satisfies the specified requirements.

Requirement:

> OPAQUE produces two outputs: a session secret and an export key. The export key may be used for additional application-specific purposes, as outlined in Section 6.4. The output "export_key" MUST NOT be used in any way before the HMAC value in the envelope is validated.

The export key is contained in an envelope returned from the server. As part of `ClientLogin::finish()` the envelope is opened, which will throw an error if the HMAC is not successfully validated. All retrieved values are taken from the "opened" envelope after validation: https://github.com/novifinancial/opaque-ke/blob/v0.5.0/src/opaque.rs#L591

Requirement:

> We note that by the results in [OPAQUE], KE2 and KE3 must authenticate credential_request and credential_response, respectively, for binding between the underlying OPRF protocol messages and the KE session.

The 3DH protocol used in `opaque-ke` includes HMACs on KE2 and KE3, which are verified during final processing of the respective messages.

- KE2: https://github.com/novifinancial/opaque-ke/blob/v0.5.0/src/key_exchange/tripledh.rs#L184
- KE3: https://github.com/novifinancial/opaque-ke/blob/v0.5.0/src/key_exchange/tripledh.rs#L225

Note that these HMAC verifications **are not constant time**, as documented in finding NCC-E001000K-006 on page 11. The OPAQUE specification mandates a constant time MAC validation for credential envelopes, but does not formally specify the same for the Authenticated Key Exchange (AKE) protocol itself. To prevent timing attacks, use of the `ct_equal()` primitive described in the OPAQUE specification should be adopted. This issue has been fixed in newer releases of the specification and library.

Requirement:

> We use the parameters Npk and Nsk to denote the size of the public and private keys used in the AKE instantiation. Npk and Nsk must adhere to the output size limitations of the HKDF Expand function from [RFC5869], which means that Npk, Nsk <= 255 * Nh. ... The parameters Npk and Nsk are set to be equal to the size of an element and scalar, respectively, in the associated prime order group.

The default (and only supported) configuration of `opaque-ke` uses the Ristretto Group and SHA-512 such that:

- Nh = 64
- Npk = 32
- Nsk = 32

which satisfy the necessary constraints.

Requirement:

> The Group mode identifies the group used in the OPAQUE-3DH AKE. This SHOULD match that of the OPRF.

The default (and only) configuration of `opaque-ke` uses the Ristretto Group for both the AKE and OPRF. The `CipherSuite` trait only contains one `Group`, which ensures this requirement is likely to be met by any additional algorithm that may be added in the future.

Requirement:

> Applications SHOULD select parameters that balance cost and complexity.

The OPAQUE specification provides concrete recommendations for 3DH groups, but not parameters for Memory Hard Functions (MHFs). In particular, specific MHFs are suggested, but no work factor is defined.

> The OPAQUE MHFs include Argon2 [I-D.irtf-cfrg-argon2], scrypt [RFC7914], and PBKDF2 [RFC2898] with fixed parameter choices.

In `opaque-ke`, these are defined in *slow_hash.rs* for scrypt.

```
#[cfg(feature = "slow-hash")]
const DEFAULT_SCRYPT_LOG_N: u8 = 15u8;
#[cfg(feature = "slow-hash")]
const DEFAULT_SCRYPT_R: u32 = 8u32;
#[cfg(feature = "slow-hash")]
const DEFAULT_SCRYPT_P: u32 = 1u32;
```

These parameters are in line with common recommendations, e.g. by OWASP,[9] thus satisfying this requirement.

Note that newer releases of the library have removed scrypt in favor of Argon2. These changes were not part of the `v0.5.0` release targeted by this review.

- Argon 2 Support: opaque-ke/commit/535b9b8ee41392c2a0c100f71f5cd04497a61817
- Remove scrypt: opaque-ke/commit/ca50d92f966e5f5dd17f1b96a79acddce7f4bc42

Requirement: Client Enumeration Protections

> Note that if the same CredentialRequest is received twice by the server, the response needs to be the same in both cases (since this would be the case for real clients).
>
> Client enumeration refers to attacks where the attacker tries to learn whether a given client identity is registered with a server. Preventing such attacks requires the server to act with unknown client identities in a way that is indistinguishable from its behavior with existing clients. ... Care needs to be taken to avoid side-channel leakage (e.g., timing) from helping differentiate these operations from a regular server response.

The specification provides a solution wherein the server uses a dummy key and a zero-vector in place of the actual credentials when a user does not exist. As noted in **??** on page ??, explicit protections against client enumeration are not implemented in `v0.5.0`, but have been added to newer releases of the library.

Other Notes

OPAQUE Draft 03 specifies the following data types for `client_identity` and `server_identity`:

```
struct {
 opaque server_public_key[Npk];
 opaque client_identity<0..2^16-1>;
 opaque server_identity<0..2^16-1>;
 } CleartextCredentials
```

[9] OWASP Password Storage Cheat Sheet: https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html

The following guidance is provided:

> 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.

