

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: 6 October 2022

T. Pauly
Apple Inc.
S. Valdez
Google LLC
C. A. Wood
Cloudflare
4 April 2022

The Privacy Pass HTTP Authentication Scheme
draft-ietf-privacypass-auth-scheme-02

Abstract

This document defines an HTTP authentication scheme that can be used by clients to redeem Privacy Pass tokens with an origin. It can also be used by origins to challenge clients to present an acceptable Privacy Pass token.

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 6 October 2022.

Copyright Notice

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

Internet-Draft

Privacy Pass Authentication

April 2022

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

1.	Introduction	2
1.1.	Terminology	3
2.	HTTP Authentication Scheme	4
2.1.	Token Challenge	4
2.1.1.	Redemption Context Construction	6
2.1.2.	Token Caching	7
2.2.	Token Redemption	7
3.	Issuance Protocol Requirements	9
4.	User Interaction	9
5.	Security Considerations	10
6.	IANA Considerations	11
6.1.	Authentication Scheme	11
6.2.	Token Type Registry	12
7.	References	12
7.1.	Normative References	12
7.2.	Informative References	13
	Authors' Addresses	13

[1.](#) Introduction

Privacy Pass tokens are unlinkable authenticators that can be used to anonymously authorize a client (see [\[I-D.ietf-privacypass-architecture\]](#)). A client possessing such a token is able to prove that it was able to get a token issued by a token issuer -- based on some check from a token issuer, such as authentication or solving a CAPTCHA -- without allowing the relying party redeeming the client's token (the origin) to link it with issuance flow.

Different types of authenticators, using different token issuance protocols, can be used as Privacy Pass tokens.

This document defines a common HTTP authentication scheme ([RFC7235]), PrivateToken, that allows clients to redeem various kinds of Privacy Pass tokens.

Clients and relying parties interact using this scheme to perform the token challenge and token redemption flow. Clients use a token issuance protocol to actually fetch tokens to redeem.

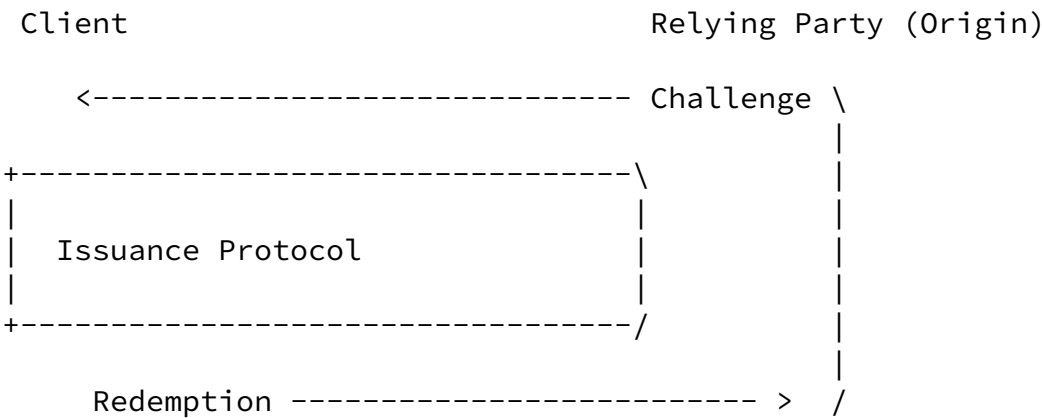


Figure 1: Token Architectural Components

In addition to working with different token issuance protocols, this scheme supports optionally associating tokens with origin-chosen contexts and specific origin names. Relying parties that request and redeem tokens can choose a specific kind of token, as appropriate for its use case. These options allow for different deployment models to prevent double-spending, and allow for both interactive (online challenges) and non-interactive (pre-fetched) tokens.

1.1. 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.

Unless otherwise specified, this document encodes protocol messages in TLS notation from [TLS13], Section 3.

This document uses the terms "Client", "Origin", "Issuer", "Issuance Protocol", and "Token" as defined in [\[I-D.ietf-privacypass-architecture\]](#). It additionally uses the following terms in more specific ways:

- * Issuer key: Keying material that can be used with an issuance protocol to create a signed token.

