Workgroup: Privacy Pass
Internet-Draft:
draft-hendrickson-pp-public-metadata-00
Published: 29 March 2023
Intended Status: Informational
Expires: 30 September 2023
Authors: S. Hendrickson
Google

Public Metadata Issuance

Abstract

This document specifies a Privacy Pass token type that encodes public metadata visible to the Client, Attester, Issuer, and Origin.

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://smhendrickson.github.io/draft-hendrickson-privacypass-public-metadata-issuance/draft-hendrickson-pp-public-metadata.html. Status information for this document may be found at https://datatracker.ietf.org/doc/draft-hendrickson-pp-public-metadata/.

Discussion of this document takes place on the Privacy Pass Working Group 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/smhendrickson/draft-hendrickson-privacypass-public-metadata-issuance.

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 30 September 2023.

Copyright Notice

Copyright (c) 2023 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

1. Introduction

This document specifies a Privacy Pass token type that encodes public metadata visible to the Client, Attester, Issuer, and Origin. This allows deployments to encode a small amount of information visible to all parties participating in the protocol.

2. Motivation

Public metadata enables Privacy Pass deployments that share information between clients, attesters, issuers and origins. In a 0x01 (VOPRF) or 0x02 (Blind RSA) deployment, the only information

available to all parties is the issuer_name. If one wants to differentiate bits of information at the origin, many TokenChallenges must be sent - one for each issuer_name that attests to the bit required.

For example, if a deployment was built that attested to an app's published state in an app store, it requires 1 bit {published, not_published} and can be built with a single issuer. To build an app version attester, we need one issuer_name for each app version, and one challenge per version the origin needs to differentiate on each origin load. For each new app version that requires attesting, a new issuer_name must be deployed. If we build this system with public metadata a single TokenChallenge for a single issuer_name can be used. Deployment specific logic could allow adding a new app version into the metadata once sufficient users are on the new version, ensuring users are private when providing attested metadata bits to the origins.

2.1. Alternatives

To implement this scheme without new cryptographic primitives, one could deploy an issuer signing key per metadata value, and publish each key's bit assignment in Issuer Configuration. This many-key metadata deployment should provide metadata visible to all parties in the same way as the [PBLINDRSA] proposal outlined here, however it has reliability and scalability tradeoffs. Imagine a Privacy Pass deployment using a client cached redemption context where max-age cannot be used to expire signed tokens due to the cache, yet the deployment requires fast token expiration. Handling this requires either deploying one key per expiration period or rotating keys quickly. Many simultaneous deployed keys could be difficult to scale - for example some HSM implementations have fixed per-key costs, slow key generation, and minimum key lifetimes. Quick key rotation creates reliability risk to the system, as a pause or slowdown in key rotation could cause the system to run out of active signing or verification keys. [PBLINDRSA] allows deployments to change metadata sets without publishing new keys, and challenge expiry can be encoded within metadata. It pushes all metadata design into the deployment domain, instead of defining them in issuer configuration.

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.

The following terms are used throughout this document.

* Public Metadata: Information that can be viewed by the Client, Attester, Issuer, and Origin, and cryptographically bound to a token.

4. Notation

The following terms are used throughout this document to describe the protocol operations in this document:

* len(s): the length of a byte string, in bytes.

* concat(x0, ..., xN): Concatenation of byte strings. For example, concat(0x01, 0x0203, 0x040506) = 0x010203040506

* int_to_bytes: Convert a non-negative integer to a byte string. int_to_bytes is implemented as I2OSP as described in Section 4.1 of [RFC8017]. Note that these functions operate on byte strings in big-endian byte order.

5. Issuance Protocol for Publicly Verifiable Tokens

This section describes a variant of the issuance protocol in Section 6 of [PROTOCOL] for producing publicly verifiable tokens including public metadata using cryptography specified in [PBLINDRSA]. In particular, this variant of the issuance protocol works for the RSAPBSSA-SHA384-PSSZERO-Deterministic or RSAPBSSA-SHA384-PSS-Deterministic variant of the blind RSA protocol variants described in Section 6 of [PBLINDRSA].