There is potential ambiguity here, as a custom envelope with empty identifiers could be interpreted as intentionally supplied empty strings to be used as such, or as omitted values where the appropriate public key should be used. The reviewed version of `opaque-ke` will use empty identifiers as-is.

The most recent OPAQUE specification (Draft 05) has been updated to disallow empty identifiers:

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

Appendix C: Client Enumeration Attacks

This section summarizes a finding related to client enumeration that is better characterized as a potential attack against the reviewed version of OPAQUE, rather than against the implementation in `opaque-ke`. Although client enumeration attacks may not apply in all use cases, the content within this section may be of interest to users of the `opaque-ke` library.

Details

In many applications, the fact that a given client is registered with a server may be considered sensitive information. Therefore, a server utilizing OPAQUE must not treat a client login request for an unregistered user in a manner that is distinguishable from a registered user. Draft 03 of the OPAQUE specification suggests potential safeguards to protect against client enumeration in section 6.8:[10]

> Here we suggest a way to implement such defense, namely, a way for simulating a CredentialResponse for non-existing clients. Note that if the same CredentialRequest is received twice by the server, the response needs to be the same in both cases (since this would be the case for real clients).
>
> ...
>
> Upon receiving a CredentialRequest for a non-existing client client_identity, S computes oprf_key = f(MK; client_identity) and oprf_key' = f(MK'; client_identity) and responds with CredentialResponse carrying Z = M^oprf_key and envelope, where the latter is computed as follows. prk is set to oprf_key' and secret_creds is set to the all-zero string (of the length of a regular envelope plaintext). Care needs to be taken to avoid side- channel leakage (e.g., timing) from helping differentiate these operations from a regular server response. The above requires changes to the server-side implementation but not to the protocol itself or the client-side.

Note that more recent drafts of the specification have formally specified application-specific requirements for enumeration protection (e.g. Section 8 of Draft 6[11]):

> 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 9.8. In settings where this attack is not a concern, servers may choose to not support this functionality.

Also,

> In the case of a record that does not exist, the server SHOULD invoke the CreateCredentialResponse function where the record argument is configured so that: record.masking_key is set to a random byte string of length Nh, and record.envelope is set to the byte string consisting only of zeros, of length Ne.

In the reviewed `opaque-ke` library, the CredentialResponse message is computed by the `ServerLogin::start()` function in *opaque.rs*:

```
pub fn start<R: RngCore + CryptoRng>(
        rng: &mut R,
        password_file: ServerRegistration<CS>,
        server_s_sk: &Key,
        l1: CredentialRequest<CS>,
        params: ServerLoginStartParameters,
    ) -> Result<ServerLoginStartResult<CS>, ProtocolError> {
```

In this function the input to the PRF is taken directly from the provided `password_file`, which is stored at the time of registration, and passed to this function in order to enable the retrieval of the client's credentials. Part of this process

[10] https://datatracker.ietf.org/doc/html/draft-irtf-cfrg-opaque-03#section-6.8
[11] https://datatracker.ietf.org/doc/html/draft-irtf-cfrg-opaque#section-8

involves the evaluation of the OPRF function, which uses the `oprf_key` as provided in the `password_file`:

```
let beta = oprf::evaluate(l1.alpha, &password_file.oprf_key);
```

The function then proceeds on the envelope as stored in the `password_file`. There is no provided interface or logic to accommodate an unregistered user (where no `password_file` exists), which would require the calling library to re-implement portions of the OPAQUE protocol to compute a dummy response, or to reject requests in a way that prevents enumeration.

It is important to note that the above protection refer specifically to the client login message flows. Alternate protections on the registration process are still required, such as rate limiting, due to the need for distinguishable registration messages, but are out scope of the OPAQUE protocol itself.

Recommendation

The enumeration protections suggested (and now required, in recent drafts) in the OPAQUE specification should be implemented, and clear guidance on the correct usage should be provided in the function documentation and the provided examples.

Client Response

Properly supporting client enumeration protections requires a change in the specification. The `v1.0.0` [and the updated `v1.2.0`] release targets Draft 03 of the OPAQUE specification which did not formally specify requirements for client enumeration protections, and hence cannot have proper protections in place. Releases of `opaque-ke` targeting later versions of the specification do implement the required protections.

Re-test Results

This issue was identified in `v0.5.0` release of the library. Subsequently, the following patch added a large disclaimer within the code explaining client enumeration attacks and providing additional guidance to users of the library: https://github.com/novifinancial/opaque-ke/commit/eb50728849bca4fc93fe7fb6999c3d7da696d1d1

The current master branch, which targets a newer version of the OPAQUE specification, includes additional protections against client enumeration. While a formal review of the current master branch was not performed, it was examined specifically for changes related to client enumeration. The changes described here implement current recommendations to avoid enumeration attacks, thereby fixing the issue.

