Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: 23 April 2026

C. Yun
C. A. Wood
Apple, Inc.

20 October 2025

# Privacy Pass Issuance Protocol for Anonymous Rate-Limited Credentials
draft-yun-privacypass-arc-02

## Abstract

This document specifies the issuance and redemption protocols for tokens based on the Anonymous Rate-Limited Credential (ARC) protocol.

## About This Document

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

The latest revision of this draft can be found at https://chris-wood.github.io/draft-arc/draft-yun-privacypass-arc.html. Status information for this document may be found at https://datatracker.ietf.org/doc/draft-yun-privacypass-arc/.

Discussion of this document takes place on the PRIVACYPASS Privacy Pass mailing list (mailto:privacy-pass@ietf.org), which is archived at https://mailarchive.ietf.org/arch/browse/privacy-pass. Subscribe at https://www.ietf.org/mailman/listinfo/privacy-pass/.

Source for this draft and an issue tracker can be found at https://github.com/chris-wood/draft-arc.

## Status of This Memo

Table of Contents

## 1. Introduction

[ARCHITECTURE] describes the Privacy Pass architecture, and [ISSUANCE] and [AUTHSCHEME] describe the issuance and redemption protocols for basic Privacy Pass tokens, i.e., those computed using blind RSA signatures (as specified in [BLIND-RSA]) or verifiable oblivious pseudorandom functions (as specified in [VOPRFS]). These protocols are widely deployed in practice for a variety of applications, including anonymous authentication for protocols such as Oblivious HTTP [OHTTP] and the Distributed Aggregation Protocol [DAP]. While effective, these variants of Privacy Pass tokens are limited in that each token can only be spent once. These are often calle "one-time-use" tokens. This means that applications which wish to limit access to a given user, e.g., for the purposes of throttling or rate limiting them, must issue one token for each redemption.

The Anonymous Rate-Limited Credential (ARC) protocol, as specified in [ARC], offers a more scalable approach to rate limiting. In particular, ARC credentials can be issued once and then presented (or redeemed) up to some fixed-amount of time for distinct, per-origin presentation contexts. This means that a Client will only be able to present a limited number of tokens associated with a given context.

This document specifies the issuance and redemption protocols for ARC. Section 2 describes motivation for this new type of token, Section 4 presents an overview of the protocols, and the remainder of the document specifies the protocols themselves.

## 2. Motivation

To demonstrate how ARC is useful, consider the case where a client wishes to keep its IP address private while accessing a service. The client can hide its IP address using a proxy service or a VPN. However, doing so severely limits the client’s ability to access services and content, since servers might not be able to enforce their policies without a stable and unique client identifier.

With one-time-use tokens, the server can verify that each client access meets a particular bar for attestation, i.e., the bar that is enforced during issuance, but cannot be used by the server to rate limit a specific client. This is because there is no mechanism in the issuance protocol to link repeated Client token requests in order to apply rate-limiting.

There are several use cases for rate-limiting anonymous clients that are common on the Internet. These routinely use client IP address tracking, among other characteristics, to implement rate-limiting.

One example of this use case is rate-limiting website accesses to a client to help prevent abusive behavior. Operations that are sensitive to abuse, such as account creation on a website or logging into an account, often employ rate-limiting as a defense-in-depth strategy. Additional verification can be required by these pages when a client exceeds a set rate-limit.

## 3. 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 from Section 3 of [TLS13]. Moreover, all constants are in network byte order.

Encoding an integer to a sequence of bytes in network byte order is described using the function encode(n, v), where n is the number of bytes and v is the integer value. The function len(x) returns the length in bytes of the byte string x.

## 4. Protocol Overview

The issuance and redemption protocols defined in this document are built on the Anonymous Rate-Limited Credential (ARC) protocol. In contrast to the core Privacy Pass protocols which are one-time-use anonymous credentials, ARC allows clients to turn a single credential output from an issuance protocol into a fixed number of unlinkable tokens, each of which are bound to some agreed-upon public presentation context.