The public metadata issuance protocol differs from the protocol in Section 6 of [PROTOCOL] in that the issuance and redemption protocols carry metadata provided by the Client and visible to the Attester, Issuer, and Origin. This means Clients can set arbitrary metadata when requesting a token, but specific values of metadata may be rejected by any of Attester, Issuer, or Origin. Similar to a token nonce, metadata is cryptographically bound to a token and cannot be altered.

Clients provide the following as input to the issuance protocol:

* Issuer Request URI: A URI to which token request messages are sent. This can be a URL derived from the "issuer-request-uri" value in the Issuer's directory resource, or it can be another Client-configured URL. The value of this parameter depends on the Client configuration and deployment model. For example, in the 'Split Origin, Attester, Issuer' deployment model, the Issuer Request URI might be 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 5.5.

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

*Public Metadata: metadata, an opaque byte string of length at most 2^{16-1} bytes.

Given this configuration and these inputs, the two messages exchanged in this protocol are described below. The constant Nk is defined as 256 for token type 0xDA7A.

5.1. 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)
challenge_digest = SHA256(challenge)
token_input = concat(0xDA7A, // Token type field is 2 bytes long
                     int_to_bytes(len(metadata), 2),
                     metadata,
                     nonce,
                     challenge_digest,
                     token_key_id)
blinded_msg, blind_inv = Blind(pkI, PrepareIdentity(token_input), metada
```

Where PrepareIdentity is defined in Section 6 of [PBLINDRSA] and Blind is defined in Section 4.2 of [PBLINDRSA]

The Client stores the nonce, challenge_digest, and metadata values locally for use when finalizing the issuance protocol to produce a token (as described in Section 5.3).

The Client then creates a TokenRequest structured as follows:

```
struct {
  uint16_t token_type = 0xDA7A; /* Type Public Metadata Blind RSA (2048-
  uint8_t truncated_token_key_id;
  opaque metadata<1..2^16-1>;
  uint8_t blinded_msg[Nk];
} MetadataTokenRequest;
```

The structure fields are defined as follows:

- "token_type" is a 2-octet integer, which matches the type in the challenge.

- "truncated_token_key_id" is the least significant byte of the token_key_id (Section 5.5) in network byte order (in other words, the last 8 bits of token_key_id).

- "metadata" is the opaque metadata value input to the issuance protocol.

- "blinded_msg" is the Nk-octet request defined above.

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

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

<Bytes containing the TokenRequest>
```

5.2. Issuer-to-Client Response

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

- The TokenRequest contains a supported token_type.

- The TokenRequest.truncated_token_key_id corresponds to the truncated key ID of an Issuer Public Key.

- 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. Otherwise, if the Issuer is willing to produce a token token to the Client for the provided metadata value, the Issuer completes the issuance flow by computing a blinded response as follows:

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

Where BlindSign is defined in Section 4.3 of [PBLINDRSA].

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

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

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

```
:status = 200
content-type = application/private-token-response
content-length = <Length of TokenResponse>

<Bytes containing the TokenResponse>
```

5.3. Finalization

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

```
authenticator = Finalize(pkI, nonce, metadata, blind_sig, blind_inv)
```

Where Finalize function is defined in Section 4.4 of [PBLINDRSA].

If this succeeds, the Client then constructs a Token as described in [AUTHSCHEME] as follows:

```
struct {
  uint16_t token_type = 0xDA7A; /* Type Partially Blind RSA (2048-bit) *
  opaque metadata<1..2^16-1>;
  uint8_t nonce[32];
  uint8_t challenge_digest[32];
  uint8_t token_key_id[32];
  uint8_t authenticator[Nk];
} Token;
```

The Token.nonce value is that which was sampled in Section 5.1 of [PROTOCOL]. If the Finalize function fails, the Client aborts the protocol.

