

Network Working Group  
Internet-Draft  
Intended status: Informational  
Expires: 7 October 2022

S. Celi  
Cloudflare  
A. Davidson  
Brave Software  
A. Faz-Hernandez  
Cloudflare  
S. Valdez  
Google LLC  
C. A. Wood  
Cloudflare  
5 April 2022

Privacy Pass Issuance Protocol  
draft-ietf-privacypass-protocol-04

## Abstract

This document specifies two variants of the the two-message issuance protocol for Privacy Pass tokens: one that produces tokens that are privately verifiable, and another that produces tokens that are publicly verifiable. The privately verifiable issuance protocol optionally supports public metadata during the issuance flow.

## Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of [BCP 78](#) and [BCP 79](#).

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 7 October 2022.

## Copyright Notice

Copyright (c) 2022 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to [BCP 78](#) and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Revised BSD License text as described in Section 4.e of the [Trust Legal Provisions](#) and are provided without warranty as described in the Revised BSD License.

## Table of Contents

<a href="#">1.</a>	Introduction . . . . .	<a href="#">2</a>
<a href="#">2.</a>	Terminology . . . . .	<a href="#">3</a>
<a href="#">3.</a>	Configuration . . . . .	<a href="#">3</a>
<a href="#">4.</a>	Token Challenge Requirements . . . . .	<a href="#">4</a>
<a href="#">5.</a>	Issuance Protocol for Privately Verifiable Tokens with Public Metadata . . . . .	<a href="#">4</a>
<a href="#">5.1.</a>	Client-to-Issuer Request . . . . .	<a href="#">5</a>
<a href="#">5.2.</a>	Issuer-to-Client Response . . . . .	<a href="#">6</a>
<a href="#">5.3.</a>	Finalization . . . . .	<a href="#">7</a>
<a href="#">5.4.</a>	Issuer Configuration . . . . .	<a href="#">8</a>
<a href="#">6.</a>	Issuance Protocol for Publicly Verifiable Tokens . . . . .	<a href="#">8</a>
<a href="#">6.1.</a>	Client-to-Issuer Request . . . . .	<a href="#">9</a>
<a href="#">6.2.</a>	Issuer-to-Client Response . . . . .	<a href="#">10</a>
<a href="#">6.3.</a>	Finalization . . . . .	<a href="#">11</a>
<a href="#">6.4.</a>	Issuer Configuration . . . . .	<a href="#">11</a>
<a href="#">7.</a>	Security considerations . . . . .	<a href="#">11</a>
<a href="#">8.</a>	IANA considerations . . . . .	<a href="#">11</a>
<a href="#">8.1.</a>	Token Type . . . . .	<a href="#">12</a>
<a href="#">8.2.</a>	Media Types . . . . .	<a href="#">12</a>
<a href="#">8.2.1.</a>	"message/token-request" media type . . . . .	<a href="#">12</a>
<a href="#">8.2.2.</a>	"message/token-response" media type . . . . .	<a href="#">13</a>
<a href="#">9.</a>	Normative References . . . . .	<a href="#">14</a>
<a href="#">Appendix A.</a>	Acknowledgements . . . . .	<a href="#">15</a>
<a href="#">Appendix B.</a>	Test Vectors . . . . .	<a href="#">15</a>
<a href="#">B.1.</a>	Issuance Protocol 1 - VOPRF(P-384, SHA-384) . . . . .	<a href="#">15</a>
<a href="#">B.2.</a>	Issuance Protocol 2 - Blind RSA, 4096 . . . . .	<a href="#">16</a>
	Authors' Addresses . . . . .	<a href="#">20</a>

## [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 [[I-D.ietf-privacypass-architecture](#)].

This document describes the issuance protocol for Privacy Pass. It specifies two variants: one that is privately verifiable based on the oblivious pseudorandom function from [[OPRF](#)], and one that is publicly verifiable based on the blind RSA signature scheme [[BLINDRSA](#)].

This document DOES NOT cover the architectural framework required for running and maintaining the Privacy Pass protocol in the Internet setting. In addition, it DOES NOT cover the choices that are necessary for ensuring that client privacy leaks do not occur. Both of these considerations are covered in [[I-D.ietf-privacypass-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.

The following terms are used throughout this document.

- \* Client: An entity that provides authorization tokens to services across the Internet, in return for authorization.
- \* Issuer: A service produces Privacy Pass tokens to clients.
- \* Private Key: The secret key used by the Issuer for issuing tokens.
- \* Public Key: The public key used by the Issuer for issuing and verifying tokens.

We assume that all protocol messages are encoded into raw byte format before being sent across the wire.

## [3.](#) Configuration

Issuers MUST provide one parameter for configuration:

1. Issuer Request URI: a token request URL for generating access tokens. For example, an Issuer URL might be `https://issuer.example.net/example-token-request`. This parameter uses resource media type "text/plain".

