Stream: Internet Engineering Task Force (IETF)
RFC: 9578
Category: Standards Track
Published: June 2024
ISSN: 2070-1721
Authors: S. Celi, *Brave Software*; A. Davidson, *NOVA LINCS, Universidade NOVA de Lisboa*; S. Valdez, *Google LLC*; C. A. Wood, *Cloudflare*

RFC 9578
Privacy Pass Issuance Protocols

Abstract

This document specifies two variants of the two-message issuance protocol for Privacy Pass tokens: one that produces tokens that are privately verifiable using the Issuer Private Key and one that produces tokens that are publicly verifiable using the Issuer Public Key. Instances of "issuance protocol" and "issuance protocols" in the text of this document are used interchangeably to refer to the two variants of the Privacy Pass issuance protocol.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

The Privacy Pass protocol provides a privacy-preserving authorization mechanism. In essence, the protocol allows Clients to provide cryptographic tokens that prove nothing other than that they have been created by a given server in the past [ARCHITECTURE].

This document describes two issuance protocols for Privacy Pass, each of which is built on [HTTP]. It specifies two variants: one that is privately verifiable using the Issuer Private Key based on the Oblivious Pseudorandom Function (OPRF) as defined in [OPRF] and one that is publicly verifiable using the Issuer Public Key based on the blind RSA signature scheme [BLINDRSA]. Instances of "issuance protocol" and "issuance protocols" in the text of this document are used interchangeably to refer to the two variants of the Privacy Pass issuance protocol.

This document does not cover the Privacy Pass architecture, which includes (1) choices that are necessary for deployment and (2) application-specific choices for protecting Client privacy. This information is covered in [ARCHITECTURE].

2. Terminology

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.

This document uses the terms "Origin", "Client", "Issuer", and "Token" as defined in Section 2 of [ARCHITECTURE]. Moreover, the following additional terms are used throughout this document.

Issuer Public Key: The public key (from a private-public key pair) used by the Issuer for issuing and verifying tokens.

Issuer Private Key: The private key (from a private-public key pair) used by the Issuer for issuing and verifying tokens.

Unless otherwise specified, this document encodes protocol messages in TLS notation ([TLS13], Section 3). Moreover, all constants are in network byte order.

3. Protocol Overview

The issuance protocols defined in this document embody the core of Privacy Pass. Clients receive TokenChallenge inputs from the redemption protocol ([AUTHSCHEME], Section 2.1) and use the issuance protocols to produce corresponding token values ([AUTHSCHEME], Section 2.2). The issuance protocol describes how Clients and Issuers interact to compute a token using a one-round protocol consisting of a TokenRequest from the Client and a TokenResponse from the Issuer. This interaction is shown below.

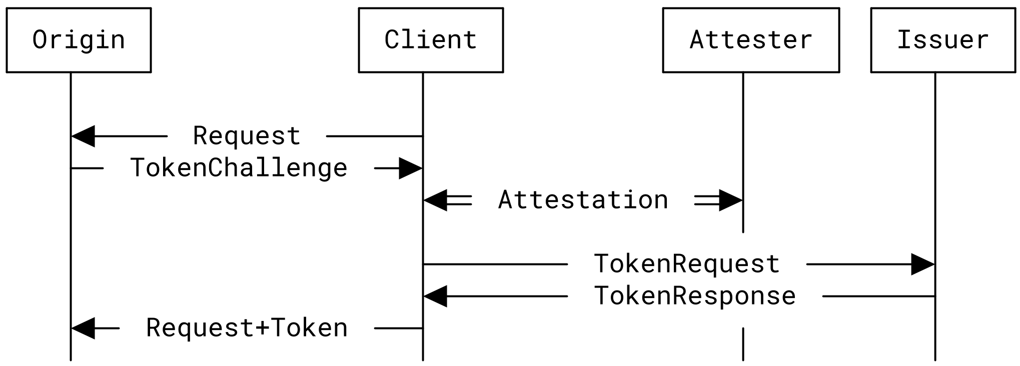

Figure 1: Issuance Overview

The TokenChallenge inputs to the issuance protocols described in this document can be interactive or non-interactive and can be per Origin or across Origins.

The issuance protocols defined in this document are compatible with any deployment model defined in Section 4 of [ARCHITECTURE]. The details of attestation are outside the scope of the issuance protocol; see Section 4 of [ARCHITECTURE] for information about how attestation can be implemented in each of the relevant deployment models.

This document describes two variants of the issuance protocol: one that is privately verifiable (Section 5) using the Issuer Private Key based on the OPRF [OPRF] and one that is publicly verifiable (Section 6) using the Issuer Public Key based on the blind RSA signature scheme [BLINDRSA].

4. Configuration

Issuers **MUST** provide two parameters for configuration:

Issuer Request URL: A token request URL for generating access tokens. For example, an Issuer Request URL might be <https://issuer.example.net/request>.

Issuer Public Key values: A list of Issuer Public Keys for the issuance protocol.

The Issuer parameters can be obtained from an Issuer via a directory object, which is a JSON object ([RFC8259], Section 4) whose values are other JSON values ([RFC8259], Section 3) for the parameters. The contents of this JSON object are defined in Table 1.

Field Name	Value
issuer-request-uri	Issuer Request URL value (as an absolute URL or as a URL relative to the directory object) as a percent-encoded URL string, represented as a JSON string ([RFC8259], Section 7)
token-keys	List of Issuer Public Key values, each represented as JSON objects ([RFC8259], Section 4)

Table 1: Issuer Directory Object Description

Each "token-keys" JSON object contains the fields and corresponding raw values defined in Table 2.

Field Name	Value
token-type	Integer value of the token type, as defined in Section 8.2, represented as a JSON number ([RFC8259], Section 6)
token-key	The base64url public key, encoded per [RFC4648], for use with the issuance protocol as determined by the token-type field, including padding, represented as a JSON string ([RFC8259], Section 7)

Table 2: Issuer "token-keys" Object Description

Each "token-keys" JSON object may also contain the optional field "not-before". The value of this field is the UNIX timestamp (number of seconds since January 1, 1970, UTC -- see Section 4.2.1 of [TIMESTAMP]) at which the key can be used. If this field is present, Clients **SHOULD NOT** use a token key before this timestamp, as doing so can lead to issuance failures. The purpose of this field is to assist in scheduled key rotations.

Beyond staging keys with the "not-before" value, Issuers **MAY** advertise multiple "token-keys" for the same token-type to facilitate key rotation. In this case, Issuers indicate their preference for which token key to use based on the order of keys in the list, with preference given to keys earlier in the list. Clients **SHOULD** use the first key in the "token-keys" list that either does not have a "not-before" value or has a "not-before" value in the past, since the first such key is the most likely to be valid in the given time window. Origins can attempt to use any key in the "token-keys" list to verify tokens, starting with the most preferred key in the list. Trial verifications like this can help deal with Client clock skew.

Altogether, the Issuer's directory could look like the following (with the "token-key" fields abbreviated):


```
{
   "issuer-request-uri": "https://issuer.example.net/request",
   "token-keys": [
     {
       "token-type": 2,
       "token-key": "MI...AB",
       "not-before": 1686913811,
     },
     {
       "token-type": 2,
       "token-key": "MI...AQ",
     }
   ]
}
```


Clients that use this directory resource before 1686913811 in UNIX time would use the second key in the "token-keys" list, whereas Clients that use this directory after 1686913811 in UNIX time would use the first key in the "token-keys" list.

A complete "token-key" value, encoded as it would be in the Issuer directory, would look like the following (line breaks are inserted to fit within the per-line character limits):

```
$ echo MIIBUjA9BgkqhkiG9w0BAQowMKANMAsGCWCGSAFlAwQCAqEaMBgGCSqGSIb3DQE \
 BCDALBglghkgBZQMEAgKiAwIBMAOCAQ8AMIIBCgKCAQEAmKHGAMyeoJt1pj3n7xTtqAPr \
 _DhZAPhJM7Pc8ENR2BzdZwPTTF7KFKms5wt-mL01at0SC-cdBuIj6WYK8Ovz0AyaBuvTv \
 W6SKCh7ZPXEqCGRsq5I0nthREtrYkGo113oMVPVp3sy4VHPgzd8KdzTLGzOrjiUOsSFWb \
 jf21iaVjXJ2VdwdS-8O-43OwkucYjGeOJwi8rWx_ZkcHtav0S67Q_SlExJel6nyRzpuuI \
 D9OQm1nxfs1Z4PhWBzt93T2ozTnda3OklF5n0pIXD6bttmTekIw_8Xx2LMis0jfJ1QL99 \
 aA-muXRFN4ZUwORrF7cAcCUD_-56_6fh9s34FmqBGwIDAQAB \
 | sed s/-/+/g | sed s/_/\\//g | openssl base64 -d \
 | openssl asn1parse -dump -inform DER
    0:d=0  hl=4 l= 338 cons: SEQUENCE
    4:d=1  hl=2 l=  61 cons: SEQUENCE
    6:d=2  hl=2 l=   9 prim: OBJECT            :rsassaPss
   17:d=2  hl=2 l=  48 cons: SEQUENCE
   19:d=3  hl=2 l=  13 cons: cont [ 0 ]
   21:d=4  hl=2 l=  11 cons: SEQUENCE
   23:d=5  hl=2 l=   9 prim: OBJECT            :sha384
   34:d=3  hl=2 l=  26 cons: cont [ 1 ]
   36:d=4  hl=2 l=  24 cons: SEQUENCE
   38:d=5  hl=2 l=   9 prim: OBJECT            :mgf1
   49:d=5  hl=2 l=  11 cons: SEQUENCE
   51:d=6  hl=2 l=   9 prim: OBJECT            :sha384
   62:d=3  hl=2 l=   3 cons: cont [ 2 ]
   64:d=4  hl=2 l=   1 prim: INTEGER           :30
   67:d=1  hl=4 l= 271 prim: BIT STRING
   ... truncated public key bytes ...
```

Issuer directory resources have the media type "application/private-token-issuer-directory" and are located at the well-known location /.well-known/private-token-issuer-directory; see Section 8.1 for the registration information for this well-known URI. This resource is located at a well-known URI because Issuers are defined by an Origin name in TokenChallenge structures; see Section 2.1 of [AUTHSCHEME].

The Issuer directory and Issuer resources **SHOULD** be available on the same Origin. If an Issuer wants to service multiple different Issuer directories, they **MUST** create unique subdomains for each directory so the TokenChallenge defined in Section 2.1 of [AUTHSCHEME] can be differentiated correctly.

Issuers **SHOULD** use HTTP cache directives to permit caching of this resource [RFC5861]. The cache lifetime depends on the Issuer's key rotation schedule. Regular rotation of token keys is recommended to minimize the risk of key compromise and any harmful effects that happen due to key compromise.

Issuers can control the cache lifetime with the Cache-Control header, as follows:

```
Cache-Control: max-age=86400
```

Consumers of the Issuer directory resource **SHOULD** follow the usual HTTP caching semantics [RFC9111] when processing this resource. Long cache lifetimes may result in the use of stale Issuer configuration information, whereas short lifetimes may result in decreased performance. When the use of an Issuer configuration results in token issuance failures, e.g., because the Issuer has invalidated its directory resource before its expiration time and issuance requests using this configuration are unsuccessful, the directory **SHOULD** be fetched and revalidated. Issuance will continue to fail until the Issuer configuration is updated.

5. Issuance Protocol for Privately Verifiable Tokens

The privately verifiable issuance protocol allows Clients to produce token values that verify using the Issuer Private Key. This protocol is based on the OPRF [OPRF].

Issuers provide an Issuer Private Key and Public Key, denoted `skI` and `pkI`, respectively, used to produce tokens as input to the protocol. See Section 5.5 for information about how these keys are generated.

