Workgroup: Network Working Group
Internet-Draft:
draft-ietf-privacypass-auth-scheme-10
Published: 8 May 2023
Intended Status: Standards Track
Expires: 9 November 2023
Authors: T. Pauly S. Valdez C. A. Wood
Apple Inc. Google LLC Cloudflare

The Privacy Pass HTTP Authentication Scheme

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 9 November 2023.

Table of Contents

1. Introduction

Privacy Pass tokens are unlinkable authenticators that can be used to anonymously authorize a client (see [ARCHITECTURE]). Tokens are generated by token issuers, on the basis of authentication, attestation, or some previous action such as solving a CAPTCHA. A client possessing such a token is able to prove that it was able to get a token issued, without allowing the relying party redeeming the client's token (the origin) to link it with the 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 ([RFC9110], Section 11), PrivateToken, that allows clients to redeem various kinds of Privacy Pass tokens.

Clients and relying parties (origins) interact using this scheme to perform the token challenge and token redemption flow. In particular, origins challenge clients for a token with an HTTP Authentication challenge (using the WWW-Authenticate response header field). Clients then respond to that challenge with an HTTP authentiation response (using the Authorization request header field). Clients produce an authentication response based on the origin's token challenge by running the token issuance protocol [ISSUANCE]. The act of presenting a token in an Authorization

request header is referred to as token redemption. This interaction between client and origin is shown below.

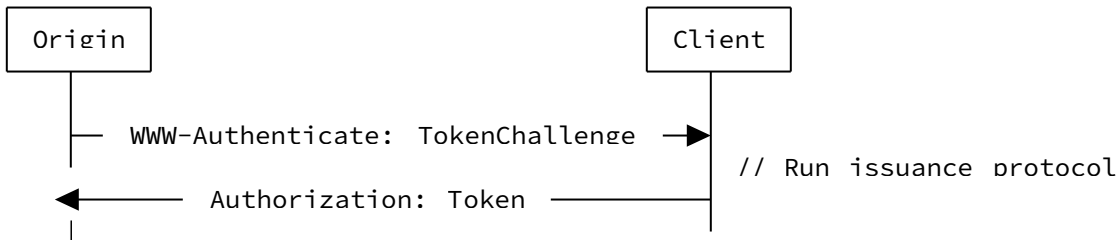

Figure 1: Challenge-response redemption protocol flow

In addition to working with different token issuance protocols, this scheme optionally supports use of tokens that are associated 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 [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 field. 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 in an HTTP request, using the "Authorization" HTTP header field.

2. HTTP Authentication Scheme

Token redemption is performed using HTTP Authentication ([RFC9110], Section 11), 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).

Unlike many authentication schemes in which a client will present the same credentials across multiple requests, tokens used with the "PrivateToken" scheme are single-use credentials, and are not reused. Spending the same token value more than once allows the origin to link multiple transactions to the same client. In deployment scenarios where origins send token challenges to request tokens, origins ought to expect at most one request containing a token from the client in reaction to a particular challenge.

Origins SHOULD minimize the number of challenges sent on a particular client session, such as a unique TLS session between a client and origin (referred to as the "redemption context" in [ARCHITECTURE]). Clients can have implementation-specific policy to minimize the number of tokens that can be retrieved by origins, so origins are advised to only request tokens when necessary within a single session. See Section 3 for more discussion on how to optimize token challenges to improve the user experience.

2.1. Token Challenge

Origins send a token challenge to clients in an "WWW-Authenticate" header field 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 itself.

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 5.2. Challenges with unsupported token_type values MUST be ignored. This value determines the structure and semantics of the rest of the structure.

* "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. If this field is not empty, the Origin MUST include its own name as one of the names in the list.

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

- "challenge", which contains a base64url-encoded [RFC4648] TokenChallenge value. Since the length of the challenge is not fixed, the base64url value MUST include padding. As an Authentication Parameter (auth-param from [RFC9110], Section 11.2), the value can be either a token or a quoted-string, and might be required to be a quoted-string if the base64url string includes "=" characters. This challenge value MUST be unique for every 401 HTTP response to prevent replay attacks. This parameter 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 value MUST include padding. As an Authentication Parameter (auth-param from [RFC9110], Section 11.2), the value can be either a token or a quoted-string, and might be required to be a quoted-string if the base64url string includes "=" characters. This parameter MAY be omitted in deployments where clients are able to retrieve the issuer key using an out-of-band mechanism.