- * Token challenge: A requirement for tokens sent from an origin to a client, using the "WWW-Authenticate" HTTP header. This challenge is bound to a specific token issuer and issuance protocol, and may be additionally bound to a specific context or origin name.
- * Token redemption: An action by which a client presents a token to an origin, using the "Authorization" HTTP header.

[2.](#) HTTP Authentication Scheme

Token redemption is performed using HTTP Authentication ([\[RFC7235\]](#)), with the scheme "PrivateToken". Origins challenge clients to present a token from a specific issuer ([Section 2.1](#)). Once a client has received a token from that issuer, or already has a valid token available, it presents the token to the origin ([Section 2.2](#)).

[2.1.](#) Token Challenge

Origins send a token challenge to clients in an "WWW-Authenticate" header with the "PrivateToken" scheme. This challenge includes a TokenChallenge message, along with information about what keys to use when requesting a token from the issuer.

Origins that support this authentication scheme need to handle the following tasks:

1. Select which issuer to use, and configure the issuer name and token-key to include in WWW-Authenticate challenges.

2. Determine a redemption context construction to include in the TokenChallenge, as discussed in [Section 2.1.1](#).
3. Select the origin information to include in the TokenChallenge. This can be empty to allow fully cross-origin tokens, a single origin name that matches the origin itself, or a list of origin names containing the origin.

The TokenChallenge message has the following structure:

```
struct {  
    uint16_t token_type;  
    opaque issuer_name<1..2^16-1>;  
    opaque redemption_context<0..32>;  
    opaque origin_info<0..2^16-1>;  
} TokenChallenge;
```

The structure fields are defined as follows:

- * "token_type" is a 2-octet integer, in network byte order. This type indicates the issuance protocol used to generate the token. Values are registered in an IANA registry, [Section 6.2](#). Challenges with unsupported token_type values MUST be ignored.
- * "issuer_name" is a string containing the name of the issuer. This is a hostname that is used to identify the issuer that is allowed to issue tokens that can be redeemed by this origin. The string is prefixed with a 2-octet integer indicating the length, in network byte order.
- * "redemption_context" is an optional field. If present, it allows the origin to require that clients fetch tokens bound to a specific context, as opposed to reusing tokens that were fetched for other contexts. See [Section 2.1.1](#) for example contexts that might be useful in practice. When present, this value is a 32-byte context generated by the origin. Valid lengths for this field are either 0 or 32 bytes. The field is prefixed with a single octet indicating the length. Challenges with redemption_context values of invalid lengths MUST be ignored.
- * "origin_info" is an optional string containing one or more origin

names, which allows a token to be scoped to a specific set of origins. The string is prefixed with a 2-octet integer indicating the length, in network byte order. If empty, any non-origin-specific token can be redeemed. If the string contains multiple origin names, they are delimited with commas ",", without any whitespace.

When used in an authentication challenge, the "PrivateToken" scheme uses the following attributes:

- * "challenge", which contains a base64url-encoded [\[RFC4648\]](#) TokenChallenge value. Since the length of the challenge is not fixed, the base64url data MUST include padding. This MUST be unique for every 401 HTTP response to prevent replay attacks. This attribute is required for all challenges.
- * "token-key", which contains a base64url encoding of the public key for use with the issuance protocol indicated by the challenge. Since the length of the key is not fixed, the base64url data MUST include padding. This attribute MAY be omitted in deployments where clients are able to retrieve the issuer key using an out-of-band mechanism.
- * "max-age", an optional attribute that consists of the number of seconds for which the challenge will be accepted by the origin.

Clients can ignore the challenge if the token-key is invalid or otherwise untrusted.

The header MAY also include the standard "realm" attribute, if desired. Issuance protocols MAY require other attributes.

As an example, the WWW-Authenticate header could look like this:

WWW-Authenticate: PrivateToken challenge=abc..., token-key=123...

Upon receipt of this challenge, a client uses the message and keys in the issuance protocol indicated by the token_type. If the TokenChallenge has a token_type the client does not recognize or support, it MUST NOT parse or respond to the challenge. If the TokenChallenge contains a non-empty origin_info field, the client

MUST validate that the name of the origin that issued the authentication challenge is included in the list of origin names. Clients MAY have further restrictions and requirements around validating when a challenge is considered acceptable or valid. For example, clients can choose to reject challenges that list origin names for which current connection is not authoritative (according to the TLS certificate).