Clients provide the following as input to the issuance protocol:

Issuer Request URL: A URL identifying the location to which issuance requests are sent. This can be a URL derived from the "issuer-request-uri" value in the Issuer's directory resource, or it can be another Client-configured URL. The value of this parameter depends on the Client configuration and deployment model. For example, in the "Joint Origin and Issuer" deployment model ([ARCHITECTURE], Section 4.3), the Issuer Request URL might correspond to the Client's configured Attester, and the Attester is configured to relay requests to the Issuer.

Issuer name: An identifier for the Issuer. This is typically a hostname that can be used to construct HTTP requests to the Issuer.

Issuer Public Key: `pkI`, with a key identifier `token_key_id` computed as described in Section 5.5.

Challenge value: `challenge` -- an opaque byte string. For example, this might be provided by the redemption protocol described in [AUTHSCHEME].

Given this configuration and these inputs, the two messages exchanged in this protocol are described below. This section uses notation described in [OPRF], Section 4, including SerializeElement and DeserializeElement, SerializeScalar and DeserializeScalar, and DeriveKeyPair.

The constants `Ne` and `Ns` are as defined in Section 4.4 ("OPRF(P-384, SHA-384)") of [OPRF]. For this protocol, the constant `Nk`, which is also equal to `Nh` as defined in Section 4.4 of [OPRF], is defined by Section 8.2.1.

5.1. Client-to-Issuer Request

The Client first creates a context as follows:

```
client_context = SetupVOPRFClient("P384-SHA384", pkI)
```

Here, "P384-SHA384" is the identifier corresponding to the OPRF(P-384, SHA-384) ciphersuite defined in [OPRF]. SetupVOPRFClient is defined in [OPRF], Section 3.2.

The Client then creates an issuance request message for a random 32-byte nonce with the input challenge and Issuer key identifier as described below:

```
nonce = random(32)
challenge_digest = SHA256(challenge)
token_input = concat(0x0001, // token-type field is 2 bytes long
                     nonce,
                     challenge_digest,
                     token_key_id)
blind, blinded_element = client_context.Blind(token_input)
```

The Blind function is discussed in Sections 3.3.1 and 3.3.2 of [OPRF]. If the Blind function fails, the Client aborts the protocol. The Client stores the nonce and challenge_digest values locally for use when finalizing the issuance protocol to produce a token (as described in Section 5.3).

The Client then creates a TokenRequest structured as follows:

```
struct {
  uint16_t token_type = 0x0001; /* Type VOPRF(P-384, SHA-384) */
  uint8_t truncated_token_key_id;
  uint8_t blinded_msg[Ne];
} TokenRequest;
```

The structure fields are defined as follows:

- "token_type" is a 2-octet integer, which matches the type in the challenge.
- "truncated_token_key_id" is the least significant byte of the token_key_id (Section 5.5) in network byte order (in other words, the last 8 bits of token_key_id). This value is truncated so that Issuers cannot use token_key_id as a way of uniquely identifying Clients; see referenced information from Section 7 for more details.
- "blinded_msg" is the Ne-octet blinded message defined above, computed as SerializeElement(blinded_element).

The values token_input and blinded_element are stored locally for use when finalizing the issuance protocol to produce a token (as described in Section 5.3). The Client then generates an HTTP POST request to send to the Issuer Request URL, with the TokenRequest as the content. The media type for this request is "application/private-token-request". An example request for the Issuer Request URL "https://issuer.example.net/request" is shown below.

```
POST /request HTTP/1.1
Host: issuer.example.net
Accept: application/private-token-response
Content-Type: application/private-token-request
Content-Length: <Length of TokenRequest>

<Bytes containing the TokenRequest>
```

5.2. Issuer-to-Client Response

Upon receipt of the request, the Issuer validates the following conditions:

- The TokenRequest contains a supported token_type.
- The TokenRequest.truncated_token_key_id corresponds to the truncated key ID of a public key owned by the Issuer.
- The TokenRequest.blinded_msg is of the correct size.

If any of these conditions are not met, the Issuer **MUST** return an HTTP 422 (Unprocessable Content) error to the Client.

If these conditions are met, the Issuer then tries to deserialize TokenRequest.blinded_msg using DeserializeElement ([OPRF], Section 2.1), yielding `blinded_element`. If this fails, the Issuer **MUST** return an HTTP 422 (Unprocessable Content) error to the Client. Otherwise, if the Issuer is willing to produce a token to the Client, the Issuer completes the issuance flow by computing a blinded response as follows:

```
server_context = SetupVOPRFServer("P384-SHA384", skI)
evaluate_element, proof =
  server_context.BlindEvaluate(skI, pkI, blinded_element)
```

SetupVOPRFServer is defined in [OPRF], Section 3.2, and BlindEvaluate is defined in [OPRF], Section 3.3.2. The Issuer then creates a TokenResponse structured as follows:

```
struct {
   uint8_t evaluate_msg[Ne];
   uint8_t evaluate_proof[Ns+Ns];
} TokenResponse;
```

The structure fields are defined as follows:

- "evaluate_msg" is the Ne-octet evaluated message, computed as `SerializeElement(evaluate_element)`.
- "evaluate_proof" is the (Ns+Ns)-octet serialized proof, which is a pair of Scalar values, computed as `concat(SerializeScalar(proof[0]), SerializeScalar(proof[1]))`.

The Issuer generates an HTTP response with status code 200 whose content consists of TokenResponse, with the content type set as "application/private-token-response".

```
HTTP/1.1 200 OK
Content-Type: application/private-token-response
Content-Length: <Length of TokenResponse>

<Bytes containing the TokenResponse>
```

5.3. Finalization

Upon receipt, the Client handles the response and, if successful, deserializes the content values TokenResponse.evaluate_msg and TokenResponse.evaluate_proof, yielding `evaluated_element` and `proof`. If deserialization of either value fails, the Client aborts the protocol. Otherwise, the Client processes the response as follows:

```
authenticator = client_context.Finalize(token_input, blind,
                                        evaluated_element,
                                        blinded_element,
                                        proof)
```

The Finalize function is defined in [OPRF], Section 3.3.2. If this succeeds, the Client then constructs a token as follows:

```
struct {
  uint16_t token_type = 0x0001; /* Type VOPRF(P-384, SHA-384) */
  uint8_t nonce[32];
  uint8_t challenge_digest[32];
  uint8_t token_key_id[32];
  uint8_t authenticator[Nk];
} Token;
```

The Token.nonce value is the value that was created according to Section 5.4. If the Finalize function fails, the Client aborts the protocol.

5.4. Token Verification

Verifying a token requires creating a Verifiable Oblivious Pseudorandom Function (VOPRF) context using the Issuer Private Key and Public Key, evaluating the token contents, and comparing the result against the token authenticator value:

```
server_context = SetupVOPRFServer("P384-SHA384", skI)
token_authenticator_input =
  concat(Token.token_type,
         Token.nonce,
         Token.challenge_digest,
         Token.token_key_id)
token_authenticator =
  server_context.Evaluate(token_authenticator_input)
valid = (token_authenticator == Token.authenticator)
```

5.5. Issuer Configuration

Issuers are configured with Issuer Private Keys and Public Keys, each denoted `skI` and `pkI`, respectively, used to produce tokens. These keys **MUST NOT** be reused in other protocols. A **RECOMMENDED** method for generating keys is as follows:

```
seed = random(Ns)
(skI, pkI) = DeriveKeyPair(seed, "PrivacyPass")
```

The DeriveKeyPair function is defined in [OPRF], Section 3.2.1. The key identifier for a public key `pkI`, denoted `token_key_id`, is computed as follows:

```
token_key_id = SHA256(SerializeElement(pkI))
```

Since Clients truncate `token_key_id` in each `TokenRequest`, Issuers **SHOULD** ensure that the truncated forms of new key IDs do not collide with other truncated key IDs in rotation. Collisions can cause the Issuer to use the wrong Issuer Private Key for issuance, which will in turn cause the resulting tokens to be invalid. There is no known security consequence of using the wrong Issuer Private Key. A possible exception to this constraint would be a colliding key that is still in use but is in the process of being rotated out, in which case the collision cannot reasonably be avoided; however, this situation is expected to be transient.

6. Issuance Protocol for Publicly Verifiable Tokens

This section describes a variant of the issuance protocol discussed in Section 5 for producing publicly verifiable tokens using the protocol defined in [BLINDRSA]. In particular, this variant of the issuance protocol works for the RSABSSA-SHA384-PSS-Deterministic and RSABSSA-SHA384-PSSZERO-Deterministic blind RSA protocol variants described in Section 5 of [BLINDRSA].

The publicly verifiable issuance protocol differs from the protocol defined in Section 5 in that the output tokens are publicly verifiable by anyone with the Issuer Public Key. This means any Origin can select a given Issuer to produce tokens, as long as the Origin has the Issuer Public Key, without explicit coordination or permission from the Issuer. This is because the Issuer does not learn the Origin that requested the token during the issuance protocol.

Beyond this difference, the publicly verifiable issuance protocol variant is nearly identical to the privately verifiable issuance protocol variant. In particular, Issuers provide an Issuer Private Key and Public Key, denoted skI and pkI, respectively, used to produce tokens as input to the protocol. See Section 6.5 for information about how these keys are generated.

Clients provide the following as input to the issuance protocol:

Issuer Request URL: A URL identifying the location to which issuance requests are sent. This can be a URL derived from the "issuer-request-uri" value in the Issuer's directory resource, or it can be another Client-configured URL. The value of this parameter depends on the Client configuration and deployment model. For example, in the "Split Origin, Attester, Issuer" deployment model ([ARCHITECTURE], Section 4.4), the Issuer Request URL might correspond to the Client's configured Attester, and the Attester is configured to relay requests to the Issuer.

Issuer name: An identifier for the Issuer. This is typically a hostname that can be used to construct HTTP requests to the Issuer.

Issuer Public Key: `pkI`, with a key identifier `token_key_id` computed as described in Section 6.5.

Challenge value: `challenge` -- an opaque byte string. For example, this might be provided by the redemption protocol described in [AUTHSCHEME].

Given this configuration and these inputs, the two messages exchanged in this protocol are described below. For this protocol, the constant `Nk` is defined by Section 8.2.2.

6.1. Client-to-Issuer Request

The Client first creates an issuance request message for a random 32-byte `nonce` using the input challenge and Issuer key identifier as follows:

```
nonce = random(32)
challenge_digest = SHA256(challenge)
token_input = concat(0x0002, // token-type field is 2 bytes long
                     nonce,
                     challenge_digest,
                     token_key_id)
blinded_msg, blind_inv =
  Blind(pkI, PrepareIdentity(token_input))
```

The PrepareIdentity and Blind functions are defined in Sections 4.1 and 4.2 of [BLINDRSA], respectively. The Client stores the nonce and challenge_digest values locally for use when finalizing the issuance protocol to produce a token (as described in Section 6.3).

The Client then creates a TokenRequest structured as follows:

```
struct {
  uint16_t token_type = 0x0002; /* Type Blind RSA (2048-bit) */
  uint8_t truncated_token_key_id;
  uint8_t blinded_msg[Nk];
} TokenRequest;
```

The structure fields are defined as follows:

- "token_type" is a 2-octet integer, which matches the type in the challenge.
- "truncated_token_key_id" is the least significant byte of the `token_key_id` (Section 6.5) in network byte order (in other words, the last 8 bits of `token_key_id`). This value is truncated so that Issuers cannot use `token_key_id` as a way of uniquely identifying Clients; see referenced information from Section 7 for more details.
- "blinded_msg" is the Nk-octet request defined above.