- "max-age", an optional parameter 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 field MAY also include the standard "realm" parameter, if desired. Issuance protocols MAY require other parameters. Clients SHOULD ignore unknown parameters in challenges, except if otherwise specified by issuance protocols.

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

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

Upon receipt of this challenge, a client validates the TokenChallenge before responding to it. Validation requirements are as follows:

- The TokenChallenge structure is well-formed;

- The token_type is recognized and supported by the client; and

*If the origin_info field is non-empty, the name of the origin that issued the authentication challenge is included in the list of origin names.

If validation fails, the client MUST NOT process or respond to the challenge. Clients MAY have further restrictions and requirements around validating when a challenge is considered acceptable or valid. For example, clients can choose to ignore 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 field 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 field could look like this:

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

Origins should only include challenges for different types of issuance protocols with functionally equivalent properties. For instance, both issuance protocols in [ISSUANCE] have the same functional properties, albeit with different mechanisms for verifying the resulting tokens during redemption. Since clients are free to choose which challenge they want to consume when presented with options, mixing multiple challenges with different functional properties for one use case is nonsensical. If the origin has a preference for one challenge over another (for example, if one uses a token type that is faster to verify), it can sort it to be first in the list of challenges as a hint to the client.

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.

Origins that share redemption contexts, i.e., by using the same redemption context, choosing the same issuer, and providing the same origin_info field in the TokenChallenge, must necessarily share state required to enforce double spend prevention. Origins should consider the operational complexity of this shared state before choosing to share redemption contexts. Failure to successfully synchronize this state and use it for double spend prevention can allow Clients to redeem tokens to one Origin that were issued after an interaction with another Origin that shares the context.

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). This value determines the structure and semantics of the rest of the structure.

- "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 value of constant Nk depends on token_type, as defined in Section 5.2.

The authenticator value in the Token structure is computed over the token_type, nonce, challenge_digest, 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 value MUST include padding. As an Authentication Parameter (auth-param from [RFC9110], Section 11.2), the value can be either a token or a quoted-string, and might be required to be a quoted-string if the base64url string includes "=" characters. 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 field 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 TLS session with a 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. 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.

An origin MUST NOT assume that token challenges will always yield a valid token. Clients might experience issues running the issuance protocol, e.g., because the attester or issuer is unavailable, or clients might simply not support the requested token type. Origins SHOULD account for such operational or interoperability failures by offering clients an alternative type of challenge such as CAPTCHA for accessing a resource.

For example, consider a scenario in which the client is a web browser, and the origin can accept either a token or a solution to a puzzle intended to determine if the client is a real human user. The origin would send clients a 401 HTTP response that contains a token challenge in a "WWW-Authenticate" header field along with content that contains the puzzle to display to the user. Clients that are able to respond with a token will be able to automatically return the token and not show the puzzle, while clients that either do not support tokens or are unable to fetch tokens at a particular time can present the user with the puzzle.

To mitigate the risk of deployments becoming dependent on tokens, clients and servers SHOULD grease their behavior unless explicitly configured not to. In particular, clients SHOULD ignore token challenges with some non-zero probability. Likewise, origins SHOULD randomly choose to not challenge clients for tokens with some non-zero probability. Moreover, origins SHOULD include random token types, from the Reserved list of "greased" types (defined in Section 5.2), with some non-zero probability.

4. 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

Moreover, when a Client holds cross-origin tokens with empty contexts, it is possible for any Origin in the cross-origin set to deplete that Client set of tokens. To prevent this from happening, tokens can be scoped to single Origins (with non-empty origin_info) such that they can only be redeemed for a single Origin. Alternatively, if tokens are cross-Origin, Clients can use alternate methods to prevent many tokens from being redeemed at once. For example, if the Origin requests an excess of tokens, the Client could choose to not present any tokens for verification if a redemption had already occurred in a given time window.

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 parameter, 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 [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. Replays are also possible if the client redeems a token sent as part of 0-RTT data. If successful token redemption produces side effects, origins SHOULD implement an anti-replay mechanism to mitigate the harm of such replays. See [TLS13], Section 8 and [RFC9001], Section 9.2 for details about anti-replay mechanisms, as well as [RFC8470], Section 3 for discussion about safety considerations for 0-RTT HTTP data.

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

5. IANA Considerations

5.1. Authentication Scheme

This document registers the "PrivateToken" authentication scheme in the "Hypertext Transfer Protocol (HTTP) Authentication Scheme Registry" defined in [RFC9110], Section 16.4.

