Network Working Group
Internet-Draft
Intended status: Informational
Expires: 22 January 2026

P. Pfeiffenberger
J. Katz
T. Olsauskus-Warren
Google

21 July 2025

Probabilistic Reveal Tokens
draft-pfeiffenberger-prtokens-00

Abstract

Fraud detection often relies on high-entropy signals that can also be used to track users across sites. Probabilistic Reveal Tokens (PRTs) attempt to balance the needs of fraud detection and tracking prevention by sampling at a rate that is too low for scaled cross-site tracking, but sufficient for fraud detection in aggregate scenarios. This document describes the PRT protocol, which allows browsers to reveal sensitive signals (e.g., IP address) on a per-site basis with provable probability p_reveal, while websites can use PRTs to measure traffic quality and update denylists.

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://philippp.github.io/id-template/draft-pfeiffenberger-prtokens.html. Status information for this document may be found at https://datatracker.ietf.org/doc/draft-pfeiffenberger-prtokens/.

Source for this draft and an issue tracker can be found at https://github.com/philippp/id-template.

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 22 January 2026.

Table of Contents

1. Introduction

Cross-site tracking threatens user privacy, yet websites require access to certain signals for legitimate fraud detection and security purposes. IP addresses, in particular, serve as critical signals for identifying and mitigating fraudulent activity, but their universal availability enables large-scale tracking across websites.

Probabilistic Reveal Tokens (PRTs) provide a solution that balances these competing needs. PRTs allow browsers to share sensitive signals with websites at a controlled probability rate that is too low for effective cross-site tracking but sufficient for aggregate fraud detection and analysis.

The PRT protocol enables:

* Browsers to reveal sensitive signals on a per-site basis with provable probability p_reveal

* Websites to measure traffic quality for a sample of entities or combinations of entities

* Users to validate that the reveal rate is as expected after tokens have been spent

* Prevention of scaled cross-site tracking while preserving fraud detection capabilities

PRTs utilize ElGamal encryption [ELGAMAL] [RFC3526] with re-randomization properties to ensure that tokens remain unlinkable beyond the probabilistically-included high-entropy signals. Tokens are unforgeable while eligible to be spent and become refutable after the spending period (epoch) ends through the publication of decryption keys.

This document specifies the PRT protocol, including token generation, transmission, validation, and the associated cryptographic operations. The protocol is designed to provide verifiable privacy properties while enabling legitimate fraud detection use cases.

2. Conventions and Definitions

Key Coordinator: The entity that generates the cryptographic keypair necessary for token encryption and decryption, and is responsible for keeping the secret key material secret while tokens are eligible to be spent. This may be implemented as part of the Issuer.

Epoch: A time period during which tokens are eligible to be spent by a browser. After the epoch ends, the Key Coordinator reveals the key material needed to decrypt and verify the tokens. This key material also allows anyone to generate tokens of this now-revealed epoch, creating deniability for token bearers.

Embargo Period: An additional duration after the epoch has ended, and before keys are released. Without an embargo period, keys minted at the exact end of the epoch are at risk of having been minted with recently released keys.

Issuer: An internet-facing service from which the browser fetches PRTs. The issuer computes an array of N = N_signal + N_NULL tokens (where N_signal tokens contain the signal, and N_NULL tokens do not) so that p_reveal = N_signal/N. The issuer shuffles these tokens and passes them to the browser.

Browser: The client software that fetches tokens from the issuer, re-randomizes them to prevent linkability, and sends the tokens to websites. After the key material is published, the browser helps the user validate the privacy properties of the tokens.

Website: The recipient of tokens from the browser. After the key material is published, websites validate the legitimacy of the tokens and leverage sampled signals.

Signal: The sensitive data (such as an IP address) that is probabilistically included in tokens.

p_reveal: The probability that any given token contains the signal rather than a NULL value.

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.

3. Protocol Overview

The PRT protocol involves four main participants: the Key Coordinator, Issuer, Browser, and Websites. The protocol operates in epochs, with each epoch having a defined start and end time.