The Client then generates an HTTP POST request to send to the Issuer Request URL, with the TokenRequest as the content. The media type for this request is "application/private-token-request". An example request for the Issuer Request URL "https://issuer.example.net/request" is shown below.

```
POST /request HTTP/1.1
Host: issuer.example.net
Accept: application/private-token-response
Content-Type: application/private-token-request
Content-Length: <Length of TokenRequest>

<Bytes containing the TokenRequest>
```

6.2. Issuer-to-Client Response

Upon receipt of the request, the Issuer validates the following conditions:

- The TokenRequest contains a supported token_type.
- The TokenRequest.truncated_token_key_id corresponds to the truncated key ID of an Issuer Public Key.
- The TokenRequest.blinded_msg is of the correct size.

If any of these conditions are not met, the Issuer **MUST** return an HTTP 422 (Unprocessable Content) error to the Client. Otherwise, if the Issuer is willing to produce a token to the Client, the Issuer completes the issuance flow by computing a blinded response as follows:

```
blind_sig = BlindSign(skI, TokenRequest.blinded_msg)
```

The BlindSign function is defined in Section 4.3 of [BLINDRSA]. The result is encoded and transmitted to the Client in the following TokenResponse structure:

```
struct {
  uint8_t blind_sig[Nk];
} TokenResponse;
```

The Issuer generates an HTTP response with status code 200 whose content consists of TokenResponse, with the content type set as "application/private-token-response".

```
HTTP/1.1 200 OK
Content-Type: application/private-token-response
Content-Length: <Length of TokenResponse>

<Bytes containing the TokenResponse>
```

6.3. Finalization

Upon receipt, the Client handles the response and, if successful, processes the content as follows:

```
authenticator =
  Finalize(pkI, PrepareIdentity(token_input), blind_sig, blind_inv)
```

The Finalize function is defined in Section 4.4 of [BLINDRSA]. If this succeeds, the Client then constructs a token as described in [AUTHSCHEME] as follows:

```
struct {
  uint16_t token_type = 0x0002; /* Type Blind RSA (2048-bit) */
  uint8_t nonce[32];
  uint8_t challenge_digest[32];
  uint8_t token_key_id[32];
  uint8_t authenticator[Nk];
} Token;
```

The Token.nonce value is the value that was sampled according to Section 6.1. If the Finalize function fails, the Client aborts the protocol.

6.4. Token Verification

Verifying a token requires checking that Token.authenticator is a valid signature over the remainder of the token input using the Issuer Public Key. The function `RSASSA-PSS-VERIFY` is defined in Section 8.1.2 of [RFC8017], using SHA-384 as the hash function, MGF1 with SHA-384 as the Probabilistic Signature Scheme (PSS) mask generation function (MGF), and a 48-byte salt length (sLen).

```
token_authenticator_input =
  concat(Token.token_type,
         Token.nonce,
         Token.challenge_digest,
         Token.token_key_id)
valid = RSASSA-PSS-VERIFY(pkI,
                          token_authenticator_input,
                          Token.authenticator)
```

6.5. Issuer Configuration

Issuers are configured with Issuer Private Keys and Public Keys, each denoted skI and pkI, respectively, used to produce tokens. Each key **SHALL** be generated securely -- for example, as specified in FIPS 186-5 [DSS]. These keys **MUST NOT** be reused in other protocols.

The key identifier for an Issuer Private Key and Public Key (skI, pkI), denoted `token_key_id`, is computed as SHA256(encoded_key), where encoded_key is a DER-encoded SubjectPublicKeyInfo (SPKI) object [RFC5280] carrying pkI as a DER-encoded RSAPublicKey value [RFC5756] in the subjectPublicKey field. Additionally, (1) the SPKI object **MUST** use the id-RSASSA-PSS object identifier in the algorithm field within the SPKI object and (2) the parameters field **MUST** contain an RSASSA-PSS-params value and **MUST** include the hashAlgorithm, maskGenAlgorithm, and saltLength values. The saltLength **MUST** match the output size of the hash function associated with the public key and token type.

An example sequence of the SPKI object (in ASN.1 format, with the actual public key bytes truncated) for a 2048-bit key is shown below:

```
$ cat spki.bin | xxd -r -p | openssl asn1parse -dump -inform DER
    0:d=0  hl=4 l= 338 cons: SEQUENCE
    4:d=1  hl=2 l=  61 cons: SEQUENCE
    6:d=2  hl=2 l=   9 prim: OBJECT            :rsassaPss
   17:d=2  hl=2 l=  48 cons: SEQUENCE
   19:d=3  hl=2 l=  13 cons: cont [ 0 ]
   21:d=4  hl=2 l=  11 cons: SEQUENCE
   23:d=5  hl=2 l=   9 prim: OBJECT            :sha384
   34:d=3  hl=2 l=  26 cons: cont [ 1 ]
   36:d=4  hl=2 l=  24 cons: SEQUENCE
   38:d=5  hl=2 l=   9 prim: OBJECT            :mgf1
   49:d=5  hl=2 l=  11 cons: SEQUENCE
   51:d=6  hl=2 l=   9 prim: OBJECT            :sha384
   62:d=3  hl=2 l=   3 cons: cont [ 2 ]
   64:d=4  hl=2 l=   1 prim: INTEGER           :30
   67:d=1  hl=4 l= 271 prim: BIT STRING
   ... truncated public key bytes ...
```

Since Clients truncate `token_key_id` in each `TokenRequest`, Issuers **SHOULD** ensure that the truncated forms of new key IDs do not collide with other truncated key IDs in rotation. Collisions can cause the Issuer to use the wrong Issuer Private Key for issuance, which will in turn cause the resulting tokens to be invalid. There is no known security consequence of using the wrong

Issuer Private Key. A possible exception to this constraint would be a colliding key that is still in use but is in the process of being rotated out, in which case the collision cannot reasonably be avoided; however, this situation is expected to be transient.

7. Security Considerations

This document outlines how to instantiate the issuance protocol based on the VOPRF defined in [OPRF] and the blind RSA protocol defined in [BLINDRSA]. All security considerations described in the VOPRF and blind RSA documents also apply in the Privacy Pass use case. Considerations related to broader privacy and security concerns in a multi-Client and multi-Issuer setting are covered in the architecture document [ARCHITECTURE]. In particular, Sections 4 and 5 of [ARCHITECTURE] discuss relevant privacy considerations influenced by the Privacy Pass deployment models, and Section 6 of [ARCHITECTURE] discusses privacy considerations that apply regardless of deployment model. Notable considerations include those pertaining to Issuer Public Key rotation and consistency -- where consistency is as described in [CONSISTENCY] -- and Issuer selection.

8. IANA Considerations

8.1. Well-Known "private-token-issuer-directory" URI

IANA has updated the "Well-Known URIs" registry [WellKnownURIs] with the following values.

URI Suffix	Change Controller	Reference	Status	Related Information
private-token-issuer-directory	IETF	RFC 9578	permanent	None

Table 3: "private-token-issuer-directory" Well-Known URI

8.2. Privacy Pass Token Types

IANA has updated the "Privacy Pass Token Types" registry [PrivPassTokenTypes] with the entries below.

8.2.1. Token Type VOPRF(P-384, SHA-384)

Value: 0x0001
Name: VOPRF(P-384, SHA-384)
Token Structure: As defined in Section 2.2 of [AUTHSCHEME].
Token Key Encoding: Serialized using SerializeElement (Section 2.1 of [OPRF]).
TokenChallenge Structure: As defined in Section 2.1 of [AUTHSCHEME].
Publicly Verifiable: N
Public Metadata: N
Private Metadata: N

Nk: 48
Nid: 32
Change controller: IETF
Reference: RFC 9578, Section 5
Notes: None

8.2.2. Token Type Blind RSA (2048-bit)

Value: 0x0002
Name: Blind RSA (2048-bit)
Token Structure: As defined in Section 2.2 of [AUTHSCHEME].
Token Key Encoding: Serialized as a DER-encoded SubjectPublicKeyInfo (SPKI) object using the RSASSA-PSS OID [RFC5756].
TokenChallenge Structure: As defined in Section 2.1 of [AUTHSCHEME].
Publicly Verifiable: Y
Public Metadata: N
Private Metadata: N
Nk: 256
Nid: 32
Change controller: IETF
Reference: RFC 9578, Section 6
Notes: The RSABSSA-SHA384-PSS-Deterministic and RSABSSA-SHA384-PSSZERO-Deterministic variants are supported.

8.3. Media Types

IANA has added the following entries to the "Media Types" registry [MediaTypes]:

- "application/private-token-issuer-directory"
- "application/private-token-request"
- "application/private-token-response"

The templates for these entries are listed below. The reference is this RFC.

8.3.1. "application/private-token-issuer-directory" Media Type

Type name: application

Subtype name: private-token-issuer-directory

Required parameters: N/A

Optional parameters: N/A

Encoding considerations: binary

Security considerations: See Section 7 of RFC 9578.

Interoperability considerations: N/A

Published specification: RFC 9578

Applications that use this media type: Services that implement the Privacy Pass Issuer role, and Client applications that interact with the Issuer for the purposes of issuing or redeeming tokens.

Fragment identifier considerations: N/A

Additional information:

Deprecated alias names for this type: N/A

Magic number(s): N/A

File extension(s): N/A

Macintosh file type code(s): N/A

Person & email address to contact for further information: See the Authors' Addresses section of RFC 9578.

Intended usage: COMMON

Restrictions on usage: N/A

Author: See the Authors' Addresses section of RFC 9578.

Change controller: IETF

8.3.2. "application/private-token-request" Media Type

Type name: application

Subtype name: private-token-request

Required parameters: N/A

Optional parameters: N/A

Encoding considerations: binary

Security considerations: See Section 7 of RFC 9578.

Interoperability considerations: N/A

Published specification: RFC 9578

Applications that use this media type: Applications that want to issue or facilitate issuance of Privacy Pass tokens, including Privacy Pass Issuer applications themselves.

Fragment identifier considerations: N/A

Additional information:

Deprecated alias names for this type: N/A

Magic number(s): N/A

File extension(s): N/A

Macintosh file type code(s): N/A

Person & email address to contact for further information: See the Authors' Addresses section of RFC 9578.

Intended usage: COMMON

Restrictions on usage: N/A

Author: See the Authors' Addresses section of RFC 9578.

Change controller: IETF

8.3.3. "application/private-token-response" Media Type

Type name: application

Subtype name: private-token-response

Required parameters: N/A

Optional parameters: N/A

Encoding considerations: binary

Security considerations: See Section 7 of RFC 9578.

Interoperability considerations: N/A

Published specification: RFC 9578

Applications that use this media type: Applications that want to issue or facilitate issuance of Privacy Pass tokens, including Privacy Pass Issuer applications themselves.

Fragment identifier considerations: N/A

Additional information:

Deprecated alias names for this type: N/A

Magic number(s): N/A

File extension(s): N/A

Macintosh file type code(s): N/A

Person & email address to contact for further information: See the Authors' Addresses section of RFC 9578.

Intended usage: COMMON

Restrictions on usage: N/A

Author: See the Authors' Addresses section of RFC 9578.

Change controller: IETF

9. References

9.1. Normative References

[ARCHITECTURE] Davidson, A., Iyengar, J., and C. A. Wood, "The Privacy Pass Architecture", RFC 9576, DOI 10.17487/RFC9576, June 2024, <https://www.rfc-editor.org/info/rfc9576>.

[AUTHSCHEME] Pauly, T., Valdez, S., and C. A. Wood, "The Privacy Pass HTTP Authentication Scheme", RFC 9577, DOI 10.17487/RFC9577, June 2024, <https://www.rfc-editor.org/info/rfc9577>.