Authentication Scheme Name: PrivateToken

Pointer to specification text: Section 2 of this document

5.2. Token Type Registry

IANA is requested to create a new "Privacy Pass Token Type" registry in a new "Privacy Pass Parameters" page to list 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

* Token Structure: The contents of the Token structure

* TokenChallenge Structure: The contents of the TokenChallenge structure

* 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

*Notes: Any notes associated with the entry

New entries in this registry are subject to the Specification Required registration policy ([RFC8126], Section 4.6). Designated experts need to ensure that the token type is sufficiently clearly defined to be used for both token issuance and redemption, and meets the common security and privacy requirements for issuance protocols defined in Section 3.2 of [ARCHITECTURE].

This registry also will also allow provisional registrations to allow for experimentation with protocols being developed. Designated experts review, approve, and revoke provisional registrations.

Values 0xFF00-0xFFFF are reserved for private use, to enable proprietary uses and limited experimentation.

This document defines several Reserved values, which can be used by clients and servers to send "greased" values in token challenges and responses to ensure that implementations remain able to handle unknown token types gracefully (this technique is inspired by [RFC8701]). Implemenations SHOULD select reserved values at random when including them in greased messages. Servers can include these in TokenChallenge structures, either as the only challenge when no real token type is desired, or as one challenge in a list of challenges that include real values. Clients can include these in Token structures when they are not able to present a real token response. The contents of the Token structure SHOULD be filled with random bytes when using greased values.

The initial contents for this registry consist of the following Values. For each Value, the Name is "RESERVED", the Publicly Verifiable, Public Metadata, Private Metadata, Nk, and Nid attributes are all assigned "N/A", the Reference is this document, and the Notes attribute is "None". The iniital list of Values is as follows:

*0x0000

*0x02AA

*0x1132

*0x2E96

*0x3CD3

*0x4473

*0x5A63

*0x6D32

*0x7F3F

*0x8D07

*0x916B

*0xA6A4

*0xBEAB

*0xC3F3

*0xDA42

*0xE944

*0xF057

Additionally, the registry is to be initialized with the following entry for Private Use.

*Value: 0xFF00-0xFFFF

*Name: Private Use

*Token Structure: As defined in Section 2.2

*TokenChallenge Structure: As defined in Section 2.1

*Publicly Verifiable: N/A

*Public Metadata: N/A

*Private Metadata: N/A

*Nk: N/A

*Nid: N/A

*Reference: This document

*Notes: None

6. References

6.1. Normative References

[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>.

[RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <https://www.rfc-editor.org/rfc/rfc8126>.

[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>.

[RFC9110] 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/rfc/rfc9110>.

[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>.

6.2. Informative References

[ARCHITECTURE] Davidson, A., Iyengar, J., and C. A. Wood, "The Privacy Pass Architecture", Work in Progress, Internet-Draft, draft-ietf-privacypass-architecture-12, 2 May 2023, <https://datatracker.ietf.org/doc/html/draft-ietf-privacypass-architecture-12>.

[COOKIES] Bingler, S., West, M., and J. Wilander, "Cookies: HTTP State Management Mechanism", Work in Progress, Internet-Draft, draft-ietf-httpbis-rfc6265bis-11, 7 November 2022, <https://datatracker.ietf.org/doc/html/draft-ietf-httpbis-rfc6265bis-11>.

[ISSUANCE] Celi, S., Davidson, A., Faz-Hernandez, A., Valdez, S., and C. A. Wood, "Privacy Pass Issuance Protocol", Work in Progress, Internet-Draft, draft-ietf-privacypass-

protocol-10, 6 March 2023, <https://datatracker.ietf.org/doc/html/draft-ietf-privacypass-protocol-10>.

[RFC8470] Thomson, M., Nottingham, M., and W. Tarreau, "Using Early Data in HTTP", RFC 8470, DOI 10.17487/RFC8470, September 2018, <https://www.rfc-editor.org/rfc/rfc8470>.

[RFC8701] Benjamin, D., "Applying Generate Random Extensions And Sustain Extensibility (GREASE) to TLS Extensibility", RFC 8701, DOI 10.17487/RFC8701, January 2020, <https://www.rfc-editor.org/rfc/rfc8701>.

[RFC9001] Thomson, M., Ed. and S. Turner, Ed., "Using TLS to Secure QUIC", RFC 9001, DOI 10.17487/RFC9001, May 2021, <https://www.rfc-editor.org/rfc/rfc9001>.