The following patch introduces initial protections against enumeration attacks, including the notion of a dummy envelope and the ability to call the `ServerLogin::start()` function with an *optional* password file: https://github.com/novifinancial/opaque-ke/pull/153. Note that the actual handling of a dummy `password_file` was not included as part of this patch initially.

Subsequently, `v0.6.0` does include handling for this case in *opaque.rs*:

```
pub fn start<R: RngCore + CryptoRng>(
        rng: &mut R,
        server_setup: &ServerSetup<CS>,
        password_file: Option<ServerRegistration<CS>>,
        l1: CredentialRequest<CS>,
        credential_identifier: &[u8],
        params: ServerLoginStartParameters,
    ) -> Result<ServerLoginStartResult<CS>, ProtocolError> {
        let record = match password_file {
            Some(x) => x,
            None => ServerRegistration::dummy(rng, server_setup),
        };
```

Where a dummy `password_file` is instantiated by the `ServerRegistration::dummy()` function in *messages.rs*:

```
// Creates a dummy instance used for faking a [CredentialResponse]
pub(crate) fn dummy<R: RngCore + CryptoRng>(
    rng: &mut R,
    server_setup: &ServerSetup<CS>,
) -> Self {
    let mut masking_key = vec![0u8; <CS::Hash as Digest>::OutputSize::to_usize()];
    rng.fill_bytes(&mut masking_key);

    Self {
        envelope: Envelope::<CS>::dummy(),
        masking_key: GenericArray::clone_from_slice(&masking_key),
        client_s_pk: server_setup.fake_keypair.public().clone(),
    }
}
```

With the corresponding dummy `Envelope` as defined in *envelope.rs*:

```
// Creates a dummy envelope object that serializes to the all-zeros byte string
pub(crate) fn dummy() -> Self {
    Self {
        mode: InnerEnvelopeMode::Zero,
        nonce: vec![0u8; NONCE_LEN],
        hmac: GenericArray::clone_from_slice(&vec![
            0u8;
            <CS::Hash as Digest>::OutputSize::to_usize()
        ]),
    }
}
```

This dummy envelope satisfies the current requirements cited above:

- 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

The function proceeds using the same codepath for both dummy and real envelopes, thus implementing the required protections against enumeration attacks.

As a result of the above changes, this issue is considered to be addressed in `v0.6.0`.

Appendix D: OPAQUE RFC Comments

While a formal review of the OPAQUE RFC was not in scope, this section highlights findings from Draft 03 and Draft 05 of the RFC that may be of interest to the `opaque-ke` developers, or editors of the RFC.

Serialization of Identities

Client and server information is stored in an `Envelope` structure, which can be a "base" configuration containing public keys, or a "custom" configuration containing a server-defined `client_identity` and `server_identity`, such as an email address, account name, etc. These values are serialized as part of the authenticated key exchange using I2OSP. While investigating finding NCC-E001000K-008 on page 14, it was noted that `opaque-ke` uses I2OSP when serializing these values when sending / receiving an `Envelope`. However, the Draft 03 of the OPAQUE specification does not appear to provide any guidance on the serialization of these values outside of their use in 3DH.

During the final step of registration, the client sends a record of the registration to the server to complete the process, which includes the `Envelope` containing `client_identity` and `server_identity`. Concrete recommendations on the serialization of this structure (e.g., using I2OSP) would remove ambiguity in this process.

Invalid Hyperlinks

Draft 05 of the RFC contains several enumerated lists with incorrect IDs such that section anchors do not work as expected. For example, the link https://datatracker.ietf.org/doc/html/draft-irtf-cfrg-opaque#section-8 is expected to target Section 8 of the document, but instead targets Step 8 of the `CreateEnvelope()` function due to the following incorrect label associated with the enumerated list:

```
<a class="selflink" id="section-8" href="#section-8">8</a>
```

This appears to be a consistent problem across all enumerated lists with hyperrefs in the document.

Constant Time MAC Verification

As part of this report, finding NCC-E001000K-006 on page 11 observed that the validation of some HMACs was not constant time. It was similarly observed that DRAFT 03 of the OPAQUE RFC did not require constant time checks on these values, despite requiring a constant time check on Envelope HMAC validation. Explicit use of constant time checks has since been added to Draft 05 and is implemented in `opaque-ke` `v0.6.0`.

Mitigating Server Reflection

This report highlighted a potential reflection attack in finding NCC-E001000K-010 on page 7 where a malicious server can force the client's randomized password to a non-randomized password value, potentially leading to additional unexpected exposure of the client's password such as through usage of the export key.