The Issuer parameters can be obtained from an Issuer via a directory object, which is a JSON object whose field names and values are raw values and URLs for the parameters.

Field Name	Value
issuer-request-uri	Issuer Request URI resource
	URL as a JSON string

Table 1

As an example, the Issuer's JSON directory could look like:

```
{
  "issuer-request-uri": "https://issuer.example.net/example-token-request"
}
```

Issuer directory resources have the media type "application/json" and are located at the well-known location `/.well-known/token-issuer-directory`.

4. Token Challenge Requirements

Clients receive challenges for tokens, as described in [\[AUTHSCHEME\]](#). The basic token issuance protocols described in this document can be interactive or non-interactive, and per-origin or cross-origin.

5. Issuance Protocol for Privately Verifiable Tokens with Public Metadata

The Privacy Pass issuance protocol is a two message protocol that

takes as input a challenge from the redemption protocol and produces a token, as shown in the figure below.



Issuers provide a Private and Public Key, denoted `skI` and `pkI`, respectively, used to produce tokens as input to the protocol. See [Section 5.4](#) for how this key pair is generated.

Clients provide the following as input to the issuance protocol:

- \* Issuer name, identifying the Issuer. This is typically a host name that can be used to construct HTTP requests to the Issuer.

- \* Issuer Public Key `pkI`, with a key identifier `key_id` computed as described in [Section 5.4](#).
- \* Challenge value `challenge`, an opaque byte string. For example, this might be provided by the redemption protocol in [\[HTTP-Authentication\]](#).

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`.

### [5.1](#). Client-to-Issuer Request

The Client first creates a context as follows:

```
client_context = SetupVOPRFClient(0x0004, pkI)
```

Here, `0x0004` is the two-octet identifier corresponding to the OPRF(P-384, SHA-384) ciphersuite in [\[OPRF\]](#). `SetupVOPRFClient` is defined in [\[OPRF\]](#), Section 3.2.

The Client then creates an issuance request message for a random value nonce using the input challenge and Issuer key identifier as follows:

```
nonce = random(32)
context = SHA256(challenge)
token_input = concat(0x0001, nonce, context, key_id)
blind, blinded_element = client_context.Blind(token_input)
```

The Blind function is defined in [\[OPRF\]](#), Section 3.3.2. If the Blind function fails, the Client aborts the protocol. Otherwise, the Client then creates a TokenRequest structured as follows:

```
struct {
    uint16_t token_type = 0x0001;
    uint8_t 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.
- \* "token\_key\_id" is the least significant byte of the key\_id.

- \* "blinded\_msg" is the Ne-octet blinded message defined above, computed as SerializeElement(blinded\_element). Ne is as defined in [\[OPRF\]](#), Section 4.

The values token\_input and blinded\_element are stored locally and used later as described in [Section 5.3](#). The Client then generates an HTTP POST request to send to the Issuer, with the TokenRequest as the body. The media type for this request is "message/token-request". An example request is shown below.

```
:method = POST
:scheme = https
:authority = issuer.example.net
:path = /example-token-request
accept = message/token-response
```

```
cache-control = no-cache, no-store
content-type = message/token-request
content-length = <Length of TokenRequest>
```

<Bytes containing the TokenRequest>

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

- \* The TokenRequest contains a supported token\_type.
- \* The TokenRequest.token\_key\_id corresponds to a key ID of a Public Key owned by the issuer.
- \* The TokenRequest.blinded\_request is of the correct size.

If any of these conditions is not met, the Issuer MUST return an HTTP 400 error to the client.

## [5.2.](#) Issuer-to-Client Response

Upon receipt of a TokenRequest, the Issuer tries to deserialize TokenRequest.blinded\_msg using DeserializeElement from Section 2.1 of [\[OPRF\]](#), yielding blinded\_element. If this fails, the Issuer MUST return an HTTP 400 error to the client. Otherwise, if the Issuer is willing to produce a token token to the Client, the Issuer completes the issuance flow by computing a blinded response as follows:

```
server_context = SetupVOPRFServer(0x0004, skI, pkI)
evaluate_element, proof = server_context.Evaluate(skI, blinded_element)
```

SetupVOPRFServer is in [\[OPRF\]](#), Section 3.2 and Evaluate is defined in [\[OPRF\]](#), Section 3.3.2. The Issuer then creates a TokenResponse structured as follows:

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

The structure fields are defined as follows:

- \* "evaluate\_msg" is the Ne-octet evaluated messaged, 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]))`, where Ns is as defined in [\[OPRF\]](#), Section 4.

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

```
:status = 200
content-type = message/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 body values `TokenResponse.evaluate_response` 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,
```

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
uint8_t nonce[32];
uint8_t challenge_digest[32];
uint8_t token_key_id[32];
uint8_t authenticator[Nk];
} Token;
```

Otherwise, the Client aborts the protocol.