Appendix A. Test Vectors

This section includes test vectors for the HTTP authentication scheme specified in this document. It consists of the following types of test vectors:

1. Test vectors for the challenge and redemption protocols. Implementations can use these test vectors for verifying code that builds and encodes TokenChallenge structures, as well as code that produces a well-formed Token bound to a TokenChallenge.

2. Test vectors for the HTTP headers used for authentication. Implementations can use these test vectors for validating whether they parse HTTP authentication headers correctly to produce TokenChallenge structures and the other associated parameters, such as the token-key and max-age values.

A.1. Challenge and Redemption Structure Test Vectors

This section includes test vectors for the challenge and redemption functionalities described in Section 2.1 and Section 2.2. Each test vector lists the following values:

*token_type: The type of token issuance protocol, a value from Section 5.2. For these test vectors, token_type is 0x0002, corresponding to the issuance protocol in [ISSUANCE].

*issuer_name: The name of the issuer in the TokenChallenge structure, represented as a hexadecimal string.

*redemption_context: The redemption context in the TokenChallenge structure, represented as a hexadecimal string.

- origin_info: The origin info in the TokenChallenge structure, represented as a hexadecimal string.

- nonce: The nonce in the Token structure, represented as a hexadecimal string.

- token_key: The public token-key, encoded based on the corresponding token type, represented as a hexadecimal string.

- token_authenticator_input: The values in the Token structure used to compute the Token authenticator value, represented as a hexadecimal string.

Test vectors are provided for each of the following TokenChallenge configurations:

- TokenChallenge with a single origin and non-empty redemption context

- TokenChallenge with a single origin and empty redemption context

- TokenChallenge with an empty origin and redemption context

- TokenChallenge with an empty origin and non-empty redemption context

- TokenChallenge with a multiple origins and non-empty redemption context

These test vectors are below.

```
token_type: 2
issuer_name: 6973737565722e6578616d706c65
redemption_context:
9d262778b3dc2be365d667b03f9cca99efd049e76eb53a6de37120ca34da373b
origin_info: 6f726967696e2e6578616d706c65
nonce:
86ed4bb9f76ab1107a05a9af4aa4eec84dd02f390f9bf5ef14730e0ee15aa92d
token_key_id:
f861220ad4241ee0e33eb4a486a32f05af05ee33fcfdd1104c665eb827c20621
token_authenticator_input: 000286ed4bb9f76ab1107a05a9af4aa4eec84d
d02f390f9bf5ef14730e0ee15aa92df099ea46d6c892cdbc8513586fa8518a6d6
3f28fe4da6f8ddd2a46a405c14488f861220ad4241ee0e33eb4a486a32f05af05
ee33fcfdd1104c665eb827c20621

token_type: 2
issuer_name: 6973737565722e6578616d706c65
redemption_context:
origin_info: 6f726967696e2e6578616d706c65
nonce:
86ed4bb9f76ab1107a05a9af4aa4eec84dd02f390f9bf5ef14730e0ee15aa92d
token_key_id:
f861220ad4241ee0e33eb4a486a32f05af05ee33fcfdd1104c665eb827c20621
token_authenticator_input: 000286ed4bb9f76ab1107a05a9af4aa4eec84d
d02f390f9bf5ef14730e0ee15aa92d11e15c91a7c2ad02abd66645802373db1d8
23bea80f08d452541fb2b62b5898bf861220ad4241ee0e33eb4a486a32f05af05
ee33fcfdd1104c665eb827c20621

token_type: 2
issuer_name: 6973737565722e6578616d706c65
redemption_context:
origin_info:
nonce:
86ed4bb9f76ab1107a05a9af4aa4eec84dd02f390f9bf5ef14730e0ee15aa92d
token_key_id:
f861220ad4241ee0e33eb4a486a32f05af05ee33fcfdd1104c665eb827c20621
token_authenticator_input: 000286ed4bb9f76ab1107a05a9af4aa4eec84d
d02f390f9bf5ef14730e0ee15aa92db741ec1b6fd05f1e95f8982906aec161289
6d9ca97d53eef94ad3c9fe023f7a4f861220ad4241ee0e33eb4a486a32f05af05
ee33fcfdd1104c665eb827c20621

token_type: 2
issuer_name: 6973737565722e6578616d706c65
redemption_context:
9d262778b3dc2be365d667b03f9cca99efd049e76eb53a6de37120ca34da373b
origin_info:
nonce:
86ed4bb9f76ab1107a05a9af4aa4eec84dd02f390f9bf5ef14730e0ee15aa92d
token_key_id:
f861220ad4241ee0e33eb4a486a32f05af05ee33fcfdd1104c665eb827c20621
```