[BLINDRSA] Denis, F., Jacobs, F., and C. A. Wood, "RSA Blind Signatures", RFC 9474, DOI 10.17487/RFC9474, October 2023, <https://www.rfc-editor.org/info/rfc9474>.

[HTTP] Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "HTTP Semantics", STD 97, RFC 9110, DOI 10.17487/RFC9110, June 2022, <https://www.rfc-editor.org/info/rfc9110>.

[MediaTypes] IANA, "Media Types", <https://www.iana.org/assignments/media-types/>.

[OPRF] Davidson, A., Faz-Hernandez, A., Sullivan, N., and C. A. Wood, "Oblivious Pseudorandom Functions (OPRFs) Using Prime-Order Groups", RFC 9497, DOI 10.17487/RFC9497, December 2023, <https://www.rfc-editor.org/info/rfc9497>.

[PrivPassTokenTypes] IANA, "Privacy Pass Token Types", <https://www.iana.org/assignments/privacy-pass/>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, DOI 10.17487/RFC4648, October 2006, <https://www.rfc-editor.org/info/rfc4648>.

[RFC5756] Turner, S., Brown, D., Yiu, K., Housley, R., and T. Polk, "Updates for RSAES-OAEP and RSASSA-PSS Algorithm Parameters", RFC 5756, DOI 10.17487/RFC5756, January 2010, <https://www.rfc-editor.org/info/rfc5756>.

[RFC5861] Nottingham, M., "HTTP Cache-Control Extensions for Stale Content", RFC 5861, DOI 10.17487/RFC5861, May 2010, <https://www.rfc-editor.org/info/rfc5861>.

[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/info/rfc8017>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[RFC8259] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", STD 90, RFC 8259, DOI 10.17487/RFC8259, December 2017, <https://www.rfc-editor.org/info/rfc8259>.

[RFC9111] Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "HTTP Caching", STD 98, RFC 9111, DOI 10.17487/RFC9111, June 2022, <https://www.rfc-editor.org/info/rfc9111>.

[TIMESTAMP] Mizrahi, T., Fabini, J., and A. Morton, "Guidelines for Defining Packet Timestamps", RFC 8877, DOI 10.17487/RFC8877, September 2020, <https://www.rfc-editor.org/info/rfc8877>.

[TLS13] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <https://www.rfc-editor.org/info/rfc8446>.

[WellKnownURIs] IANA, "Well-Known URIs", <https://www.iana.org/assignments/well-known-uris/>.

9.2. Informative References

[CONSISTENCY] Davidson, A., Finkel, M., Thomson, M., and C. A. Wood, "Key Consistency and Discovery", Work in Progress, Internet-Draft, draft-ietf-privacypass-key-consistency-01, 10 July 2023, <https://datatracker.ietf.org/doc/html/draft-ietf-privacypass-key-consistency-01>.

[DSS] National Institute of Standards and Technology, "Digital Signature Standard (DSS)", NIST FIPS Publication 186-5, DOI 10.6028/NIST.FIPS.186-5, February 2023, <https://doi.org/10.6028/nist.fips.186-5>.

[RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, DOI 10.17487/RFC5280, May 2008, <https://www.rfc-editor.org/info/rfc5280>.

Appendix A. Test Vectors

This section includes test vectors for the two basic issuance protocols specified in this document. Appendix A.1 contains test vectors for token issuance protocol 1 (0x0001), and Appendix A.2 contains test vectors for token issuance protocol 2 (0x0002).

A.1. Issuance Protocol 1 - VOPRF(P-384, SHA-384)

The test vectors below list the following values:

skI: The Issuer Private Key, serialized using SerializeScalar ([OPRF], Section 2.1) and represented as a hexadecimal string.

pkI: The Issuer Public Key, serialized according to the encoding in Section 8.2.1.

token_challenge: A randomly generated TokenChallenge structure, represented as a hexadecimal string.

nonce: The 32-byte Client nonce generated according to Section 5.1, represented as a hexadecimal string.

blind: The blind used when computing the OPRF blinded message, serialized using SerializeScalar ([OPRF], Section 2.1) and represented as a hexadecimal string.

token_request: The TokenRequest message constructed according to Section 5.1, represented as a hexadecimal string.

token_response: The TokenResponse message constructed according to Section 5.2, represented as a hexadecimal string.

token: The output token from the protocol, represented as a hexadecimal string.

```
// Test vector 1
skI: 39b0d04d3732459288fc5edb89bb02c2aa42e06709f201d6c518871d5181
14910bee3c919bed1bbffe3fc1b87d53240a
pkI: 02d45bf522425cdd2227d3f27d245d9d563008829252172d34e48469290c
21da1a46d42ca38f7beabdf05c074aee1455bf
token_challenge: 0001000e6973737565722e6578616d706c65205de58a52fc
daef25ca3f65448d04e040fb1924e8264acfccfc6c5ad451d582b3000e6f72696
7696e2e6578616d706c65
nonce:
6aa422c41b59d3e44a136dd439df2454e3587ee5f3697798cdc05fafe73073b8
blind: 8e7fd80970b8a00b0931b801a2e22d9903d83bd5597c6a4dc1496ed2b1
7ef820445ef3bd223f3ab2c4f54c5d1c956909
token_request: 0001f4030ab3e23181d1e213f24315f5775983c678ce22eff9
427610832ab3900f2cd12d6829a07ec8a6813cf0b5b886f4cc4979
token_response: 036bb3c5c397d88c3527cf9f08f1fe63687b867e85c930c49
ee2c222408d4903722a19ff272ac97e3725b947c972784ebfe86eb9ea54336e43
34ea9660212c0c85fbadfbf491a1ce2446fc3379337fccd45c1059b2bc760110e
e1ec227d8e01c9f482c00c47ffa0dbe2fb58c32dde2b1dbe69fff920a528e68dd
```

```
9b3c2483848e57c30542b8984fa6bfecd6d71d54d53eda
token: 00016aa422c41b59d3e44a136dd439df2454e3587ee5f3697798cdc05f
afe73073b8501370b494089dc462802af545e63809581ee6ef57890a12105c283
68169514bf260d0792bf7f46c9866a6d37c3032d8714415f87f5f6903d7fb071e
253be2f4e0a835d76528b8444f73789ee7dc90715b01c17902fd87375c00a7a9d
3d92540437f470773be20f71e721da3af40edeb

// Test vector 2
skI: 39efed331527cc4ddff9722ab5cd35aeafe7c27520b0cfa2eedbdc298dc3
b12bc8298afcc46558af1e2eeacc5307d865
pkI: 038017e005904c6146b37109d6c2a72b95a183aaa9ed951b8d8fb1ed9033
f68033284d175e7df89849475cd67a86bfbf4e
token_challenge: 0001000e6973737565722e6578616d706c6500000e6f7269
67696e2e6578616d706c65
nonce:
7617bc802cfdb5d74722ef7418bdbb4f2c88403820e55fe7ec07d3190c29d665
blind: 6492ee50072fa18d035d69c4246362dffe2621afb95a10c033bb0109e0
f705b0437c425553272e0aa5266ec379e7015e
token_request: 000133033a5fe04a39da1bbfb68ccdeecd1917474dd525462e
5a90a6ba53b42aaa1486fe443a2e1c7f3fd5ff028a1c7cf1aeac5d
token_response: 023bf8cd624880d669c5cc6c88b056355c6e8e1bcbf3746cf
b9ab9248a4c056f23a4876ef998a8b6b281d50f852c6fa868fc4fa135c79ccb5f
bdf8bf3c926e10c7c12f934a887d86da4a4e5be70f5a169aa75720887bb690536
92a8f11f9cda7a72f281e4e3568e848225367946c70db09e718e3cba16193987b
c10bede3ef54c4d036c17cd4015bb113be60d7aa927e0d
token: 00017617bc802cfdb5d74722ef7418bdbb4f2c88403820e55fe7ec07d3
190c29d665c994f7d5cdc2fb970b13d4e8eb6e6d8f9dcdaa65851fb091025dfe1
34bd5a62a116477bc9e1a205cca95d0c92335ca7a3e71063b2ac020bdd231c660
97f12333ef438d00801bca5ace0fab8eb483dc04cd62578b95b5652921cd2698c
45ea74f6c8827b4e19f01140fa5bd039866f562

// Test vector 3
skI: 2b7709595b62b784f14946ae828f65e6caeba6eefe732c86e9ae50e818c0
55b3d7ca3a5f2beecaa859a62ff7199d35cc
pkI: 03a0de1bf3fd0a73384283b648884ba9fa5dee190f9d7ad4292c2fd49d8b
4d64db674059df67f5bd7e626475c78934ae8d
token_challenge: 0001000e6973737565722e6578616d706c65000017666f6f
2e6578616d706c652c6261722e6578616d706c65
nonce:
87499b5930918d2d83ecebf92d25ca0722aa11b80dbbfd950537c28aa7d3a9df
blind: 1f659584626ba15f44f3d887b2e5fe4c27315b185dfbfaea4253ebba30
610c4d9b73c78714c142360e85a00942c0fcff
token_request: 0001c8024610a9f3aac21090f3079d6809437a2b94b4403c7e
645f849bc6c505dade154c258c8ecd4d2bdcf574daca65db671908
token_response: 03c2ab925d03e7793b4a4df6eb505210139f620359e142449
1b8143c06a3e5298b25b662c33256411be7277233e1a34570f7a4d142d931e4b5
ff8829e27aaf7eb2cc7f9ab655477d71c01d5da5aef44dd076b5820b4710ef025
a9e6c6b50a95af6105c5987c1b834d615008cf6370556ed00c6671e69776c09a9
2b5ac84804750dd867c78817bdf69f1443002b18ae7a52
token: 000187499b5930918d2d83ecebf92d25ca0722aa11b80dbbfd950537c2
8aa7d3a9df1949fd455872478ba87e2e6c513c3261cddbe57220581245e4c9c91
1dd1c0bb865785bff8f3cfe08cccbb3a7b8e41d23a172871be4828cc54582d87b
c7cfc5c8bcedc1868ebc845b000c317ed75312274a42b10be6db23bd8a168fd2f
021c23925d72c4d14cd7588c03845da0d41a326

// Test vector 4
skI: 22e237b7b983d77474e4495aff2fc1e10422b1d955192e0fbf2b7b618fba
625fcb94b599da9113da49c495a48fbf7f7f
```

```
pkI: 028cd68715caa20d19b2b20d017d6a0a42b9f2b0a47db65e5e763e23744f
e14d74e374bbc93a2ec3970eb53c8aa765ee21
token_challenge: 0001000e6973737565722e6578616d706c65000000
nonce:
02f0a206752d555a24924f2da5942a1bb4cb2d83ff473aa8b2bc3a89e820cd43
blind: af91d1dbcf6b46baecde70eb305b8fe75629199cca19c7f9344b8607b9
0def27bc53e0345ade32c9fd0a1efda056d1c0
token_request: 0001a503632ebb003ed15b6de4557c047c7f81a58688143331
2ad3ad7f9416f2dfc940d3f439ad1e8cd677d94ae7c05bc958d134
token_response: 032018bc3f180d9650e27f72de76a90b47e336ae9cb058548
d851c7046fa0875d96346c15cb39d8083cc6fb57216544c6a815c37d792769e12
9c0513ce2034c3286cb212548f4aed1b0f71b28e219a71874a93e53ab2f473282
71d1e9cbefc197a4f599a6825051fa1c6e55450042f04182b86c9cf12477a9f16
849396c051fa27012e81a86e6c4a9204a063f1e1722dd7
token: 000102f0a206752d555a24924f2da5942a1bb4cb2d83ff473aa8b2bc3a
89e820cd43085cb06952044c7655b412ab7d484c97b97c48c79c568140b8d49a0
2ca47a9cfb0a5cfb861290c4dbd8fd9b60ee9b1a1a54cf47c98531fe253f1ed6d
875de5a58f42db12b540b0d11bc5d6b42e6d17c2b73e98631e54d40fd2901ebec
4268668535b03cbf76f7f15a29d623a64cab0c4

// Test vector 5
skI: 46f3d4f562002b85ffcfdb4d06835fb9b2e24372861ecaa11357fd1f29f9
ed26e44715549ccedeb39257f095110f0159
pkI: 02fbe9da0b7cabe3ec51c36c8487b10909142b59af030c728a5e87bb3b30
f54c06415d22e03d9212bd3d9a17d5520d4d0f
token_challenge: 0001000e6973737565722e6578616d706c65205de58a52fc
daef25ca3f65448d04e040fb1924e8264acfccfc6c5ad451d582b30000
nonce:
9ee54942d8a1604452a76856b1bfaf1cd608e1e3fa38acfd9f13e84483c90e89
blind: 76e0938e824b6cda6c163ff55d0298d539e222ed3984f4e31bbb654a8c
59671d4e0a7e264ca758cd0f4b533e0f60c5aa
token_request: 0001e10202bc92ac516c867f39399d71976018db52fcab5403
f8534a65677ba9e1e7d9b1d01767d137884c86cf5fe698c2f5d8e9
token_response: 0322ea3856a71533796393229b33d33c02cd714e40d5aa4e0
71f056276f32f89c09947eca8ff119d940d9d57c2fcbd83d2da494ddeb37dc1f6
78e5661a8e7bcc96b3477eb89d708b0ce10e0ea1b5ce0001f9332f743c0cc3d47
48233fea6d3152fae7844821268eb96ba491f60b1a3a848849310a39e9ef59121
669aa5d5dbb4b4deb532d2f907a01c5b39efaf23985080
token: 00019ee54942d8a1604452a76856b1bfaf1cd608e1e3fa38acfd9f13e8
4483c90e89d4380df12a1727f4e2ca1ee0d7abea0d0fb1e9506507a4dd618f9b8
7e79f9f3521a7c9134d6722925bf622a994041cdb1b082cdf1309af32f0ce00ca
1dab63e1b603747a8a5c3b46c7c2853de5ec7af8cac7cf3e089cecdc9ed3ff05c
d24504fe4f6c52d24ac901471267d8b63b61e6b
```