With ARC, Clients receive TokenChallenge inputs from the redemption protocol ([AUTHSCHEME], Section 2.1). If they have a valid credential for the designated Issuer, Clients can use the TokenChallenge to produce a single token for presentation. Otherwise, Clients invoke the issuance protocol to obtain a credential. This interaction is shown below.

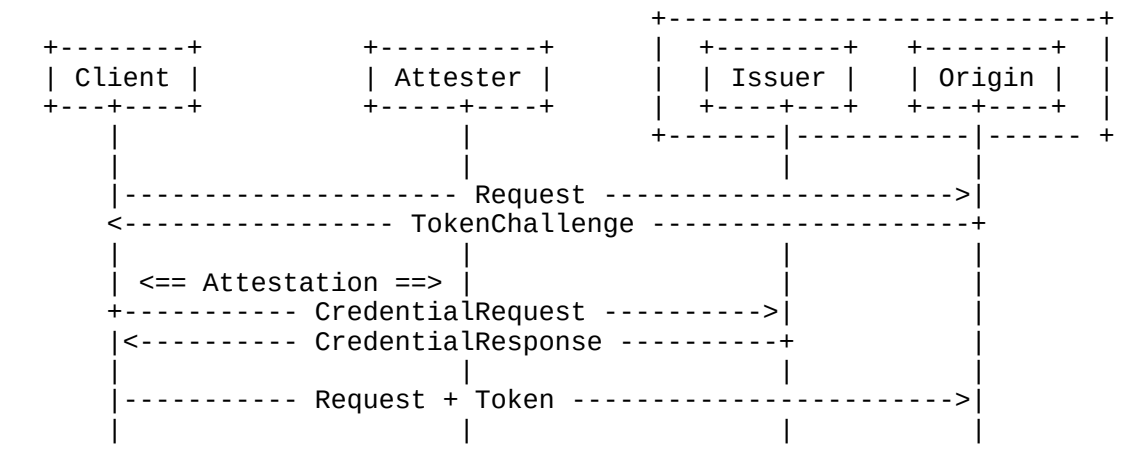


Figure 1: Issuance and Redemption Overview

Similar to the core Privacy Pass protocols, the TokenChallenge can be interactive or non-interactive, and per-origin or cross-origin.

ARC is only compatible with deployment models where the Issuer and Origin are operated by the same entity (see Section 4 of [ARCHITECTURE]), as tokens produced from a credential are not publicly verifiable. 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.

The issuance and redemption protocols in this document are built on [ARC].

## 5. Configuration

ARC Issuers are configured with key material used for issuance and token verification. Concretely, Issuers run the SetupServer function from [ARC] to produce a private and public key, denoted skI and pkI, respectively.

```
skI, pkI = SetupServer()
```

The Issuer Public Key ID, denoted issuer_key_id, is computed as the SHA-256 hash of the Issuer Public Key, i.e., issuer_key_id = SHA-256(pkI_serialized), where pkI_serialized is the serialized version of pkI as described in Section 4.1 of [ARC].

## 6. Token Challenge Requirements

The ARC protocol uses a modified TokenChallenge structure from the one specified in [AUTHSCHEME]. In particular, the updated TokenChallenge structure is as follows:

```
struct {
    uint16_t token_type = 0xE5AC; /* Type ARC(P-256) */
    opaque issuer_name<1..2^16-1>;
    opaque redemption_context<0..32>;
    opaque origin_info<0..2^16-1>;
    opaque credential_context<0..32>;
} TokenChallenge;
```

With the exception of credential_context, all fields are exactly as specified in Section 2.1.1 of [AUTHSCHEME]. The credential_context field is defined as follows:

* "credential_context" is a field that is either 0 or 32 bytes, prefixed with a single octet indicating the length (either 0 or 32). If value is non-empty, it is a 32-byte value generated by the origin that allows the origin to require that clients fetch credentials bound to a specific context. Challenges with credential_context values of invalid lengths MUST be ignored.