token_authenticator_input: 000286ed4bb9f76ab1107a05a9af4aa4eec84d
d02f390f9bf5ef14730e0ee15aa92dda5366799e0facc5cf9ea3dfc4a6f57072c
31fff84e7331919ebdb06445b2c50f861220ad4241ee0e33eb4a486a32f05af05
ee33fcfdd1104c665eb827c20621

token_type: 2
issuer_name: 6973737565722e6578616d706c65
redemption_context:
9d262778b3dc2be365d667b03f9cca99efd049e76eb53a6de37120ca34da373b
origin_info:
6f726967696e2e6578616d706c652c6f726967696e322e6578616d706c65
nonce:
86ed4bb9f76ab1107a05a9af4aa4eec84dd02f390f9bf5ef14730e0ee15aa92d
token_key_id:
f861220ad4241ee0e33eb4a486a32f05af05ee33fcfdd1104c665eb827c20621
token_authenticator_input: 000286ed4bb9f76ab1107a05a9af4aa4eec84d
d02f390f9bf5ef14730e0ee15aa92df7eeba1bd7c550a8184bee32ce66e6fb527
17aa67da7e0ca32f4cdca9dec7130f861220ad4241ee0e33eb4a486a32f05af05
ee33fcfdd1104c665eb827c20621

A.2. HTTP Header Test Vectors

This section includes test vectors the contents of the HTTP authentication headers. Each test vector consists of one or more challenges that comprise a WWW-Authenticate header. For each challenge, the token-type, token-key, max-age, and token-challenge parameters are listed. Each challenge also includes an unknown (not specified) parameter that implementations are meant to ignore.

The parameters for each challenge are indexed by their position in the WWW-Authentication challenge list. For example, token-key-0 denotes the token-key parameter for the first challenge in the list, whereas token-key-1 denotes the token-key for the second challenge in the list.