A.2. Issuance Protocol 2 - Blind RSA (2048-bit)

The test vectors below list the following values:

skI: The PEM-encoded PKCS #8 RSA Issuer Private Key used for signing tokens, represented as a hexadecimal string.

pkI: The Issuer Public Key, serialized according to the encoding in Section 8.2.2.

token_challenge: A randomly generated TokenChallenge structure, represented as a hexadecimal string.

nonce: The 32-byte Client nonce generated according to Section 6.1, represented as a hexadecimal string.

blind: The blind used when computing the blind RSA blinded message, represented as a hexadecimal string.

salt: The randomly generated 48-byte salt used when encoding the blinded TokenRequest message, represented as a hexadecimal string.

token_request: The TokenRequest message constructed according to Section 6.1, represented as a hexadecimal string.

token_response: The TokenResponse message constructed according to Section 6.2, represented as a hexadecimal string.

token: The output token from the protocol, represented as a hexadecimal string.

```
// Test vector 1
skI: 2d2d2d2d2d424547494e2050524956415445204b45592d2d2d2d2d0a4d49
4945765149424144414e42676b71686b6947397730424151454641415343424b6
3776767536a41674541416f49424151444c477531726170583173633420a4f6b
7a38717957355379356b6f6a41303543554b66717444774e38366a424b5a4f764
57245526b49314c527876734d645332796132633616b4745714c756b440a556a
35743561496b317241764365584464e44503442325055707851436e6969396e6
b492b6d677257697444444949871386139793137586e6c5079596f784f530a646f
6558563835464f314a752b62397336356d586d34516a75513945596149713833
1724450567a50335758712b524e4d6363793232696866763624c766d42390a6a41
355334475666325a6c74785954736f4c364872377a58696a4e394637486271656
76f753967654b524d584645352f2b4a3956595a634a734a624c756570480a544f
72535a4d4948502b5358514d4166414f454a4547426d6d443068356672f43473
475676a79486e4e51383733414e4b6a55716d36765745744413872514c620a4530
742b496c706641674d4241414543676745414c7a4362647a69316a506435384d6
b562b434c6679665351322b7266486e7266724665502f566344787275690a3270
316153584a596962653645532b4d622f4d4655646c485067414c7731785134576
5726636336444373686c6c784c57535638477342737663386f364750320a6359
366f7770424477636261684747b556b5030456b6239533058c4a5763475347356
1556e484a585237696e7834635a6c666f4c6e724551653668557834d710a6230
6448786448444424d64476656577767 4b6f6a4f6a70532f39386d4555793756422
f3661326c7265676c766a632f326e4b434b745937374437645471 6c47460a787a
414261577538364d435a342f5131334c762b426566627174493973715a5a776a7
26455685148385643787279325156 4d515751696e57684174364d7154340a5342
5354726f6c5a7a7772716a65384d504a393175614e4d6458474c63484c4932367
3587a76374b53514b42675144766377735055557641395a325a583958350a6d49
784d54424e6445467a56625550754b4b413179576e31554d444e63556a71682b7
a652f376b337946786b683051463331627136306 54c393047495369414f0a354b
4f574d39454b6f2b784151326 2614b314d664f5931472b386a7a4258557042733
9346b35335338387958 6d4b366e7964677637304 24a385a6835666b55710a5732
306f5362686b686a5264537a48326b5247697267 2b5553774b426751445a4a4d6
e7279324578612f334571375062 6f737841504d69596e6b354a415053470a7932
7a305a375455622b7548514f2f2b78504d376e433075794c494d44396c61544d4
8776e3673372f4c62476f455031575267706f59482f4231346b2f526e360a6675
77524e3632496f397463392b41434c7455423776744476179332b6752775974534
33262356564386c4969656774546b656130683075445352784174567333 0a6e35
6b796132513976514b4267464a75467a4f5a742b7467596e576e515545 6757385
0304f494a45484d453455 54644f637743784b7248527239334a6a7546320a4533
77644b6f546969375072774f59496f614a5468706a50634a6262626462664b792b6
```

```
e735170315947763977644a724d6156774a6376497077563676315570660a5674
4c61646d316c6b6c7670717336474e4d386a6e4d30587833616a6d6d6e6665573
9794758453570684d727a4c4a6c394630396349324c416f4742414e58760a7567
5658727032627354316f6b6436755361427367704a6a5065774e526433635a4b3
97a306153503144544131504e6b7065517748672f2b36665361564f487a0a7941
7844733968355272627852614e6673542b72415548377783153594456565159564
d68555262546f5a6536472f6a716e544333664e6648563178745a666f740a306c
6f4d4867776570362b53494d436f6565325a6374755a5633326c6349616639726
2484f633764416f47416551386b3853494c4e4736444f413331544535500a6d30
31414a49597737416c5233756f2f524e61432b78596450553354736b75414c787
86944522f57734c455142436a6b46576d6d4a41576e51554474626e594e0a5363
77523847324a36466e72454374627479733733574156476f6f465a6e636d504c5
0386c784c79626c534244454c79615a762f624173506c4d4f39624435630a4a2b
4e534261612b6f694c6c31776d4361354d43666c633d0a2d2d2d2d2d454e44205
0524956415445204b45592d2d2d2d2d0a
pkI: 30820152303d06092a864886f70d01010a3030a00d300b06096086480165
03040202a11a301806092a864886f70d010108300b0609608648016503040202a
2030201300382010f003082010a0282010100cb1aed6b6a95f5b1ce013a4cfcab
25b94b2e64a23034e4250a7eab43c0df3a8c12993af12b111908d4b471bec31d4
b6c9ad9cdda90612a2ee903523e6de5a224d6b02f09e5c374d0cfe01d8f529c50
0a78a2f67908fa682b5a2b430c81eaf1af72d7b5e794fc98a3139276879757ce4
53b526ef9bf6ceb99979b8423b90f4461a22af37aab0cf5733f7597abe44d31c7
32db68a181c6cbbe607d8c0e52e0655fd9996dc584eca0be87afbcd78a337d17b
1dba9e828bbd81e291317144e7ff89f55619709b096cbb9ea474cead264c2073f
e49740c01f00e109106066983d21e5f83f086e2e823c879cd43cef700d2a352a9
babd612d03cad02db134b7e225a5f0203010001
token_challenge: 0002000e6973737565722e6578616d706c65208e7acc900e
393381e8810b7c9e4a68b5163f1f880ab6688a6ffe780923609e88000e6f72696
7696e2e6578616d706c65
nonce:
aa72019d1f951df197021ce63876fe8b0a02dc1c31a12b0a2dd1508d07827f05
blind: 425421de54c7381864ce36473abfb988c454fe6c27de863de702a6a2ad
ca153fa2de47bd8fcd62734caa8ce1f920b77d980ab58c32d16dde54873f28ca9
68e8c125b8363514be68972f553655bcc7f80a284cc327e47e804a47333c5b3cd
f773312cc7ad9fda748aed0baa7e19c5a2d1dafda718f086d7fc0a4bc02d488e0
f20812daee335af7177b7a8369bd617066aed7a58f659f295c36b418827f67972
5b81ca14ea16fb82df21ad76da1ac38dcf24bf6252f8510e2308608ac9197f6cb
54fdcb19db17837302a2b87d659c5605f35f3709a130f0c3d50e172f0cae36cbc
9467f9914895a215a9e32443bcafff795273ccf8965a7eaa8c0b2184763e3e5c
salt: 3d980852fa570c064204feb8d107098db976ef8c2137e8641d234bbd88a
986fdb306a7af220cfadede08f51e1ef61766
token_request: 0002086a95be84b63cfed0993bb579194a72a95057e1548ac4
63a9a5b33b011f2b2011d59487f01862f1d8e4d5ea42e73a660fbc3d010b944a5
4da3a4e0942f8894c0884589b438cb902e9a34278970f33c16f351f7dae58d273
c3ab66ef368da36f785e89e24d1d983d5c34311cd21f290f9e89e8646ab0d0a48
988fcd46230de5e7603cd12cc95c7ec5002e5e26737aa7eb69c626476e6c8d465
10ee404a3d7daf3a23b7c66735d363ca13676925c6ed0117f60d165ce1f8ba616
d041b6384baf6da3e2f757cb18e879a4f8595c2dc895ddf1f4279c75768d108b5
c47f95f94e81e2d8b9c8b74476924ab3b7c45243fc99ac5466e8a3680ad37fa15
c96010b274094
token_response: 675d84b751d9e593330ec4b6d7ab69c9a61517e98971f4b73
6150508174b4335761464f237be2d72bbae4b94dffc6143413f6351f1aa4efde6
c32d4d6d9392a008290d56d1222f9b77a1336213e01934f7d972f3bf9ea5a5786
c321352f103b3667e605379a55f0fb925fbb09b8a9f85e7dd4b388a3b49d06fd7
0ba28f6a780e3bc8f6421554fd6c38b63ef19f84ccfcf14709dd0b4d72213c1f0
60893854eba0ea1a147e275da320db5e9849882d5f9179efa8a2d8d3b803f9d14
45ef5c1f660be08883ce9b29a0a992fc035d2938cbb61c440044438dbb8b3ce71
58a8f9827d230482f622d291406ab236b32b122627ae0fd36bd0d6b7607b8044a
```

```
ce404d44
token: 0002aa72019d1f951df197021ce63876fe8b0a02dc1c31a12b0a2dd150
8d07827f055969f643b4cfda5196d4aa86aeb5368834f4f06de46950ed435b3b8
1bd036d44ca572f8982a9ca248a3056186322d93ca147266121ddeb5632c07f1f
71cd2708bc6a21b533d07294b5e900faf5537dd3eb33cee4e08c9670d1e5358fd
184b0e00c637174f5206b14c7bb0e724ebf6b56271e5aa2ed94c051c4a433d302
b23bc52460810d489fb050f9de5c868c6c1b06e3849fd087629f704cc724bc0d0
984d5c339686fcdd75f9a9cdd25f37f855f6f4c584d84f716864f546b696d620c
5bd41a811498de84ff9740ba3003ba2422d26b91eb745c084758974642a420782
01543246ddb58030ea8e722376aa82484dca9610a8fb7e018e396165462e17a03
e40ea7e128c090a911ecc708066cb201833010c1ebd4e910fc8e27a1be467f786
71836a508257123a45e4e0ae2180a434bd1037713466347a8ebe46439d3da1970

// Test vector 2
skI: 2d2d2d2d2d424547494e2050524956415445204b45592d2d2d2d2d0a4d49