#### [5.4.](#) Issuer Configuration

Issuers are configured with Private and Public Key pairs, each denoted skI and pkI, respectively, used to produce tokens. Each key pair MUST be generated as follows:

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

The key identifier for this specific key pair, denoted key\_id, is computed as follows:

```
key_id = SHA256(0x0001 || SerializeElement(pkI))
```

#### [6.](#) Issuance Protocol for Publicly Verifiable Tokens

This section describes a variant of the issuance protocol in [Section 5](#) for producing publicly verifiable tokens. It differs from the previous variant in two important ways:

1. The output tokens are publicly verifiable by anyone with the Issuer public key; and
2. The issuance protocol does not admit public or private metadata to bind additional context to tokens.

Otherwise, this variant is nearly identical. In particular, Issuers provide a Private and Public Key, denoted skI and pkI, respectively, used to produce tokens as input to the protocol. See [Section 6.4](#) for how this key pair is generated.

Clients provide the following as input to the issuance protocol:

- \* Issuer name, identifying the Issuer. This is typically a host name that can be used to construct HTTP requests to the Issuer.
- \* Issuer Public Key pkI, with a key identifier key\_id computed as described in [Section 6.4](#).

- \* Challenge value challenge, an opaque byte string. For example, this might be provided by the redemption protocol in [\[HTTP-Authentication\]](#).

Given this configuration and these inputs, the two messages exchanged in this protocol are described below.

### [6.1.](#) Client-to-Issuer Request

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

```
nonce = random(32)
context = SHA256(challenge)
token_input = concat(0x0002, nonce, context, key_id)
blinded_msg, blind_inv = rsabssa_blind(pkI, token_input)
```

The rsabssa\_blind function is defined in [\[BLINDRSA\]](#), Section 5.1.1.. The Client then creates a TokenRequest structured as follows:

```
struct {
    uint16_t token_type = 0x0002;
    uint8_t 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.
- \* "token\_key\_id" is the least significant byte of the key\_id.
- \* "blinded\_msg" is the Nk-octet request defined above.

The Client then generates an HTTP POST request to send to the Issuer, with the TokenRequest as the body. The media type for this request is "message/token-request". An example request is shown below, where Nk = 512.

```
:method = POST
:scheme = https
:authority = issuer.example.net
:path = /example-token-request
accept = message/token-response
cache-control = no-cache, no-store
content-type = message/token-request
content-length = <Length of TokenRequest>
```

<Bytes containing the TokenRequest>

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

- \* The TokenRequest contains a supported token\_type.
- \* The TokenRequest.token\_key\_id corresponds to a key ID of a Public Key owned by the issuer.
- \* The TokenRequest.blinded\_msg is of the correct size.

If any of these conditions is not met, the Issuer MUST return an HTTP 400 error to the Client, which will forward the error to the client.

## [6.2.](#) Issuer-to-Client Response

If the Issuer is willing to produce a token token to the Client, the Issuer completes the issuance flow by computing a blinded response as follows:

```
blind_sig = rsabssa_blind_sign(skI, TokenRequest.blinded_rmsg)
```

This is encoded and transmitted to the client in the following TokenResponse structure:

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

The `rsabssa_blind_sign` function is defined in [[BLINDRSA](#)], Section 5.1.2.. The Issuer generates an HTTP response with status code 200 whose body consists of `TokenResponse`, with the content type set as "message/token-response".

```
:status = 200
content-type = message/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 body as follows:

```
authenticator = rsabssa_finalize(pkI, nonce, blind_sig, blind_inv)
```

The `rsabssa_finalize` function is defined in [[BLINDRSA](#)], Section 5.1.3.. If this succeeds, the Client then constructs a Token as described in [[HTTP-Authentication](#)] as follows:

```
struct {
    uint16_t token_type = 0x0002;
    uint8_t nonce[32];
    uint8_t challenge_digest[32];
    uint8_t token_key_id[32];
    uint8_t authenticator[Nk];
} Token;
```

Otherwise, the Client aborts the protocol.

### [6.4.](#) Issuer Configuration

Issuers are configured with Private and Public Key pairs, each denoted `skI` and `pkI`, respectively, used to produce tokens. Each key pair MUST be generated as as a valid 4096-bit RSA private key

according to [TODO]. The key identifier for a keypair (skI, pkI), denoted key\_id, is computed as SHA256(encoded\_key), where encoded\_key is a DER-encoded SubjectPublicKeyInfo object carrying pkI.

## 7. Security considerations

This document outlines how to instantiate the Issuance protocol based on the VOPRF defined in [OPRF] and blind RSA protocol defined in [BLINDRSA]. All security considerations described in the VOPRF document 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 deferred to the Architecture document [I-D.ietf-privacypass-architecture].