This attack applies to the OPAQUE protocol itself, not just the `opaque-ke` implementation. An explicit check and rejection of a reflected value could be added to the specification to ensure such an attack is mitigated appropriately.

The target of this review is the `v0.5.0` release of the `opaque-ke` library. Since this release, both the OPAQUE specification and the implementation have evolved. The following is a list of patches that appear to be security-related since the release of `v0.5.0`, which are part of the `v0.6.0` release targeting Draft 05 of the OPAQUE specification. These commits are not formally reviewed as part of this report, although some are explained alongside relevant findings where applicable.

- Zeroize keys on drop: opaque-ke/commit/468e0690d7f2ae91c9a55474fcf40b0457519385
- Add zeroize on drop for remaining intermediate API states and tests: opaque-ke/commit/8bc5e7dc0289b0e462c41176da084b6b2061bc63
- Ensuring mac operations are constant-time: opaque-ke/commit/940d1dcdb29e9de45a5123adab578b3446502b72
- Ensure that all public keys are being checked when deserialized: opaque-ke/commit/30e27a11e2386f9a4870d017ba2b13fb9f815898
- Enforce public vs private keys via types: opaque-ke/commit/210e0e99dfbe096fa9161c00aeb2cfae93ee76a2
- Adding identity element checks and ensuring non-zero scalar selection: opaque-ke/commit/98f1821897cd2800e5bffb2a70541056145e99cc
- Adding client enumeration mitigations: opaque-ke/commit/f0c13945d1bca40933bacfd156441235a54fdb63
- Argon 2 Support: opaque-ke/commit/535b9b8ee41392c2a0c100f71f5cd04497a61817
- Remove scrypt: opaque-ke/commit/ca50d92f966e5f5dd17f1b96a79acddce7f4bc42
- Update dependencies: opaque-ke/commit/ed086c95284fc06c2dbba015e9aea2bca3c3a1b6
- Fixing minor nits: conversion to u16 and removing keypair constructor: opaque-ke/commit/c8c57785afb2ea433a8f97d4b475fc1a064f2730

Appendix F: Updates and Patches in v1.2.0

In response to the initial findings in this report, `opaque-ke v1.2.0` was released. This release still targets Draft 03 of the OPAQUE RFC, but incorporates several security-related patches, some of which are picked from changes that already existed on the `v0.6.0` or `master` branches, targeting newer drafts of the RFC. The complete list of patches applied between `v0.5.0` and `v1.2.0` follows:

https://github.com/novifinancial/opaque-ke/commits/v1.2.0

- Zeroize keys on drop: opaque-ke/commit/ec8f87944baaf45296e747b66bce293d1255e46e
- Add zeroize on drop for remaining intermediate API states and tests: opaque-ke/commit/05427dd97d069aeefd732e04aea4b8c0c6f813ab
- **Ensuring mac operations are constant-time**: opaque-ke/commit/3c4b7ce4825031736a04542103ac490d34f9dae2
- **Fixing minor nits: conversion to u16 and removing keypair constructor**: opaque-ke/commit/27f6975136d10adc9a270283461ed2e5f385ec8a
- **Adding i2osp error checking condition**: opaque-ke/commit/f15b37fda4a61ef97025e91f2fcff25ec4d23362
- **Adding identity element checks and ensuring non-zero scalar selection**: opaque-ke/commit/a69ad9473aa046c03854ce23534dbfaac24ea2c1
- Switch CI to using stable instead of nightly toolchain: opaque-ke/commit/e5619d48cd02a848adee2f743e0368a425719ee3
- **Adding reflected value check on client side**: opaque-ke/commit/9c28c8597a6ab33d167f3711dab1a297b1be7d6d
- Publishing v0.5.1: opaque-ke/commit/0c535c0989ef61fb6b3de790efdad9b3e0e9917d
- Publishing v1.0.0: opaque-ke/commit/2e7147ed64d0d038e35f32706493b870c131f03c
- **Updating dependencies and bumping MSRV to 1.51**: opaque-ke/commit/e66140b71287e7227de5745dd33ae8321cee45bc
- Adding no_std support for v1: opaque-ke/commit/f17f90328879d2380e6ddab4af27daa0bb69ca13
- Publishing v1.1: opaque-ke/commit/65ce753dccc146f8c50dc126dfa87e5cc0a8c300
- **Adding warning about client enumeration attacks for v1**: opaque-ke/commit/eb50728849bca4fc93fe7fb6999c3d7da696d1d1
- Adding thumbv6m-none-eabi support for v1: opaque-ke/commit/f39f30727000b61930e69d191b220d836e03c626
- Publishing v1.2.0: opaque-ke/commit/6538ee30e118cfc99c1f5bff6d2c151e2eaffd2d

The emphasized commits directly address one or more findings identified in this report.