4945765149424144414e42676b71686b6947397730424151454641415343424b6
3776767536a41674541416f49424151444c4775317261705831736334420a4f6b
7a38717957355379356b6f6a41303543554b66717444774e38366a424b5a4f764
57245526b49314c527876734d645332796132633616b4745714c756b440a556a
35743561496b3172417643655844644e44503442325055707851436e6969396e6
b492b6d677257697444444948713861397931375866e6c5079596f784f530a646f
6558563835464f314a752b62397336356d586d34516a7551394559614971383 37
1724450567a50335758712b524e4d636379323269686763624c766d42390a6a41
35533447566632 5a6c74785954736f4c364872377a58696a4e394637486271656
76f753967654b524d584645352f2b4a3956595a634a734a624c756570480a544f
72535a4d4948502b5358514d4166414f454a4547426d6d4430683566672f43473
475676a79486e4e51383733414e4b6a55716d3676574574413872514c620a4530
742b496c706641674d42414145436767745414c7a4362647a69316a506435384d6
b562b434c6679665351322b7266486e7266724665502f566344787275690a3270
316153584a5969626536455326b4d622f4d4655646c485067414c773178513457 6
5726636633644437368 6c6c784c57535638477342737663386f364750320a6359
366f777042447763626168474b556b5030456b62395330584c4a5763475347356
1556e484a585237696e7834635a6c666f4c6e72455165366855 78734d710a6230
644878644844424d644766565777674b6f6a4f6a70532f39386d4555793756422
f3661326c7265676c766a632f326e4b434b745937374437645471 6c47460a787a
414261577538364d435a342f5131334c762b42656662717449397 3715a5a776a7
26455685148385643787279325156 4d515751696e57684174364d7154340a5342
5354726f6c5a7a7772716a65384d504a393175614e4d6458474c63484c4932367
3587a76374b53514b42675144766 3777350555576 41395a325a583958350a6d49
784d54424e6445467a56625550754b4b413179576e31554d444e63556a71682b7
a652f376b337946786b68305146333162713 630654c393047495369414f0a354b
4f574d39454b6f2b7841513262614b314d664f5931472b386a7a425855704 2733
9346b353353383879586d4b366e796467763730424a385a6835666b55710a5732
306f5362686b686a5264537a48326b52476972672b5553774b426751445a4a4d6
e7279324578612f334571375062 6f737841504d69596e6b354a415053470a7932
7a305a375455622b7548514f2f2b78504d376e433075794c494d44396c61544d4
8776e3673372f4c62476f45503157526 7706f59482f4231346b2f526e360a6675
77524e3632496f397463392b41434c74554237767447 6179332b675277597 4534
33262356564386c4969656774546b6561306830754453527841745673330a6e35
6b796132513976514b4267464a75467a4f5a742b7467596e576e5155456757385
0304f494a45484d45345554644f637743784b7248527239334a6a7546320a4533
77644b6f546969375072774f59496f614a5468706a50634a62626462664b792b6
e735170315947763977644a724d6156774a637649707756367631557 0660a5674
4c61646d316c6b6c767071 7336474e4d386a6e4d30587833616a6d6d6e6665573
9794758453570684d727a4c4a6c3946303963493 24c416f4742414e58760a7567
565872703262 7354316f6b64367553614273 67704a6a5065774e526433635a4b3
97a30615350314454413150 4e6b706551774867 2f2b36665361564f487a0a7941
78447339683552726278 52614e6673542b7241554837783153594456565159564
```

```
d68555262546f5a6536472f6a716e544333664e6648563178745a666f740a306c
6f4d4867776570362b53494d436f6565325a6374755a5633326c6349616639726
2484f633764416f47416551386b3853494c4e4736444f413331544535500a6d30
31414a49597737416c5233756f2f524e61432b78596450553354736b75414c787
86944522f57734c455142436a6b46576d6d4a41576e51554474626e594e0a5363
77523847324a36466e724543746274797337335574156476f6f465a6e636d504c5
0386c784c79626c534244454c79615a762f624173506c4d4f39624435630a4a2b
4e534261612b6f694c6c31776d4361354d43666c633d0a2d2d2d2d2d454e44205
0524956415445204b45592d2d2d2d2d0a
pkI: 30820152303d06092a864886f70d01010a3030a00d300b06096086480165
03040202a11a301806092a864886f70d010108300b0609608648016503040202a
2030201300382010f003082010a0282010100cb1aed6b6a95f5b1ce013a4cfcab
25b94b2e64a23034e4250a7eab43c0df3a8c12993af12b111908d4b471bec31d4
b6c9ad9cdda90612a2ee903523e6de5a224d6b02f09e5c374d0cfe01d8f529c50
0a78a2f67908fa682b5a2b430c81eaf1af72d7b5e794fc98a3139276879757ce4
53b526ef9bf6ceb99979b8423b90f4461a22af37aab0cf5733f7597abe44d31c7
32db68a181c6cbbe607d8c0e52e0655fd9996dc584eca0be87afbcd78a337d17b
1dba9e828bbd81e291317144e7ff89f55619709b096cbb9ea474cead264c2073f
e49740c01f00e109106066983d21e5f83f086e2e823c879cd43cef700d2a352a9
babd612d03cad02db134b7e225a5f0203010001
token_challenge: 0002000e6973737565722e6578616d706c6500000e6f7269
67696e2e6578616d706c65
nonce:
98c1345ff38a554b429b428b0f206cfe4f3892f8041995f2c24873d90e84488d
blind: 7bb85f89c9b83a0e2b02938b3396f06f8f3df0018a91f1a2cc5416aaa5
52994d063f634d50bea13bffe8d5e01431e646e2e384549cefd695ac3affff665
a1ebf0113df2520006bd66e468d37a58266daa8a3a75692535e1fc46d0c1d6fb6
f37c949808172e20c0b77a48570a1fcb474325bdd23cdbce52b5d6a9e39f7aec7
3b09004eae8c8bfff2b4b533ea63bcf467a4cd95ccfb0cb4e43bc4992c1fd0be7
a77a4475dbf8094cf25125ece901abbcea607a9050ad9f8ec3d0d66341f6eab40
ee9c9c22c0b560b8377f8543d8878c7458885fd285c7556cc88fc6021617075b4
2c83a86005169a6f13352e789b28fdbbe3d0288e1dd7c801497573893146aea3
salt: b6b4378421ab0ea677ce3f4036fd0489dee458ad81ea519c3e8bde3fcd5
ec1505d28e110d7b44dcac5e04ecedd54d11a
token_request: 00020892d26a271c0104657ba10c0b5cb2827bb209d86e8002
7f96bfb861e0f40cb897f0fc426498433141ce9bc8b4a95914fefe4e40bdd3802
a121cb0b59a4ae7e03255275c4abf071d991c82ead402606c0ef912178b0a0f68
d303e06a966079230592827b84979dbcb5f21ab8904e9908638ddf705c4f8af8a
053c19a66090726b60c6b4063976e4c66eab33522dd3f9d64828441db4aa82d55
adcc3d3920592884cd1e5a3f490d5c81f1306705dcc5c61d82373f1dbd7d2ae4b
2fea0f7339f5d868415f59312766e3074ee4a7305f5f053da82673ee6747a727a
26d8d10ea1b1a3491d26b0c38b962c02a774ac78932153aae9dcc98a9b1db1f53
89644682f7727
token_response: 113a5124c1aef6fc230d9fc42b789226f45ca941aad4da3f4
8cf37c7744a8d7fd1dcfd71cd39d09e9324760180ea0bade3360efaf7322a1fa1
5f41247be3857fde8c5c92ec6d67a7ee33be8fdadf8b27bb0db706117448e55bc
e9927cb6bfb1f87f9edb054181a4558af0c0d3973d7033b9599e674c20cf08a7b
bcf0da815a2edaab7c4fb80dee4ea2cc53576a9691e857da931c6c592d2c69dd2
1afda8ea653dd90157adfe80e2375c08e75beb497df8b7b73192fbbd4e80359d9
bbaecea14e0acebdda92596f71ec1d57e26b6497b3152976bc07a4409148cb843
89eb207fb8e841106012408c6e19b4f964008b6a909aaab767a661a061c97da16
43040455
token: 000298c1345ff38a554b429b428b0f206cfe4f3892f8041995f2c24873
d90e84488d11e15c91a7c2ad02abd66645802373db1d823bea80f08d452541fb2
b62b5898bca572f8982a9ca248a3056186322d93ca147266121ddeb5632c07f1f
71cd27083350a206c5e9b7c0898f97611ce0bb8d74d310bb194ab67e094e32ff6
da90886924b1b9e7b569402c1101d896d2fc3a7371ef77f02310db1dc9f81c853
5828c2d0e9d9051720d182cd54e1c2c3bf417da2fc7aa72bb70ccc834ef274a2e
```

```
809c9821b3d395d6535423f7428b3f29175d6eb840b4b7685336e57e2b6afeaab
c0c17ea4f557e8a9cc2f624e245c6ccd7cbdd6c32c97c5c6974e802f688e2d25f
0aba4215f609f692244517d5d3407e0172273982c001c158f5fcbe1b5d2447c26
a87e89f5a9e72b498b0c59ce749823d2cf253d3cf6cd4e64fa0e434d95e488789
247a9ceed756ff4ff33a8d2402c0db381236d331092838b608a42002552092897

// Test vector 3
skI: 2d2d2d2d2d424547494e2050524956415445204b45592d2d2d2d2d0a4d49
4945765149424144414e42676b71686b6947397730424151454641415343424b6
3776767536a41674541416f49424151444c477531726170583173633442
0a4f6b
7a38717957355379356b6f6a41303543554b66717444774e38366a424b5a4f764
57245526b49314c527876734d6453327961326333616b4745714c756b440a556a
35743561496b317241764365584464 4e4450344232505570785143 6e6969396e6
b492b6d677257697444444948713861397931375 86e6c5079596f784f530a646f
6558563835464f314a752b62397336356d586d34516a7551394559614971 38337
1724450567a50335758712b524e4d636379323269686763624c766d42390a6a41
355334475666325a6c74785954736f4c364872377a58696a4e39463748627165 6
76f753967654b524d584645352f2b4a3956595a634a734a624c756570480a544f
72535a4d4948502b5358514d4166414f454a4547426d6d4430683566672f43473
475676a79486e4e51383733414e4b6a55716d36765745744138725 14c620a4530
742b496c706641674d4241414543676745414c7a4362647a69316a506435384d6
b562b434c6679665351322b7266486e7266724665502f566344787275690a3270
316153584a59696265364553 2b4d622f4d4655646c485067414c7731785134576
57266366336444373686c6c784c57535638477342737663386f364750320a6359
366f7770424477636261 68474b556b5030456b62395330584c4a5763475347356
1556e484a585237696e7834635a6c666f4c6e72455165 366855787 34d710a6230
644878644844424d644766565777674b6f6a4f6a70532f39386d4555793756422