5.4. Token Verification

Verifying a Token requires checking that Token.authenticator is a valid signature over the remainder of the token input using the Augmented Issuer Public Key.

```
pkM = AugmentPublicKey(pkI, Token.metadata)
token_input = concat(0xDA7A, // Token type field is 2 bytes long
                     int_to_bytes(len(Token.metadata), 2),
                     Token.metadata,
                     Token.nonce,
                     Token.challenge_digest,
                     Token.token_key_id)
token_authenticator_input =  concat("msg",
    int_to_bytes(len(Token.metadata), 2),
    Token.metadata,
    token_input)
valid = RSASSA-PSS-VERIFY(pkM,
                          token_authenticator_input,
                          Token.authenticator)
```

Where AugmentPublicKey is defined in Section 4.6 of [PBLINDRSA], and message verification is redefined in Section 4.5 of [PBLINDRSA].

The function RSASSA-PSS-VERIFY is defined in Section 8.1.2 of [RFC8017], using SHA-384 as the Hash function, MGF1 with SHA-384 as the PSS mask generation function (MGF), and a 48-byte salt length (sLen).

5.5. Issuer Configuration

Issuers are configured with Private and Public Key pairs, each denoted skI and pkI, respectively, used to produce tokens. Each key pair SHALL be generated as as specified in FIPS 186-4 [DSS], where n is 2048 bits in length. These key pairs MUST NOT be reused in other protocols. Each key pair MUST comply with all requirements as specified in Section 5.2 of [PBLINDRSA].

The key identifier for a keypair (skI, pkI), denoted token_key_id, is computed as SHA256(encoded_key), where encoded_key is a DER-encoded SubjectPublicKeyInfo (SPKI) object carrying pkI. The SPKI object MUST use the RSASSA-PSS OID [RFC5756], which specifies the hash algorithm and salt size. The salt size MUST match the output size of the hash function associated with the public key and token type.

Since Clients truncate token_key_id in each TokenRequest, Issuers should ensure that the truncated form of new key IDs do not collide with other truncated key IDs in rotation.

6. Security Considerations

TODO Security

7. IANA Considerations

This extends the token type registry defined in Section 8.2.1 of [PROTOCOL] with a new token type of value 0xDA7A:

- *Value: 0xDA7A
- *Name: Partially Blind RSA (2048-bit)
- *Token Structure: As defined in Section 5.1
- *TokenChallenge Structure: As defined in Section 2.1 of [AUTHSCHEME]
- *Publicly Verifiable: Y
- *Public Metadata: Y
- *Private Metadata: N
- *Nk: 256
- *Nid: 32
- *Notes: The RSABSSA-SHA384-PSS-Deterministic and RSABSSA-SHA384-PSSZERO-Deterministic variants are supported

8. References

8.1. Normative References

[AUTHSCHEME] Pauly, T., Valdez, S., and C. A. Wood, "The Privacy Pass HTTP Authentication Scheme", Work in Progress, Internet-Draft, draft-ietf-privacypass-auth-scheme-09, 6 March 2023, <https://datatracker.ietf.org/doc/html/draft-ietf-privacypass-auth-scheme-09>.

[PBLINDRSA] Amjad, G. A., Hendrickson, S., Wood, C. A., and K. W. L. Yeo, "Partially Blind RSA Signatures", Work in Progress, Internet-Draft, draft-amjad-cfrg-partially-blind-rsa-00, 13 March 2023, <https://datatracker.ietf.org/doc/html/draft-amjad-cfrg-partially-blind-rsa-00>.

[PROTOCOL] 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>.

[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>.

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

[RFC8017] Moriarty, K., Ed., Kaliski, B., Jonsson, J., and A. Rusch, "PKCS #1: RSA Cryptography Specifications Version 2.2", RFC 8017, DOI 10.17487/RFC8017, November 2016, <https://www.rfc-editor.org/rfc/rfc8017>.

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

8.2. Informative References

[DSS] "Digital Signature Standard (DSS)", National Institute of Standards and Technology report, DOI 10.6028/nist.fips.186-4, July 2013, <https://doi.org/10.6028/nist.fips.186-4>.

Acknowledgments

TODO acknowledge.

Author's Address

Scott Hendrickson
Google

Email: scott@shendrickson.com