Similar to the redemption_context field, the credential_context is used to bind information to the credential. This might be useful, for example, to enforce some expiration on the credential. Origins might do this by constructing credential_context as F(current time window), where F is a pseudorandom function. Semantically, this is equivalent to the Origin asking the Client for a token from a credential that is bound to "current time window."

OPEN ISSUE: give more guidance about how to construct credential_context and redemption_context depending on the application’s needs.

In addition to this updated TokenChallenge, the HTTP authentication challenge also SHOULD contain the following additional attribute:

* "rate-limit", which contains a JSON number indicating the presentation limit to use for ARC.

Implementation-specific steps: the client should store the Origin-provided input tokenChallenge so that when they receive a new tokenChallenge value, they can check if it has changed and which fields are different. This will inform the client's behavior - for example, if credential_context is being used to enforce an expiration on the credential, then if the credential_context has changed, this can prompt the client to request a new credential.

## 7. Credential Issuance Protocol

Issuers provide an Issuer Private and Public Key, denoted skI and pkI respectively, used to produce tokens as input to the protocol. See Section 5 for 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, 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 host name 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.

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

### 7.1. Client-to-Issuer Request

Given Origin-provided input tokenChallenge and the fixed-length Issuer Public Key ID issuer_key_id, the Client first creates a credential request message using the CreateCredentialRequest function from [ARC] as follows:

```
request_context = concat(
  encode(2, len(tokenChallenge.issuer_name)),
  tokenChallenge.issuer_name,
  encode(2, len(tokenChallenge.origin_info)),
  tokenChallenge.origin_info,
  encode(2, len(tokenChallenge.credential_context)),
  tokenChallenge.credential_context,
  issuer_key_id)
(clientSecrets, request) = CreateCredentialRequest(request_context)
```

The Client then creates a CredentialRequest structure as follows:

```
struct {
  uint16_t token_type = 0xE5AC; /* Type ARC(P-256) */
  uint8_t truncated_issuer_key_id;
  uint8_t encoded_request[Nrequest];
} CredentialRequest;
```

The structure fields are defined as follows:

* "token_type" is a 2-octet integer.

* "truncated_issuer_key_id" is the least significant byte of the issuer_key_id (Section 5) in network byte order (in other words, the last 8 bits of issuer_key_id). This value is truncated so that Issuers cannot use issuer_key_id as a way of uniquely identifying Clients; see Section 9 and referenced information for more details.

* "encoded_request" is the Nrequest-octet request, computed as the serialization of the request value as defined in Section 4.2.1 of [ARC].

The Client then generates an HTTP POST request to send to the Issuer Request URL, with the CredentialRequest as the content. The media type for this request is "application/private-credential-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-credential-response
Content-Type: application/private-credential-request
Content-Length: <Length of CredentialRequest>

<Bytes containing the CredentialRequest>
```

### 7.2. Issuer-to-Client Response

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

* The CredentialRequest contains a supported token_type equal to value 0xE5AC.

* The CredentialRequest.truncated_token_key_id corresponds to the truncated key ID of an Issuer Public Key, with corresponding secret key skI, owned by the Issuer.

* The CredentialRequest.encoded_request is of the correct size (Nrequest).

If any of these conditions is 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 CredentialRequest.encoded_request according to Section 4.2.1 of [ARC], yielding request. 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 credential for the Client, the Issuer completes the issuance flow by an issuance response as follows:

```
response = CreateCredentialResponse(skI, pkI, request)
```

The Issuer then creates a CredentialResponse structured as follows:

```
struct {
   uint8_t encoded_response[Nresponse];
} CredentialResponse;
```

The structure fields are defined as follows:

* "encoded_response" is the Nresponse-octet encoded issuance response message, computed as the serialization of response as specified in Section 4.2.2 of [ARC].

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

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

<Bytes containing the CredentialResponse>
```

### 7.3. Credential Finalization

Upon receipt, the Client handles the response and, if successful, deserializes the content values CredentialResponse.encoded_response according to Section 4.2.2 of [ARC] yielding response. If deserialization fails, the Client aborts the protocol. Otherwise, the Client processes the response as follows:

```
credential = FinalizeCredential(clientSecrets, pkI, request, response)
```

The Client then saves the credential structure, associated with the given Issuer Name, to use when producing Token values in response to future token challenges.