f3661326c7265676c766a632f326e4b434b745937374437645 4716c47460a787a
4142615775 38364d435a342f5131334c762b426566662717449397371 5a5a776a7
2645568514838564378727932 51564d515751696e57684174364d7154340a5342
5354726f6c5a7a7772716a65384d504a393175614e4d6458474c63484c4932367
3587a76374b53514b42675144766 3777350555576 41395a325a583958350a6d49
784d54424e6445467a5662555 0754b4b413179576e31554d444e63556a71682b7
a652f376b337946786b6830514633316271363065 4c393047495369414f0a354b
4f574d39454b6f2b784151 3262614b314d664f5931472b386a7a4258557042733
9346b3533533838795 86d4b366e79646776373 0424a385a6835666b55710a5732
306f5362686b686a5264537a48326b52476972672b555377 4b426751445a4a4d6
e727932457861 2f3345713750626f737841504d69596e6b354a415053470a7932
7a305a375455622b75485 14f2f2b78504d376e433075794c494d44396c61544d4
8776e3673372f4c62476f45503157526770 6f59482f4231346b2f526e360a6675
77524e3632496f397463392b41434c7455423776744761793 32b6752775974534
332623565643 86c4969656774546b6561306830754453527841745 673330a6e35
6b79613251397651 4b4267464a75467a4f5a742b7467596e576e5155456757385
0304f494a45484d45345554644f637743784b7248527 239334a6a7546320a4533
77644b6f546969375072774f59496f614a5468706a50634a62626462664b792b6
e735170315947763977644a724d6156774a637649707756367631557 0660a5674
4c61646d316c6b6c7670717336474e4d386a6e4d30587833616a6d6d6e6665573
97947584535706 84d727a4c4a6c394630396349324c416f4742414e58760a7567
56587270326273543 16f6b6436755361427367704a6a5065774e526433635a4b3
97a306153503144544131504e6b7065517748672f2b36665361564f487a0a7941
784473396835527262785 2614e6673542b724155483778315359445656515 9564
d68555262546f5a6536472f6a716e544333664e66485631 78745a666f740a306c
6f4d4867776570362b53494d436f6565325a63747 55a5633326c6349616639726
2484f633764416f47416551386b3853494c4e4736444f41333154453550 0a6d30
31414a49597737416c5233756f2f524e61432b7859645055335473 6b75414c787
86944522f57734c455142436a6b46576d6d4a41576e51554474626e594e0a5363
77523847324a36466e72454374627479733733574156476f6f465a6e636d504c5
0386c784c79626c534244454c79615a762f624173506c4d4f39624435630a4a2b
```

```
4e534261612b6f694c6c31776d4361354d43666c633d0a2d2d2d2d2d454e44205
0524956415445204b45592d2d2d2d2d0a
pkI: 30820152303d06092a864886f70d01010a3030a00d300b0609608648016503040202a11a301806092a864886f70d010108300b0609608648016503040202a
2030201300382010f003082010a0282010100cb1aed6b6a95f5b1ce013a4cfcab
25b94b2e64a23034e4250a7eab43c0df3a8c12993af12b111908d4b471bec31d4
b6c9ad9cdda90612a2ee903523e6de5a224d6b02f09e5c374d0cfe01d8f529c50
0a78a2f67908fa682b5a2b430c81eaf1af72d7b5e794fc98a3139276879757ce4
53b526ef9bf6ceb99979b8423b90f4461a22af37aab0cf5733f7597abe44d31c7
32db68a181c6cbbe607d8c0e52e0655fd9996dc584eca0be87afbcd78a337d17b
1dba9e828bbd81e291317144e7ff89f55619709b096cbb9ea474cead264c2073f
e49740c01f00e109106066983d21e5f83f086e2e823c879cd43cef700d2a352a9
babd612d03cad02db134b7e225a5f0203010001
token_challenge: 0002000e6973737565722e6578616d706c6500017666f6f
2e6578616d706c652c6261722e6578616d706c65
nonce:
9e7a22bdc5d715682434cebc07eb5fa53f622f776a17a6d91757af1592df0e71
blind: c52cabc5e4e131e0f5860cc4c486c5ee8a5fa8ae59484446121f87b0d8
ccd037f161a99ebcc57f79d05a2ffc852656ad2d0894fab8d1b0f998e6e678254
ed5778da98b137371320314d06c24276e35435bccffa49d257687f270f9ce1792
6a074737546d5415a4bb9e624a6302562b395856632efb6992f6593a4f95fb342
002efebc3046ca96bbc26edb2f1a1454a24ce7b9a7ec8e44fb9e99c8144d409d8
cd8a5903c0a3c0acbd9f82573ed1fc4a296e3eaf4867ade30110794678f422d36
bd103ea4617d2472cf58da3381e52e5be60f4acbf685e280648cef21211a796ec
d005ecbdaa1046c40950afca4c4e7dd4b8c19e504088489a15667b45895b6e92
salt: c847b5d0fa9101a1e09954ac9f3eed6600af58936295ad2e54274e13e64
0d59f732d07530c94c19c20668f03470c77ac
token_request: 0002080f6bd84fba1822c577c8cd670f1136cca107f84ddd9d
405d5ed22ad15da975538f031433bad4a2688999732927efe2928d4c132389a12
2f40b639b083d6fcbbed7a55fb18db536d2dcbaefe6dc0a70730e6565b08a7dfd
783913a59f37d798de0cfc262c9e90a7ee884a3ec355eacbd44e5f6779fea6a78
5b05ac352fdd51a116cf2be1d8e38b0bfacd6a3d53a88c99f747cce908f86b335
62691f540e3e88562092cd17cc2f78ce0fb53312a5f2dc918bdb1dc90d9d65091
c7ba9080ccc1755cb5437989364dc92f0e8fea18f66d631451feb02a3d68af41d
e1a3f9be925dda5c4ca0706fc4ca28b3317e939f6573442c6d03be17cd141fa82
60d382d134c6b
token_response: 2dd08ce89cf4f62bc236ab7b75266e13c57c750345e328e0b
ea107537c4cbeea5bfc990716950440628ea2e37dbc5c9c6d84f9a965cbf0cbff
fb89516b1fd19a90d69cc52a28890bbdcf782f56aefadad85b6e861a74170ce91
0891c89e4293f37978dbd41cc8b5c68802de3d86d9f0326b9c22b809512245896
6a6ddd1aeb3828d239c3b359efc9b375390eb19050d5656c2b084304d9bd8a816
14f631bf82a7e4588413b44a0cb6d94e942fa134790b396cb71e3ed33b557b5bd
0734e726fa79abdca8694703b81d0e289b749801d4383e0d4f825dcde0dd98c43
d3ba81c028dd8833a4fc24961f60e118d4421dce5b611d53e9ca96156a52509bf
a9afeb7e
token: 00029e7a22bdc5d715682434cebc07eb5fa53f622f776a17a6d91757af
1592df0e710042eee45ac4dd5acb8f6e65c4d8dd47504f73f7463507ef96a4d72
27d2774f3ca572f8982a9ca248a3056186322d93ca147266121ddeb5632c07f1f
71cd270815b010bbc0d5f55e9c856d2e9ffaefba007d33c2d5452fbeb0b15919b
973e0dc9180aaeb18242043758d9fb0ac9ac5e04da9ff74ec93644ae6cdb7068e
a76ce2295b9b95e383ed3a9856e9f618dafdf4cec5d2b53ea4297c2f3990babca
71e3ccd6c07a437daae7ed27b6b81178fb7ce5fa5dd63781cc64ac1e410f441c0
34b0a5cc873a2ce875e8b38c92bab563635c4f8f4fa35d1f582ef19edf7da75aa
11a503a82e32a12bd4da41e0ca7ec7f451caf586f5b910003fcbbb9ff5ffa2408
c28d6807737d03da651ea9bfafcc2747a6830e19a1d160fcd5c25d2f79dad86a8
b3de8e926e08ca1addced72977f7b56398ef59c26e725df0a976a08f2a936ca42

// Test vector 4
```

```
skI: 2d2d2d2d2d424547494e2050524956415445204b45592d2d2d2d2d0a4d49
4945765149424144414e42676b71686b6947397730424151454641415343424b6
3776767536a41674541416f49424151444c477531726170583173633442 0a4f6b
```

```
0a78a2f67908fa682b5a2b430c81eaf1af72d7b5e794fc98a3139276879757ce4
53b526ef9bf6ceb99979b8423b90f4461a22af37aab0cf5733f7597abe44d31c7
32db68a181c6cbbe607d8c0e52e0655fd9996dc584eca0be87afbcd78a337d17b
1dba9e828bbd81e291317144e7ff89f55619709b096cbb9ea474cead264c2073f
e49740c01f00e109106066983d21e5f83f086e2e823c879cd43cef700d2a352a9
babd612d03cad02db134b7e225a5f0203010001
token_challenge: 0002000e6973737565722e6578616d706c65000000
nonce:
494dae41fc7e300c2d09990afcd5d5e1fc95305337dc12f78942c45340bfe8e6
blind: 097cb17bcedecfe058dff5c4e517d1e36d7ab8f46252b1ac1933ba378c
32625c0dbc69f5655c2003bf39e75810796cd63675b223cf3162c57108d56e058
4cfce6cad829e74369ada38a095eb3012c912b31ccde7425f93464e353fb17552
be3a8df2913daca61543a33ae45058f218c471dfbc12fb304158e29b6ed35bc07
9e23f1e6173c5dec4545840bbe58e5ad37cbea0a10dca5d9df2781589d27c3410
8477b52c0d32a1370c17f703941fbb1a007a6794e7de2758709c9bbf80f21eec7
922b9bb491eb6aac8c1a14764e648e6be4fff0ae913797067aa0826f366c3103e
103b05653c73b52d7f825a185dccfb806da700db9f53abb848554b7d4f7c28f3
salt: 49912979f1bf528e5b8228ab1328df74319dce7bdaf45821ceb1100dcf0
42a2dfe852fc9db59b64a5f6493c282504240
token_request: 000208244840027ca8c620f8b14caded9a198ba388ccd8541e
962f68a0071535d958d18494afd0bc11da4da8c8b33864f5a8f623b697cd56348
594e11a75479048a72c0ed179b070506c09a7eb6ed3582f572df38cf60fcde11a
52c5ce6d7b23435b60200ad9f66d21f40f323c9aa54307d0b966d4457c37542b6
6bb183ddeafca914fc74831698b5d52f498ee3d165685f49a8d86e39fe6c4b7ec
678f5250908d25e5b873c69b422368121aa4210cadd6fc640907d3cb9a7a3e827
a0e742470f00c2f49dc6c0e8cc9470dbfd73df0ccbb96c10b02af0dd7dee719ec
a11ff8e1b4929e59f3cf319de9bda29a6d968b43083b5d4242f3448d76ada08b8
014f70b97e719
token_response: c2746ff644cffb28a2c19395fa19dfb61fd135daa837844fb
f9fbe06c253e64e69f53aefddc0fb4833b1b5e58f571134a34f245499c3e73419
549c2c9111cf94f2f68fea3996d47f71e8d8d6fc5b1c074bf74fa59de4cbf32f5
f08d45ea45492f0279c3b1a8d852698edbe1651eb8e09eb223a27386c0feb2f6a
8260235edb36cf433da518100829b63166284b325d87fc941ea3bafe7b6761b70
82e09397837f74b4f0fc838bce8af7242089dd5561f57735926bcbad219fc9fee
85ae49a8e8951f63ca194b7ff018c06ee02267e7267bb996432dc76973819da80
e3e86947b0a4b36d3a972dafaaa3db0e1044b325f02c679996d9bcd3ce51390d5
4bc10b8c
token: 0002494dae41fc7e300c2d09990afcd5d5e1fc95305337dc12f78942c4
5340bfe8e6b741ec1b6fd05f1e95f8982906aec1612896d9ca97d53eef94ad3c9