Caching and pre-fetching of tokens is discussed in [Section 2.1.2](#).

Note that it is possible for the WWW-Authenticate header to include multiple challenges. This allows the origin to indicate support for different token types, issuers, or to include multiple redemption contexts. For example, the WWW-Authenticate header could look like this:

WWW-Authenticate: PrivateToken challenge=abc..., token-key=123..., PrivateToken challenge=def..., token-key=234...

[2.1.1](#). Redemption Context Construction

The TokenChallenge redemption context allows the origin to determine the context in which a given token can be redeemed. This value can be a unique per-request nonce, constructed from 32 freshly generated random bytes. It can also represent state or properties of the client session. Some example properties and methods for constructing the corresponding context are below. This list is not exhaustive.

- * Context bound to a given time window: Construct redemption context as SHA256(current time window).
- * Context bound to a client location: Construct redemption context as SHA256(client IP address prefix).

- * Context bound to a given time window and location: Construct redemption context as SHA256(current time window, client IP address prefix).

An empty redemption context is not bound to any property of the client session. Preventing double spending on tokens requires the origin to keep state associated with the redemption context. The size of this state varies based on the size of the redemption

context. For example, double spend state for unique, per-request redemption contexts does only needs to exist within the scope of the request connection or session. In contrast, double spend state for empty redemption contexts must be stored and shared across all requests until token-key expiration or rotation.

[2.1.2.](#) Token Caching

Clients can generate multiple tokens from a single TokenChallenge, and cache them for future use. This improves privacy by separating the time of token issuance from the time of token redemption, and also allows clients to avoid any overhead of receiving new tokens via the issuance protocol.

Cached tokens can only be redeemed when they match all of the fields in the TokenChallenge: token_type, issuer_name, redemption_context, and origin_info. Clients ought to store cached tokens based on all of these fields, to avoid trying to redeem a token that does not match. Note that each token has a unique client nonce, which is sent in token redemption ([Section 2.2](#)).

If a client fetches a batch of multiple tokens for future use that are bound to a specific redemption context (the redemption_context in the TokenChallenge was not empty), clients SHOULD discard these tokens upon flushing state such as HTTP cookies [[COOKIES](#)], or changing networks. Using these tokens in a context that otherwise would not be linkable to the original context could allow the origin to recognize a client.

[2.2.](#) Token Redemption

The output of the issuance protocol is a token that corresponds to the origin's challenge (see [Section 2.1](#)). A token is a structure that begins with a two-octet field that indicates a token type, which MUST match the token_type in the TokenChallenge structure.

```
struct {
```

```
uint16_t token_type;
uint8_t nonce[32];
uint8_t challenge_digest[32];
uint8_t token_key_id[Nid];
uint8_t authenticator[Nk];
} Token;
```

The structure fields are defined as follows:

- * "token_type" is a 2-octet integer, in network byte order. This value must match the value in the challenge ([Section 2.1](#)).
- * "nonce" is a 32-octet message containing a client-generated random nonce.
- * "challenge_digest" is a 32-octet message containing the hash of the original TokenChallenge, SHA256(TokenChallenge).
- * "token_key_id" is an Nid-octet identifier for the the token authentication key. The value of this field is defined by the token_type and corresponding issuance protocol.
- * "authenticator" is a Nk-octet authenticator that covers the preceding fields in the token. The value of this field is defined by the token_type and corresponding issuance protocol.

The authenticator value in the Token structure is computed over the token_type, nonce, context, and token_key_id fields.

When used for client authorization, the "PrivateToken" authentication scheme defines one parameter, "token", which contains the base64url-encoded Token struct. Since the length of the Token struct is not fixed, the base64url data MUST include padding. All unknown or unsupported parameters to "PrivateToken" authentication credentials MUST be ignored.

Clients present this Token structure to origins in a new HTTP request using the Authorization header as follows:

Authorization: PrivateToken token=abc...