## 8. Token Redemption Protocol

The token redemption protocol takes as input TokenChallenge and presentation limit values from [AUTHSCHEME], Section 2.1; the presentation limit is sent as an additional attribute within the HTTP challenge as described in Section 6. Clients use credentials from the issuance protocol in producing tokens bound to the TokenChallenge. The process for producing a token in this way, as well as verifying a resulting token, is described in the following sections.

### 8.1. Token Creation

Given a TokenChallenge value as input, denoted challenge, a presentation limit, denoted presentation_limit, and a previously computed credential that is valid for the Issuer identifier in the challenge, denoted credential, Clients compute a credential presentation value as follows:

```
presentation_context = concat(
  encode(2, len(challenge.issuer_name)),
  challenge.issuer_name,
  encode(2, len(challenge.origin_info)),
  challenge.origin_info,
  encode(2, len(challenge.redemption_context)),
  challenge.redemption_context,
  issuer_key_id)
state = MakePresentationState(credential, presentation_context, presentation_limit)
newState, nonce, presentation = Present(state)
```

Subsequent presentations MUST use the updated state, denoted newState. Reusing the original state will break the presentation unlinkability properties of ARC; see Section 9.

The resulting Token value is then constructed as follows:

```
struct {
    uint16_t token_type = 0xE5AC; /* Type ARC(P-256) */
    uint32_t presentation_nonce;
    uint8_t challenge_digest[32];
    uint8_t issuer_key_id[Nid];
    uint8_t presentation[Npresentation];
} Token;
```

The structure fields are defined as follows:

* "token_type" is a 2-octet integer, in network byte order, equal to 0xE5AC.

* "presentation_nonce" is a 4-octet integer, in network byte order, equal to the nonce output from ARC.

* "challenge_digest" is a 32-octet value containing the hash of the original TokenChallenge, SHA-256(TokenChallenge).

* "issuer_key_id" is a Nid-octet identifier for the Issuer Public Key, computed as defined in Section 5.

* "presentation" is a Npresentation-octet presentation, set to the serialized presentation value (see Section 4.3.2 of [ARC] for serialiation details).

### 8.2. Token Verification

Given a deserialized presentation from the token, denoted presentation and obtained by deserializing a presentation according to Section 4.3.2 of [ARC], a presentation limit, denoted presentation_limit, a fixed-length Issuer Public Key ID, denoted issuer_key_id, a presentation nonce from a token, denoted nonce, and the digest of a token challenge, denoted challenge_digest, verifying a Token requires invoking the VerifyPresentation function from Section 4.3.3 of [ARC] in the following ways:

```
request_context = concat(
  encode(2, len(tokenChallenge.issuer_name)),
  tokenChallenge.issuer_name,
  encode(2, len(tokenChallenge.origin_info)),
  tokenChallenge.origin_info,
  encode(2, len(tokenChallenge.credential_context)),
  tokenChallenge.credential_context,
  issuer_key_id)

presentation_context = concat(
  encode(2, len(tokenChallenge.issuer_name)),
  tokenChallenge.issuer_name,
  encode(2, len(tokenChallenge.origin_info)),
  tokenChallenge.origin_info,
  encode(2, len(tokenChallenge.redemption_context)),
  tokenChallenge.redemption_context,
  issuer_key_id)

valid = VerifyPresentation(
  skI,
  pkI,
  request_context,
  presentation_context,
  nonce,
  presentation,
  presentation_limit)
```

This function returns True if the CredentialToken is valid, and False otherwise.

To prevent double spending, the Origin SHOULD perform a check that the tag output from VerifyPresentation has not previously been seen. It can do this by checking the tag against previously seen tags. To improve double spend performance, the Origin can store and look up tags corresponding to the associated request_context and presentation_context values.

## 9. Security Considerations

Privacy considerations for tokens based on deployment details, such as issuer configuration and issuer selection, are discussed in Section 6.1 of [ARCHITECTURE]. Note that ARC requires a joint Origin and Issuer configuration given that it is privately verifiable.