## 8. IANA considerations

Celi, et al. Expires 7 October 2022 [Page 11]

---

Internet-Draft Privacy Pass Issuance April 2022

### 8.1. Token Type

This document updates the "Token Type" Registry with the following values.

Value	Name	Publicly Verifiable	Public Metadata	Private Metadata	Nk	Reference
0x0001	VOPRF(P-384, SHA-384)	N	N	N	48	<a href="#">Section 5</a>
0x0002	Blind RSA, 4096	Y	N	N	512	<a href="#">Section 6</a>

Table 2: Token Types

### 8.2. Media Types

This specification defines the following protocol messages, along with their corresponding media types:

\* TokenRequest: "message/token-request"

\* TokenResponse: "message/token-response"

The definition for each media type is in the following subsections.

#### [8.2.1](#). "message/token-request" media type

Type name: message

Subtype name: token-request

Required parameters: N/A

Optional parameters: None

Encoding considerations: only "8bit" or "binary" is permitted

Security considerations: see [Section 7](#)

Interoperability considerations: N/A

Published specification: this specification

Applications that use this media type: N/A

Celi, et al.

Expires 7 October 2022

[Page 12]

---

Internet-Draft

Privacy Pass Issuance

April 2022

Fragment identifier considerations: N/A

Additional information: Magic number(s): N/A

Deprecated alias names for this type: N/A

File extension(s): N/A

Macintosh file type code(s): N/A

Person and email address to contact for further information: see Authors' Addresses section

Intended usage: COMMON

Restrictions on usage: N/A

Author: see Authors' Addresses section

Change controller: IESG

[8.2.2.](#) "message/token-response" media type

Type name: message

Subtype name: access-token-response

Required parameters: N/A

Optional parameters: None

Encoding considerations: only "8bit" or "binary" is permitted

Security considerations: see [Section 7](#)

Interoperability considerations: N/A

Published specification: this specification

Applications that use this media type: N/A

Fragment identifier considerations: N/A

Additional information: Magic number(s): N/A

Deprecated alias names for this type: N/A

File extension(s): N/A

Celi, et al.

Expires 7 October 2022

[Page 13]

---

Internet-Draft

Privacy Pass Issuance

April 2022

Macintosh file type code(s): N/A

Person and email address to contact for further information: see Authors' Addresses section

Intended usage: COMMON

Restrictions on usage: N/A

Author: see Authors' Addresses section

## 9. Normative References

### [AUTHSCHEME]

Pauly, T., Valdez, S., and C. A. Wood, "The Privacy Pass HTTP Authentication Scheme", Work in Progress, Internet-Draft, [draft-pauly-privacypass-auth-scheme-00](https://datatracker.ietf.org/doc/html/draft-pauly-privacypass-auth-scheme-00), 31 January 2022, <<https://datatracker.ietf.org/doc/html/draft-pauly-privacypass-auth-scheme-00>>.

### [BLINDRSA]

Denis, F., Jacobs, F., and C. A. Wood, "RSA Blind Signatures", Work in Progress, Internet-Draft, [draft-irtf-cfrg-rsa-blind-signatures-03](https://datatracker.ietf.org/doc/html/draft-irtf-cfrg-rsa-blind-signatures-03), 2 February 2022, <<https://datatracker.ietf.org/doc/html/draft-irtf-cfrg-rsa-blind-signatures-03>>.

### [HTTP-Authentication]

"The Privacy Pass HTTP Authentication Scheme", n.d., <<https://datatracker.ietf.org/doc/html/draft-pauly-privacypass-auth-scheme-00>>.

### [I-D.ietf-privacypass-architecture]

Davidson, A., Iyengar, J., and C. A. Wood, "Privacy Pass Architectural Framework", Work in Progress, Internet-Draft, [draft-ietf-privacypass-architecture-03](https://datatracker.ietf.org/doc/html/draft-ietf-privacypass-architecture-03), 7 March 2022, <<https://datatracker.ietf.org/doc/html/draft-ietf-privacypass-architecture-03>>.

### [OPRF]

Davidson, A., Faz-Hernandez, A., Sullivan, N., and C. A. Wood, "Oblivious Pseudorandom Functions (OPRFs) using Prime-Order Groups", Work in Progress, Internet-Draft, [draft-irtf-cfrg-voprpf-09](https://datatracker.ietf.org/doc/html/draft-irtf-cfrg-voprpf-09), 8 February 2022, <<https://datatracker.ietf.org/doc/html/draft-irtf-cfrg-voprpf-09>>.