fe023f7a4ca572f8982a9ca248a3056186322d93ca147266121ddeb5632c07f1f
71cd2708a55c83dc04292b5d92add1a87b37e54f22f61c58840586f390c50b231
824423378ddcf50e69dc817d45bfad06c7f2a0ac35d2acd7f26b0bc9954c192b0
a0ef28a2a5650e390098dd3cb1166a7cb1716d3dd2d19dc5ca3b1ea6206359de0
002d82bc4fa7e69fb07214b06addcbd2203d1e17f57fc580bcc5a13e0ac15cf94
2182cc2b5d6eaa737a712704114e357e2ec2f10047463ded02a1a0766dc346dd7
212b9711e03ac95eb258ac1164104dc9a0d3e738ae742ab5ed8c5139fc07145a7
88b9f891741ee68f0a66782b7b84a9bb4cb4b3d1b26b67106f397b35b641d882d
7b0185168946de898ef72349a44a47dbdd6d46e9ba9ba543d5701b65c63d645c2

// Test vector 5
skI: 2d2d2d2d2d424547494e2050524956415445204b45592d2d2d2d2d0a4d49
4945765149424144414e42676b71686b6947397730424151454641415343424b6
3776767536a41674541416f49424151444c4775317261705831736334420a4f6b
7a38717957355379356b6f6a41303543554b66717444774e38366a424b5a4f764
57245526b49314c527876734d6453327961326333616b4745714c756b440a556a
35743561496b3172417643655844644e445034423250557078514367e6969396e6
b492b6d6772576974444494871386139793137586e6c5079596f784f530a646f
6558563835464f314a752b62397336356d586d34516a7551394559614971383337
```

```
1724450567a50335758712b524e4d636379323269686763624c766d42390a6a41
3553344756663 25a6c74785954736f4c364872377a58696a4e394637486271656
76f753967654b524d584645352f2b4a3956595a634a734a624c756570480a544f
72535a4d4948502b5358514d4166414f454a4547426d6d4430683566672f43473
475676a79486e4e51383733414e4b6a55716d367657457441387251 4c620a4530
742b496c706641674d4241414543676745414c7a4362647a69316a506435384d6
b562b434c6679665351322b7266486e7266724665502f566344787275690a3270
316153584a596962653645532b4d622f4d4655646c485067414c7731785134576
5726636633644437368 6c6c784c57535638477342737663386f364750320a6359
366f77704244776362616847 4b556b5030456b62395330584c4a5763475347356
1556e484a585237696e7834635a6c666f4c6e724551653668557 8734d710a6230
644878644844424d644766565777674b6f6a4f6a70532f39386d4555793756422
f3661326c7265676c766a632f326e4b434b745937374437645471 6c47460a787a
414261577538364d435a342f5131334c762b426566627174493973715a5a776a7
26455685148385643787279325156 4d515751696e57684174364d7154340a5342
5354726f6c5a7a7772716a65384d504a393175614e4d6458474c63484c4932367
3587a76374b53514b42675144766377735055557641395a325a583958350a6d49
784d54424e6445467a5662555075 4b4b413179576e31554d444e63556a71682b7
a652f376b337946786b683051463331627136306 54c393047495369414f0a354b
4f574d39454b6f2b784151326261 4b314d664f5931472b386a7a4258557042733
9346b353353383879586d4b366e796467763730424a385a6835666b55710a5732
306f5362686b686a5264537a48326b52476972672b5553774b4267514 45a4a4d6
e7279324578612f3345713750626f737841504d69596e6b354a415053470a7932
7a305a375455622b7548514f2f2b78504d376e433075794c494d44396c61544d4
8776e3673372f4c62476f455031575267706f59482f4231346b2f526e360a6675
77524e3632496f397463392b41434c745542377767 4476179332b675277597453 4
33262356564386c4969656774546b656130683075445352784174567 3330a6e35
6b79613251 3976514b4267464a75467a4f5a742b7467596e576e515545675 7385
0304f494a45484d45345554644f637743784b7248527239334a6a7546320a4533
77644b6f546969375072774f59496f614a5468706a50634a6262646 2664b792b6
e73517031594776397 7644a724d6156774a637649707756367631557 0660a5674
4c61646d316c6b6c7670717336474e4d386a6e4d30587833616a6d6d6e6665573
9794758453570684d727a4c4a6c394630396349324c416f4742414e58760a7567
5658727032627354316f6b643675536142736 7704a6a5065774e526433635a4b3
97a306153503144544131504e6b7065517748672f2b36665361564f487a0a7941
7844733968355272627852614e6673542b7241554837783153594456565515 9564
d68555262546f5a6536472f6a716e544333664e664856317 8745a666f740a306c
6f4d4867776570362b53494d436f6565325a6374755a5633326c6349616639726
2484f633764416f47416551386b3853494c4e4736444f413331544535500a6d30
31414a49597737416c5233756f2f524e61432b7859645 0553354736b75414c787
86944522f57734c455142436a6b46576d6d4a41576e51554474626e594e0a5363
77523847324a36466e72454374627479733733574156 6476f6f465a6e636d504c5
0386c784c79626c534244454c79615a762f624173506c4d4f39624435630a4a2b
4e534261612b6f694c6c31776d4361354d43666c633d0a2d2d2d2d2d454e44205
0524956415445204b45592d2d2d2d2d0a
pkI: 30820152303d06092a864886f70d01010a3030a00d300b0609608648016 5
03040202a11a301806092a864886f70d010108300b0609608648016503040202a
2030201300382010f003082010a0282010100cb1aed6b6a95f5b1ce013a4cfcab
25b94b2e64a23034e4250a7eab43c0df3a8c12993af12b111908d4b471bec31d4
b6c9ad9cdda90612a2ee903523e6de5a224d6b02f09e5c374d0cfe01d8f529c50
0a78a2f67908fa682b5a2b430c81eaf1af72d7b5e794fc98a3139276879757ce4
53b526ef9bf6ceb99979b8423b90f4461a22af37aab0cf5733f7597abe44d31c7
32db68a181c6cbbe607d8c0e52e0655fd9996dc584eca0be87afbcd78a337d17b
1dba9e828bbd81e291317144e7ff89f55619709b096cbb9ea474cead264c2073f
e49740c01f00e109106066983d21e5f83f086e2e823c879cd43cef700d2a352a9
babd612d03cad02db134b7e225a5f0203010001
token_challenge: 0002000e697373756572 2e6578616d706c65208e7acc900e
393381e8810b7c9e4a68b5163f1f880ab6688a6ffe780923609e880000
```

```
nonce:
a1aa8b371c37c9a8ddbd7342ab4f9dd5227d5b1600dca6517b60f63143cd43a3
blind: ad7a32e1ac31b91daefd7042cc23d5621ab3e870d87297bbfe1ee8a518
ffc5b84770d3b77775c485b2d2199548348688842d2f11877ac4bceb5da88944cc
a043a9afa52f9c9998a5dea7ab7c1f82662d0d327e29705a269ad221ae74a7c11
72ff89c48997a9fda08886d3998bb538868396c0ace71d260cc71f768001939b2
4d80d88979f0244a3dbc004eadfac81e138d430b9fa51c1aad21b957ff96b3123
c91c2fff362a386f0f99a3f9fc906ca626fd9107648f87532b44c4fe3856ecae1
f46d8ebf5d2f46e52034478e5e883015666574dd80bd5c036c4b55ebcc8b66068
8d23944cc1932d075b559dcdc269fae3511761f71c113634e60d67accc8875fb
salt: 35c04710ce866d879447b6230ce098a49e81be5c067881cce7bd5f92c1e
5bd9b3c7d4d795cfad134fdfe916d735a624a
token_request: 0002083d6495c72529bbc4f5c0b49e94e4561baec1ca638a93
b2940ea9e37b838db7b1a91ec1f257d49b45c4f75119c2ab9eb5578541ad2b9ba
c1bd627abc709097f503f83d98fed6dbeb615c3be9bf09cbf8ea25ea8026c1b8b
a1c704ff516ed87c3d7d85342fd00111d8a80492d4b8fdbb092a282f74f13901e
5edc1b3b02cfe24c950affe6130fbb57c1482d674db3c6944812ba081c2235a16
d01eeec0932a8619d85732fc3e36179f0b50377bf9cb7a50ce3abeb3f31ed5f0f
3deec7aae7290f5397cec61318357d652b029a0fda0f100a78e36c4ef56ba3779
963e8745fdf4e347763c63d825836878e249833a0f4bd315392cc06ccca2c955e
921efbc4f941d
token_response: 8db727000018a98a2fe9fda8bbde5b8e9cedc31efbcaed695
0eb1e0f8d9af9db632def52f74f07cdab304bbde40519080dd0388fb2b8900528
b4791d2bca40aa2c2a6d1b92f010c1849bfb781cc813cc204855dd05e8a2dd31e
a5220981b8ab6b008e153083dc8f594206440d66286fea9c21b56807be8655506
ab7818bb9c8c69489dda56fe6390a5397268c8b5711f9d2df6f2584740cccf034
5fd67f93f345426f33c078a0aceb90845df9eef74f6248d06c36d19e191da325b
721ddc12ea78ed37b0c3b6170590536e3aee7eb0efc7d11a2c9d072a394f12ffa
67ecf316c49efd8f31723b11fe46740636bd89ad4f7ef96bc38b2cb4916d9dc04
ba1b2fc6
token: 0002a1aa8b371c37c9a8ddbd7342ab4f9dd5227d5b1600dca6517b60f6
3143cd43a3bb8a8cf1c59e7a251358ed76fe0ccff61044bc79dd261f16020324d
22f2d434cca572f8982a9ca248a3056186322d93ca147266121ddeb5632c07f1f
71cd27082899f4bc385b4147a650a3e7efc7d23a743cb944bb53e2ed8b88ee03a
9c0b1cf1f73fd7f15c1bd1f850a5a96a34d4ee9f295543f30ac920825e9a0ce26
e924f2c145583019dd14408c565b660e8b702fbea559908b1a8c8f9d21ef22f7c
c6a4641c57c146e6c5497d2890ca230a6749f5f83a8fdd79eba0722f10dff9e81
a2fb2d05fa4d989acc2e93f595ae69c98c3daa3b169efcdd6e59596b2f049f16b
a49528761f661032da65a3ee0fe8a22409766e90daf8c60323c16522818c49273
c795f26bbab306dc63cfc16fe1702af2464028cf819cc647d6f9b8a8f54d7d658
5a268fdb8c75f76618c64aba266836a3b2db7cdd739815a021d7ff2b36ef91f23
```

Acknowledgements

The authors of this document would like to acknowledge the helpful feedback and discussions from Benjamin Schwartz, Joseph Salowey, and Tara Whalen.

Authors' Addresses

Sofía Celi
Brave Software
Lisbon
Portugal
Email: cherenkov@riseup.net

Alex Davidson
NOVA LINCS, Universidade NOVA de Lisboa
Largo da Torre
Caparica
Portugal
Email: alex.davidson92@gmail.com

Steven Valdez
Google LLC
Email: svaldez@chromium.org

Christopher A. Wood
Cloudflare
101 Townsend St
San Francisco, CA 94107
United States of America
Email: caw@heapingbits.net