The resulting wire-encoded WWW-Authentication header based on this list of challenges is then listed at the end.

```
token-type-0: 0x0002
token-key-0: 30820152303d06092a864886f70d01010a3030a00d300b060960864
8016503040202a11a301806092a864886f70d010108300b060960864801650304020
2a2030201300382010f003082010a0282010100cb1aed6b6a95f5b1ce013a4cfcab2
5b94b2e64a23034e4250a7eab43c0df3a8c12993af12b111908d4b471bec31d4b6c9
ad9cdda90612a2ee903523e6de5a224d6b02f09e5c374d0cfe01d8f529c500a78a2f
67908fa682b5a2b430c81eaf1af72d7b5e794fc98a3139276879757ce453b526ef9b
f6ceb99979b8423b90f4461a22af37aab0cf5733f7597abe44d31c732db68a181c6c
bbe607d8c0e52e0655fd9996dc584eca0be87afbcd78a337d17b1dba9e828bbd81e2
91317144e7ff89f55619709b096cbb9ea474cead264c2073fe49740c01f00e109106
066983d21e5f83f086e2e823c879cd43cef700d2a352a9babd612d03cad02db134b7
e225a5f0203010001
max-age-0: 10
token-challenge-0: 0002000e6973737565722e6578616d706c65208a3e83a33d9
8005d2f30bef419fa6bf4cd5c6005e36b1285bbb4ccd40fa4b383000e6f726967696
e2e6578616d706c65

WWW-Authenticate: PrivateToken challenge="AAIADmlzc3Vlci5leGFtcGxlII
o-g6M9mABdLzC-9Bn6a_TNXGAF42sShbu0zNQPpLODAA5vcmlnaW4uZXhhbXBsZQ==",
 token-key="MIIBUjA9BgkqhkiG9w0BAQowMKANMAsGCWCGSAFlAwQCAqEaMBgGCSqG
SIb3DQEBCDALBglghkgBZQMEAgKiAwIBMAOCAQ8AMIIBCgKCAQEAyxrta2qV9bHOATpM
_KsluUsuZKIwNOQlCn6rQ8DfOowSmTrxKxEZCNS0cb7DHUtsmtnN2pBhKi7pA1I-beWi
JNawLwnlw3TQz-Adj1KcUAp4ovZ5CPpoK1orQwyB6vGvcte155T8mKMTknaHl1fORTtS
bvm_bOuZl5uEI7kPRGGiKvN6qwz1cz91l6vkTTHHMttooYHGy75gfYwOUuBlX9mZbcWE
7KC-h6-814ozfRex26noKLvYHikTFxROf_ifVWGXCbCWy7nqR0zq0mTCBz_kl0DAHwDh
CRBgZpg9IeX4PwhuLoI8h5zUPO9wDSo1Kpur1hLQPK0C2xNLfiJaXwIDAQAB",unknow
nChallengeAttribute="ignore-me", max-age="10"

token-type-0: 0x0002
token-key-0: 30820152303d06092a864886f70d01010a3030a00d300b060960864
8016503040202a11a301806092a864886f70d010108300b060960864801650304020
2a2030201300382010f003082010a0282010100cb1aed6b6a95f5b1ce013a4cfcab2
5b94b2e64a23034e4250a7eab43c0df3a8c12993af12b111908d4b471bec31d4b6c9
ad9cdda90612a2ee903523e6de5a224d6b02f09e5c374d0cfe01d8f529c500a78a2f
67908fa682b5a2b430c81eaf1af72d7b5e794fc98a3139276879757ce453b526ef9b
f6ceb99979b8423b90f4461a22af37aab0cf5733f7597abe44d31c732db68a181c6c
bbe607d8c0e52e0655fd9996dc584eca0be87afbcd78a337d17b1dba9e828bbd81e2
91317144e7ff89f55619709b096cbb9ea474cead264c2073fe49740c01f00e109106
066983d21e5f83f086e2e823c879cd43cef700d2a352a9babd612d03cad02db134b7
e225a5f0203010001
max-age-0: 10
token-challenge-0: 0002000e6973737565722e6578616d706c65208a3e83a33d9
8005d2f30bef419fa6bf4cd5c6005e36b1285bbb4ccd40fa4b383000e6f726967696
e2e6578616d706c65
token-type-1: 0x0001
token-key-1: ebb1fed338310361c08d0c7576969671296e05e99a17d7926dfc28a
53fabd489fac0f82bca86249a668f3a5bfab374c9
max-age-1: 10
token-challenge-1: 0001000e6973737565722e6578616d706c65208a3e83a33d9
```

```
8005d2f30bef419fa6bf4cd5c6005e36b1285bbb4ccd40fa4b383000e6f726967696
e2e6578616d706c65

WWW-Authenticate: PrivateToken challenge="AAIADmlzc3Vlci5leGFtcGxlII
o-g6M9mABdLzC-9Bn6a_TNXGAF42sShbu0zNQPpLODAA5vcmlnaW4uZXhhbXBsZQ==",
 token-key="MIIBUjA9BgkqhkiG9w0BAQowMKANMAsGCWCGSAFlAwQCAqEaMBgGCSqG
SIb3DQEBCDALBglghkgBZQMEAgKiAwIBMAOCAQ8AMIIBCgKCAQEAyxrta2qV9bHOATpM
_KsluUsuZKIwNOQlCn6rQ8DfOowSmTrxKxEZCNS0cb7DHUtsmtnN2pBhKi7pA1I-beWi
JNawLwnlw3TQz-Adj1KcUAp4ovZ5CPpoK1orQwyB6vGvcte155T8mKMTknaHl1fORTtS
bvm_bOuZl5uEI7kPRGGiKvN6qwz1cz91l6vkTTHHMttooYHGy75gfYwOUuBlX9mZbcWE
7KC-h6-814ozfRex26noKLvYHikTFxROf_ifVWGXCbCWy7nqR0zq0mTCBz_kl0DAHwDh
CRBgZpg9IeX4PwhuLoI8h5zUPO9wDSo1Kpur1hLQPK0C2xNLfiJaXwIDAQAB",unknow
nChallengeAttribute="ignore-me", max-age="10", PrivateToken challeng
e="AAEADmlzc3Vlci5leGFtcGxlIIo-g6M9mABdLzC-9Bn6a_TNXGAF42sShbu0zNQPp
LODAA5vcmlnaW4uZXhhbXBsZQ==", token-key="67H-0zgxA2HAjQx1dpaWcSluBem
aF9eSbfwopT-r1In6wPgryoYkmmaPOlv6s3TJ",unknownChallengeAttribute="ig
nore-me", max-age="10"
```

Authors' Addresses

Tommy Pauly
Apple Inc.
One Apple Park Way
Cupertino, California 95014,
United States of America

Email: tpauly@apple.com

Steven Valdez
Google LLC

Email: svaldez@chromium.org

Christopher A. Wood
Cloudflare

Email: caw@heapingbits.net