For token types that support public verifiability, origins verify the token authenticator using the public key of the issuer, and validate that the signed message matches the concatenation of the client nonce and the hash of a valid TokenChallenge. For context-bound tokens, origins store or reconstruct the contexts of previous TokenChallenge structures in order to validate the token. A TokenChallenge MAY be

bound to a specific HTTP session with client, but origins can also accept tokens for valid challenges in new sessions. Origins SHOULD implement some form of double-spend prevention that prevents a token with the same nonce from being redeemed twice. This prevents clients from "replaying" tokens for previous challenges. For context-bound tokens, this double-spend prevention can require no state or minimal state, since the context can be used to verify token uniqueness.

If a client is unable to fetch a token, it MUST react to the challenge as if it could not produce a valid Authorization response.

[3.](#) Issuance Protocol Requirements

Clients initiate the issuance protocol using a challenge, a randomly generated nonce, and a public key for the issuer. The issuance protocol itself can be any interactive protocol between client, issuer, or other parties that produces a valid authenticator over the client's input, subject to the following security requirements.

1. Unconditional input secrecy. The issuance protocol MUST NOT reveal anything about the client's private input, including the challenge and nonce. The issuance protocol can reveal the issuer public key for the purposes of determining which private key to use in producing the issuance protocol. A result of this property is that the redemption flow is unlinkable from the issuance flow.
2. One-more forgery security. The issuance protocol MUST NOT allow malicious clients to forge tokens without interacting with the issuer directly.
3. Concurrent security. The issuance protocol MUST be safe to run concurrently with arbitrarily many clients.

[4.](#) User Interaction

When used in contexts like websites, origins that challenge clients for tokens need to consider how to optimize their interaction model to ensure a good user experience.

Tokens challenges can be performed without explicit user involvement, depending on the issuance protocol. If tokens are scoped to a specific origin, there is no need for per-challenge user interaction. Note that the issuance protocol may separately involve user interaction if the client needs to be newly validated.

If a client cannot use cached tokens to respond to a challenge (either because it has run out of cached tokens or the associated context is unique), the token issuance process can add user-perceivable latency. Origins need not block useful work on token authentication. Instead, token authentication can be used in similar ways to CAPTCHA validation today, but without the need for user interaction. If issuance is taking a long time, a website could show an indicator that it is waiting, or fall back to another method of user validation.

An origin **MUST NOT** use more than one redemption context value for a given token type and issuer per client request. If an origin issues a large number of challenges with unique contexts, such as more than once for each request, this can indicate that the origin is either not functioning correctly or is trying to attack or overload the client or issuance server. In such cases, a client **MUST** ignore redundant token challenges for the same request and **SHOULD** alert the user if possible.

Origins **MAY** include multiple challenges, where each challenge refers to a different issuer or a different token type, to allow clients to choose a preferred issuer or type.

5. Security Considerations

The security properties of token challenges vary depending on whether the challenge contains a redemption context or not, as well as whether the challenge is per-origin or not. For example, cross-origin tokens with empty contexts can be replayed from one party by another, as shown below.

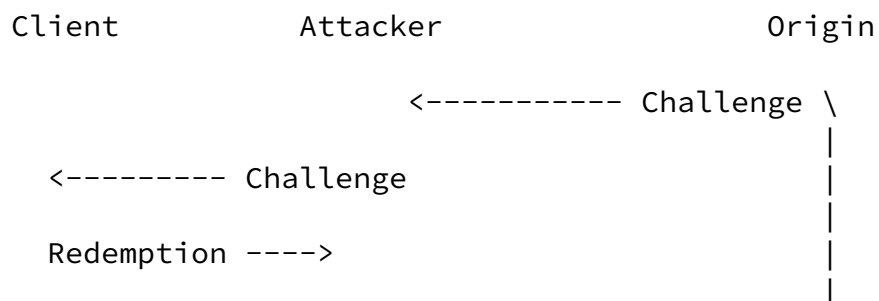


Figure 2: Token Architectural Components

Token challenges that include non-empty origin_info bind tokens to one or more specific origins. As described in [Section 2.1](#), clients only accept such challenges from origin names listed in the origin_info string. Even if multiple origins are listed, a token can only be redeemed for an origin if the challenge has an exact match

for the origin_info. For example, if "a.example.com" issues a challenge with an origin_info string of "a.example.com,b.example.com", a client could redeem a token fetched for this challenge if and only if "b.example.com" also included an origin_info string of "a.example.com,b.example.com". On the other hand, if "b.example.com" had an origin_info string of "b.example.com" or "b.example.com,a.example.com" or "a.example.com,b.example.com,c.example.com", the string would not match and the client would need to use a different token.