ARC offers Origin-Client unlinkability, Issuer-Client unlinkability, and redemption context unlinkability, as described in Section 3.3 of [ARCHITECTURE], with one exception. While redemption context unlinkability is achieved by re-randomizing credentials every time they are presented as tokens, there is a reduction in the anonymity set in the case of presentation nonce collisions, as detailed in Section 7.2 of [ARC].

## 10. IANA Considerations

This section documents IANA registry updates.

### 10.1. Privacy Pass Token Types Registry Updates

This document updates the "Privacy Pass Token Type" Registry with the following entries.

* Value: 0xE5AC
* Name: ARC (P-256)
* Token Structure: As defined in Section 2.2 of [AUTHSCHEME]
* Token Key Encoding: Serialized as described in Section 5
* TokenChallenge Structure: As defined in Section 2.1 of [AUTHSCHEME]
* Public Verifiability: N
* Public Metadata: N
* Private Metadata: N
* Nk: 0 (not applicable)
* Nid: 32
* Reference: This document
* Notes: None

### 10.2. Media Types

The following entries should be added to the IANA "media types" registry:

* "application/private-credential-request"

* "application/private-credential-response"

The templates for these entries are listed below and the reference should be this RFC.

#### 10.2.1. "application/private-credential-request" media type

Type name: application
Subtype name: private-credential-request
Required parameters: N/A
Optional parameters: N/A
Encoding considerations: "binary"
Security considerations: see Section 9
Interoperability considerations: N/A
Published specification: this specification
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: 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: IETF

#### 10.2.2. "application/private-credential-response" media type

Type name: application
Subtype name: private-credential-response
Required parameters: N/A
Optional parameters: N/A
Encoding considerations: "binary"
Security considerations: see Section 9
Interoperability considerations: N/A
Published specification: this specification
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: 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: IETF

# 11. References

## 11.1. Normative References

[ARC] Yun, C. and C. A. Wood, "Anonymous Rate-Limited Credentials", Work in Progress, Internet-Draft, draft-yun-privacypass-crypto-arc-00, 20 October 2025, <https://datatracker.ietf.org/doc/html/draft-yun-privacypass-crypto-arc-00>.

[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/rfc/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/rfc/rfc9577>.

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

[ISSUANCE] Celi, S., Davidson, A., Valdez, S., and C. A. Wood, "Privacy Pass Issuance Protocols", RFC 9578, DOI 10.17487/RFC9578, June 2024, <https://www.rfc-editor.org/rfc/rfc9578>.

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

[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>.

[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/rfc/rfc8446>.

[VOPRFS] 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/rfc/rfc9497>.

### 11.2. Informative References

[DAP] Geoghegan, T., Patton, C., Pitman, B., Rescorla, E., and C. A. Wood, "Distributed Aggregation Protocol for Privacy Preserving Measurement", Work in Progress, Internet-Draft, draft-ietf-ppm-dap-16, 2 September 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-ppm-dap-16>.

[OHTTP] Thomson, M. and C. A. Wood, "Oblivious HTTP", RFC 9458, DOI 10.17487/RFC9458, January 2024, <https://www.rfc-editor.org/rfc/rfc9458>.

[RATE-LIMITED] Hendrickson, S., Iyengar, J., Pauly, T., Valdez, S., and C. A. Wood, "Rate-Limited Token Issuance Protocol", Work in Progress, Internet-Draft, draft-ietf-privacypass-rate-limit-tokens-06, 1 April 2024, <https://datatracker.ietf.org/doc/html/draft-ietf-privacypass-rate-limit-tokens-06>.

## Acknowledgments

The authors would like to thank Tommy Pauly and the authors of [RATE-LIMITED] for helpful discussions on rate-limited tokens.

## Authors’ Addresses

Cathie Yun
Apple, Inc.

Email: cathieyun@gmail.com

Christopher A. Wood
Apple, Inc.
Email: caw@heapingbits.net