### [RFC2119]

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

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

## [Appendix A](#). Acknowledgements

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

## [Appendix B](#). Test Vectors

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

### [B.1](#). Issuance Protocol 1 – VOPRF(P-384, SHA-384)

The test vector below lists the following values:

- \* skS: The encoded OPRF private key, serialized using `SerializeScalar` from Section 2.1 of [\[OPRF\]](#) and represented as a hexadecimal string.
- \* pkS: The encoded OPRF public key, serialized using `SerializeElement` from Section 2.1 of [\[OPRF\]](#) and represented as a hexadecimal string.
- \* challenge: A random challenge, 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` from Section 2.1 of [\[OPRF\]](#) and represented as a hexadecimal string.
- \* token\_request: The `TokenRequest` message constructed according to [Section 5.1](#), represented as a hexadecimal string.

- \* `token_request`: 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.

```
skS: 0177781aeced893dcccdf80713d318a801e2a0498240fdcf650304bbbfd0f8d3b5c0
cf6cfee457aaa983ec02ff283b7a9
pkS: 022c63f79ac59c0ba3d204245f676a2133bd6120c90d67afa05cd6f8614294b7366
c252c6458300551b79a4911c2590a36
challenge:
a5d46383359ef34e3c4a7b8d1b3165778bffc9b70c9e6a60dd14143e4c9c9fbd
nonce: 5d4799f8338ddc50a6685f83b8ecd264b2f157015229d12b3384c0f199efe7b8
blind: 0322fec505230992256296063d989b59cc03e83184eb6187076d264137622d202
48e4e525bdc007b80d1560e0a6f49d9
token_request: 00011a02861fd50d14be873611cfff0131d2c872c79d0260c6763498a2
a3f14ca926009c0f247653406e1d52b68d61b7ed2bac9ea
token_response: 038e3625b6a769668a99680e46cf9479f5dc1e86d57164ab3b4a569d
dfc486bf1485d4916a5194fdc0518d3e8444968421ba36e8144aa7902705ff0f3cf40586
3d69451a2a7ba210cc45760c2f1a6045134d877b39e8bcbbf920e5de4a3372557deb211
765cd969976860bc039f9082d6a3e03f8e891246240173d2cf3d69a4613b0f8415979029
22e74c7a1f2e4639e4
token: 00015d4799f8338ddc50a6685f83b8ecd264b2f157015229d12b3384c0f199efe
7b8742cdfb0ed756ea680868ef109a280a393e001d2fa56b1be46ecb31fa25e76731a5b1
d698ea7ab843b8e8a71ed9b2fffa70457a43a8fc687939424b29a7554b40fde130ab7a82
2715909cb73f99a45b640ca1c85180ba9ca1a40bab8b664406a34bcb63b5e2e5c455cea
00001a968f7
```

## [B.2](#). Issuance Protocol 2 - Blind RSA, 4096

The test vector below lists the following values:

- \* `skS`: The PEM-encoded PKCS#8 RSA private key used for signing tokens, represented as a hexadecimal string.
- \* `pkS`: The DER-encoded `SubjectPublicKeyInfo` object carrying the public key corresponding to `skS`, as described in [Section 6.4](#), represented as a hexadecimal string.
- \* `challenge`: A random challenge, 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.

Internet-Draft

Privacy Pass Issuance

April 2022

- \* salt: The randomly generated 48-byte salt used when encoding the blinded token request message, represented as a hexadecimal string.
- \* token\_request: The TokenRequest message constructed according to [Section 6.1](#), represented as a hexadecimal string.
- \* token\_request: 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.

skS: 2d2d2d2d2d424547494e2050524956415445204b45592d2d2d2d0a4d49494a517  
749424144414e42676b71686b69473977304241514546414153434353307767676b70416  
74541416f4943415144584d4d364c5073637a6130696b0a39596c315a4e683868324a796  
d504962704c383035354d52516f6463446c4c4672764d694c57666f59535241506b4e744  
1546272324162654e334d454a2f67550a392f424958717a704d4256484852546c627a676  
b364c7866766b54505a2b4a427832517177364f5555714b442f34426f5464765761536d6  
a63644165555037760a4836376e396a383836307463367375546e67616c574d4b6d764d2  
f5a6237644262543763345647386577706c6776444f6d616641353956354f78505545704  
9510a586c454f4771414f43436c316f54686d33363836436c48413352374c5a4d78567a4  
742386f594537583548396a70793532786a2f3242724a6861625033567a430a6f4c696c3  
54f6e36487158785363466d4956573742787956495979756d75537659544e52757652777  
3566c3775595446366f4d6d2b59722f5a506372594c7a510a4e666135634f747533532b6  
649594456716f6f58375541415671382f716c4945747a794a744f7261616756393039327  
0324f474c4f6d306a52384543666963520a3868363332572f76554d7739724c4c317a4d4  
e7554424f4b74316c546c307265644a677668626b66515a5a55303541336a69366c71754  
d2f47307250553030470a3369385253732f64694e625843505453746b616f45537350345  
946496d526269756c786a544e2b626c5859744868494261556774306e2b566b544a77554  
86e380a5367447534664e547142776567517265494e427732446b6e63576e676b5644416  
867577635383669727351644c345843723376765856647a51375134586978730a465a6b6  
d7763676d665142444f3469363078536c336337316954595362783359326e74584b4e6f5  
a4c31537a424f595051496a6c734749454250677157546e5a0a64655a7852464f6c4d384  
679794b6d30746471586271594b57716b663777494441514142416f4943414364314d707  
044773645424467763558654168777252710a32726c717042492f6a6a50304e6e7057755  
a302b6e787a53624a4361784d2f4f61436844676e654e586e573259655144524e7242505  
84d5331344e646f4e554e0a56516c36497166445567637169636741696e7542622f4a687  
a6c4d74466d5350466d2b667650726a4d2b6c4831544f384863354253633274414a52574  
36468770a794a7663446349656d74646378437877754b6746485251704a5071356368526