Context-bound token challenges require clients to obtain matching tokens when challenged, rather than presenting a token that was obtained from a different context in the past. This can make it more likely that issuance and redemption events will occur at approximately the same time. For example, if a client is challenged for a token with a unique context at time T1 and then subsequently obtains a token at time T2, a colluding issuer and origin can link this to the same client if T2 is unique to the client. This linkability is less feasible as the number of issuance events at time T2 increases. Depending on the "max-age" token challenge attribute, clients MAY try to augment the time between getting challenged then redeeming a token so as to make this sort of linkability more difficult. For more discussion on correlation risks between token issuance and redemption, see [[I-D.ietf-privacypass-architecture](#)].

As discussed in [Section 2.1](#), clients SHOULD discard any context-bound tokens upon flushing cookies or changing networks, to prevent an origin using the redemption context state as a cookie to recognize clients.

Applications SHOULD constrain tokens to a single origin unless the use case can accommodate such replay attacks.

All random values in the challenge and token MUST be generated using a cryptographically secure source of randomness.

[6.](#) IANA Considerations

[6.1.](#) Authentication Scheme

This document registers the "PrivateToken" authentication scheme in the "Hypertext Transfer Protocol (HTTP) Authentication Scheme Registry" established by [[RFC7235](#)].

Authentication Scheme Name: PrivateToken

Pointer to specification text: [Section 2](#) of this document

Pauly, et al.

Expires 6 October 2022

[Page 11]

Internet-Draft

Privacy Pass Authentication

April 2022

[6.2.](#) Token Type Registry

The "Token Type" registry lists identifiers for issuance protocols defined for use with the Privacy Pass token authentication scheme. These identifiers are two-byte values, so the maximum possible value is 0xFFFF = 65535.

Template:

- * Value: The two-byte identifier for the algorithm
- * Name: Name of the issuance protocol
- * Publicly Verifiable: A Y/N value indicating if the output tokens are publicly verifiable
- * Public Metadata: A Y/N value indicating if the output tokens can contain public metadata.
- * Private Metadata: A Y/N value indicating if the output tokens can contain private metadata.
- * Nk: The length in bytes of an output authenticator
- * Nid: The length of the token key identifier

* Reference: Where this algorithm is defined

The initial contents for this registry are defined in the table below.

Value	Name	Publicly Verifiable	Public Metadata	Private Metadata	Nk	Nid	Reference
0x0000	(reserved)	N/A	N/A	N/A	N/A	N/A	N/A

Table 1: Token Types

7. References

7.1. Normative References

Pauly, et al. Expires 6 October 2022 [Page 12]

Internet-Draft Privacy Pass Authentication April 2022

- [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>>.
- [RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", [RFC 4648](#), DOI 10.17487/RFC4648, October 2006, <<https://www.rfc-editor.org/rfc/rfc4648>>.
- [RFC7235] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Authentication", [RFC 7235](#), DOI 10.17487/RFC7235, June 2014, <<https://www.rfc-editor.org/rfc/rfc7235>>.
- [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](https://www.rfc-editor.org/rfc/rfc8446), DOI 10.17487/RFC8446, August 2018, <<https://www.rfc-editor.org/rfc/rfc8446>>.

[7.2.](#) Informative References

[COOKIES] Chen, L., Englehardt, S., West, M., and J. Wilander, "Cookies: HTTP State Management Mechanism", Work in Progress, Internet-Draft, [draft-ietf-httpbis-rfc6265bis-09](https://datatracker.ietf.org/doc/html/draft-ietf-httpbis-rfc6265bis-09), 19 October 2021, <<https://datatracker.ietf.org/doc/html/draft-ietf-httpbis-rfc6265bis-09>>.

[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>>.

Authors' Addresses

Tommy Pauly
Apple Inc.
One Apple Park Way
Cupertino, California 95014,
United States of America
Email: tpauly@apple.com

Pauly, et al.

Expires 6 October 2022

[Page 13]

Internet-Draft

Privacy Pass Authentication

April 2022

Steven Valdez
Google LLC
Email: svaldez@chromium.org

Christopher A. Wood
Cloudflare
Email: caw@heapingbits.net