56431535077766f50494c7831686959356c2f5175423478717a764149483873530a34673  
949705a2f766169443558573541335847734a4f2b696a78717250706756655236474e4f5  
33763515551716a44446967665a626a5864354a322b734d79460a4c435a6e514d6771577  
03731756437366a4f4a445571563537454154534e6b56472f52633279537a554b4d6e354  
c335a314a796d6d31694f584a5136535744760a6f5375426b435652436645635a746b546  
f436c76646f5456666b63414a394e51343839686b3050754e466f593675315671614c5a4  
a7764657a3931764f5966300a4b676f665a4a6e774976547733754b786f7559685033334  
43644574a4f31763265487658767a4c744f4a54477054446f726f4c7a2f475459316b682

Celi, et al.

Expires 7 October 2022

[Page 17]

Internet-Draft

Privacy Pass Issuance

April 2022

f484d50510a416c4f333758772f526b527752684b4d6f39545555347766642b2b5348325  
4346e6730584875355254764d5339496b4266724455337a75557675434b7270356b0a377  
942364473763433517932364f6367367151584a563650676b6c5770696164445778326f4  
73935746578436f3346563434764a6d49327a706d5748514257630a48395331336b6d7a3  
06873506274576a537535494c443061694f5630764d4c61353841685a767a32774562763  
750546a494d48364d77446736326d6450744d6b0a534969517548644a38326a69316e757  
8646f5678416f4942415144773431456757476e437644496239727a73704b6e61486a654  
3574650385737664a6b446c4c0a61786c724a6f574e614d654c33306b646873576569765  
7616730436f457656556631543455374936735a33705054675653685079374b6f4c645a4  
86f757a5a390a4d774679716d694652726e574135467138544b756f4354647a3836542f5  
37859753330625a476c6d6a4c432f6664453279556141486f6f5177375a2f34646b6d0a4  
1693031696a36484d654e39486e5a425737656c436c4173443242436d4b5279655554793  
7754b472b6b3732704630336e45694f7a42615565354d6c7a7436620a3867586b63715a5  
735495478454b41726b6d486a77454379765178346a7867324573326a7653654e3041554  
b624a63534a736e7a35434a65667a697561676f650a36482b3947676f62386e496a78546  
34e784c7a58316a676e5236772f41302f6e657837714e6b57357773485a46644c58416f4  
9424151446b734d6e75503452460a7558474a6e34316559696c43716b694e4e6b4a53753  
9637456556b7772364570744230506d34556e3770535238684866767a5a43782f6b65766  
b33553336546e0a67556c72416a4b72474a6a77437231457a7252726e417153466f61395  
04851395a69697264626336726f474932576f63795a585859664b784a56646a4f754c440  
a793947564d72575133665a3571696d7a6a394b4163304852455765784b6875514259465  
554542f44323169584b4b65497568647834796b795435323857714b310a66786f4c72584  
e6d5936726b77787778763334556779685a54564a75454879506e51422f54743242377a2  
b342f763033665a46347361635646337738514a306c0a692b466879555a34326b63674a3  
52b654c6d5558726b6d5a5959314a534332417a722f336449516a4b554655515935326a3  
65775667338392f576a72516669470a31732b6c51385a7a6d4a4370416f49424151432b3  
84c43686e764e574e4b37546b365431507943546b466758726351457950374a657453766  
6316c4b6f654a430a304d6337692b58387a5a4e6674473478353941636163716c43376c6  
96a5a55394351564f6d415159652f754d4679524371524c62453271426d79694f70357a7  
00a3538487562693261513034564e554f44767644555256346468364148556e52706f534  
f49356b59723079646137746f7070376a4662565165324b4c56535a742b0a746f4448384  
a6c7a2f5374345774427033465a4538354747573748586a70746f756f6855344c777a464  
7492f4c6d37486935783733357038716a385a6366642f0a384f756632626e6354382f674

938676b35633034307451794b483177534d4e4e6d5a496c54535943635653725369346b4  
5567677684955354d726e75507647380a6256556b48584d694b7377316d63777739704d4  
6373634716f6d464337586f66594d30664d6a6c4a416f494241465079565632676354534  
b2b784e7976786b4c0a58577638522f2b57454569416256395674445572387a503079736  
f6b34333869412b57432f32367271515a676b36446d61486d673073367356622f7a495a6  
84f0a7769307a4e41446a41375751705179314f696153333522b594b56333435656c345  
35454386a433543736a79536e306559504b71396679376636614e343770570a304267664  
4346d37584b454d4c66664a744d2b437a6e56536f41504c433449677257646e5a4141376  
c306d57416c52576832645275664a3377703751763943770a2b315659446178785235334  
e2b327930686e4b696d4b613745696970555952567834566f444a6c6d2f5a525a5769545  
335796253375279514f563645334e71560a2f592f6647366563446a33674732497a50674  
c4e664f3651646b556d7679364e4155386c645638754962707045446749497042684f684  
a394865486a664343490a75326b4367674542414d68686e6f69312b78454a365339636e5  
a327977527737433057544962662b54557230436e374f344a4d76637843544b484d62773  
76a680a574c6247392f314732595966354c5673326b572f34472b2f446d586b6d4b68715

Celi, et al.

Expires 7 October 2022

[Page 18]

Internet-Draft

Privacy Pass Issuance

April 2022

4746d6f324e50463666504b73694d64477877354149677039557a50543168550a2f6c777  
26a75474d322f2b77434b70316d3645374e4d4d5659684b5841534f673473652b3676586  
455356a56436f634343576162657a6c5835513262796c51480a2b2b624f6d4661504d535  
5683761616d6657357573614553627366612f45506932446d3545574f456339567577525  
671526143527552534632507043627279410a6b376e6b6b6746474b5a523777353053465  
5366f7a776667627a2b7a33637256315535766d3076346c63794b6d524b6c4b575a51554  
9624b782f5070583737640a395057536e3569594343704c432f316245372f566f4c70467  
7757631656a773d0a2d2d2d2d2d454e442050524956415445204b45592d2d2d2d2d0a  
pkS: 30820252303d06092a864886f70d01010a3030a00d300b060960864801650304020  
2a11a301806092a864886f70d010108300b0609608648016503040202a20302013003820  
20f003082020a0282020100d730ce8b3ec7336b48a4f5897564d87c87627298f21ba4bf3  
4e7931142875c0e52c5aef3222d67e86124403e436d0136ebd806de37730427f814f7f04  
85eace93015471d14e56f3824e8bc5fbe44cf67e241c7642ac3a39452a283ff80684ddbd  
66929a371d01e50feef1faee7f63f3ceb4b5ceacb939e06a558c2a6bccfd96fb7416d3ed  
ce151bc7b0a6582f0ce99a7c0e7d5793b13d41292105e510e1aa00e082975a13866dfaf3  
a0a51c0dd1ecb64cc55cc607ca1813b5f91fd8e9cb9db18ffd81ac985a6cfdd5cc2a0b8a  
5e4e9fa1ea5f149c1662155bb071c95218cae9ae4af613351baf470b1597bb984c5ea832  
6f98aff64f72b60bcd035f6b970eb6edd2f9f2180d5aa8a17ed400056af3faa5204b73c8  
9b4eada6a057dd3dda9d8e18b3a6d2347c1027e2711f21eb7d96fef50cc3dacb2f5ccc36  
e4c138ab75953974ade74982f85b91f419654d390378e2ea5aae33f1b4acf534d06de2f1  
14acfdd88d6d708f4d2b646a8112b0fe181489916e2ba5c634cdf9b95762d1e120169482  
dd27f959132705079fc4a00eee1f353a81c1e810ade20d070d839277169e09150c08605a  
fe7cea2aec41d2f85c2af7bef5d577343b4385e2c6c159926c1c8267d00433b88bad314a  
5ddcef58936126f1dd8da7b5728da192f54b304e60f4088e5b0620404f82a5939d975e67  
14453a533c172c8a9b4b5da976ea60a5aa91fef0203010001  
challenge:

83ce743dcdadd5fc4aeb0357977bb8426635c390a15b88947f0b1c62e4a87c22  
nonce: 7e0da97bfdc4365a5f40e69262f78b81bcd2f92daf885358d9831874e3dd9d22  
blind: cd6d03e332386d0166eb76b8e78522510e5cbdcf49aaac83191ea948a7719e914  
0ccb6701f7301b7d445ede7adbc5e582b35edd9ac45bc4b8f794e150b2e3e407b7b7624b  
6f90b33845bc255174cee0c570aa781c203dce8563afe9f48e2b49c773bba1031987fb48  
d981d131876f53e264ec0609a3ea628cf2042005ed3071aeb6657472c7e7df947915b8cd  
333e3f5078e456e65e5edef8f892c4f21d25a18dcd80628ed6c7d55b0b9433bc67760be0  
8a4eacbdb16a4be4c5b8cab26b478fa6a36ea3c3dd1fffb420bf69feef52aab4892c9e60a  
df18347b4e8256b5a0e8cbf55fc97ac62af2e7349ba98ca7462cb6a41d70b0217814a06e  
1b257289c3b345be652b87d5820b06a80500880b40b8772140bf431f11497114b20fee7e  
5ffc1af5cf874cc293a0c8df65d52814bcd55ae6d3701f73d140ca82c6528627129ea389  
f3cbd6058f4f80b7df3818f36dd3489259b6b95df4511930ff02b5cbe643fea44306e7c4  
e3d9b02f1b0559aa238b8882a6e8791bfbfd366ef4fe433fd42e5c5db208c9fcebc74def  
11663ce5f793c7013116995b3fef392a8633b08179a9c8309fb69359fd8486a7a8febb42  
4d0726c2516b11e8b19a55fa54e9be606de6811059976473de8f9adb25af7e2862932bed  
c7764b4dc50bfc9d724a4aba356a7677b5aceef21876e56b4f1b65adef0fbf8bf1636815  
be01b372727e79aa6c47f41  
salt: d13a47fa6466a37203e51ac34f7319831b3f04202ff74c98ab18e78088b7ac3014  
06189783503227153871405c6a1da0  
token\_request: 0002013a370077e8259098e741dcaac8184838b7c995cd82966419064  
90205bf28e6745396dd9ec1761c4676e9fec3272588194c48bc60fc77c3fff19219a6b65  
96523044d07d9d9e4dd88f2db9d9c369597363a12a65d244d69a1b743b365f8f5bdfb8d6

52f2d2e249f417e9fd7da7db2626d6499d7d3856d8f9277385dfd776ffa4ffa74d7a7b17  
01d87f40e525bb258a7f5b4d6c134b3b242ef46d93b32f106c84c396b1fc2772796b2473  
64c48c364537708f3a8a87fb870a299ac08107a5dd3f467733d76c2359e346ee3ebcff68  
c3c7e10f0f01a2b9bbdb26ffea14f81f036a71c47725b5c9f81e0320b85abfd77d5eb1c  
ccfdd8eb0cf2735ea297e2a07c82dfee9b0a4d21baa81a2cfb2a72983107555035386c33  
973d48f04257dd8298116d2c93298810cb6b82ad033f5b16f9f7a65d8f74b7bdcae000db  
1411b40f46cb373cb69c8ab58552f98904b78229b63838ab40e833fab4ec47acf00a00db  
3290c9c74ed982c64ddbcae16fd73976ffe7da7b3b1a8e0190e95b7ce7900295f3c8c94f  
2d3d85cd1825fe27073aad63fa4c530907402dc4e3a748edb300f05ce7ac5b8c1d9aeb2f  
d6002b05dee582aeafa4f503f13bae1a51be9420e3cdcb685169bdc5ae2ebca7713ec16  
666b55b097e56e5719d1b0324ecaca613af76b9f367775a90dba5e7fdd21a8da73bd80b1  
31e6531117ef709ad8c7b2b3182255235acea  
token\_response: 061780e09bc9b851fe81e7022ee2d55b043198bcb1aa33f761d213a9  
8d831abae5417d30904a7c9d7ec531278cd9655c4b7d72f3a4e66e26565b73e5f1b9271  
b541f28556543d2473c363e104a11c6ba1a1d8f99b32d3cd8f74ae07b465afdaabd21977  
d6f114ea9484cd592f461e5dc4a97b86fe1463b1c69171cf734f49c240760dc2555d7cc8  
9d7f882d3e1b99388bc3b561d418a7fc5770ccc66e3dd41f4a74e8267492f48e8b6aabce  
592c8dc83826b1f4528f2497902a0ce4baacd4216623274057592e77091102452de115bb  
d0c98ee22b14fe30fb371277ab17bc948b62b8adb56b67e44dce74860647718c8f4bd10e

[illegible]

Celi, et al.

Internet-Draft

Sofía Celi  
Cloudflare  
Lisbon  
Portugal  
Email: [sceli@cloudflare.com](mailto:sceli@cloudflare.com)

Armando Faz-Hernandez  
Cloudflare  
101 Townsend St  
San Francisco,  
United States of America  
Email: armfazh@cloudflare.com

Steven Valdez  
Google LLC  
Email: svaldez@chromium.org

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