During each epoch:

1. The Key Coordinator generates an asymmetric keypair for ElGamal encryption

2. The Issuer generates a plaintext value containing signals with probability p_reveal, signs them with an HMAC key, and encrypts them using the public key to create tokens.

3. Browsers fetch token batches, re-randomize individual tokens, and send them to websites in HTTP headers

4. Websites store received tokens for later decryption

After the epoch ends and the embargo period has expired:

1. The Key Coordinator publishes the private key and HMAC secret

2. All parties can decrypt tokens and validate their contents

3. Users can verify the actual reveal rate matches the expected p_reveal

4. Websites can extract signals for fraud analysis

3.1. Trust Model

The PRT protocol relies on the following trust relationships:

* Websites need to trust that the issuer is writing the correct signal into the token

* The issuer needs to independently establish trust in the signal

* Browsers must trust the issuer to correctly mint tokens for the duration of the epoch (i.e. until validation)

3.2. Protocol Requirements

The protocol satisfies the following requirements:

* Ratio Inspection: Users MUST be able to verify the token reveal rate (p_reveal) after epoch completion.

* Content Inspection: Users MUST be able to verify after-the-fact that the tokens do not contain additional identifying information beyond the expected signal.

* Unlinkability: The website and issuer MUST NOT be able to re-identify the user (e.g., by colluding and matching tokens).

* Robustness: Browsers MUST NOT be able to intentionally reduce their aggregate token reveal rate below p_reveal, and websites MUST NOT be able to increase the likelihood of a browser revealing its signal.

4. Protocol Specification

4.1. Token Scope

PRTs reduce access to re-identifiable information in the absence of other cross-visit identifiers. PRT implementations MUST allocate only one token to each website during a visit, where a visit is defined as a period during which the user expects to be re-identified by a website (e.g., a sequence of navigations or a cookied session).

If this requirement is not met, a website could collect multiple tokens for the same user and the same signal, and increase their chance of recovering the signal beyond p_reveal.

4.2. Epochs and Sequencing

Epochs SHOULD be approximately one day in length, but this is not required by the protocol. In practice, epochs SHOULD be long enough to avoid leakage through user re-identification within a browsing session, and short enough that the signal in the token is still likely to be valid. A minimum epoch length of four hours is RECOMMENDED.

Each epoch MUST end after the following epoch has started. This allows browsers to start using tokens from the new epoch before the current epoch ends, avoiding both a lapse in the availability of valid tokens and a thundering herd problem when new tokens become available.

The Key Coordinator SHOULD wait an additional embargo period after the epoch ends before revealing the keys. This ensures that any tokens sent at the end of an epoch cannot be decrypted shortly afterwards when the epoch changeover occurs. It is RECOMMENDED that the embargo duration be equal to that of the epoch length to ensure that no token can be decrypted sooner than one epoch duration in length regardless of when during an epoch it is sent to the website.

4.3. Token Generation

4.3.1. Cryptographic Parameters

The following cryptographic parameters MUST be used:

* Elliptic Curve: secp256r1 (NIST P-256) as defined in [RFC5114]

* Point Compression: All elliptic curve points MUST use compressed encoding as defined in [SEC1]

* HMAC Algorithm: HMAC-SHA256 as defined in [RFC2104]

* HMAC Secret Length: 32 bytes (256 bits)

* Message Padding: 3 bytes appended for valid curve point encoding

4.3.2. Key Generation

The Key Coordinator generates an asymmetric keypair (pk_e, sk_e) to be rotated every epoch E, with sk_e published to all participants after the epoch has ended. Key generation MUST follow these steps:

1. Generate a random private key x uniformly from [1, n-1] where n is the order of the secp256r1 base point

2. Compute the public key point Y = x * G where G is the secp256r1 generator point

3. The ElGamal public key pk_e consists of (G, Y)

4. The ElGamal private key sk_e is x

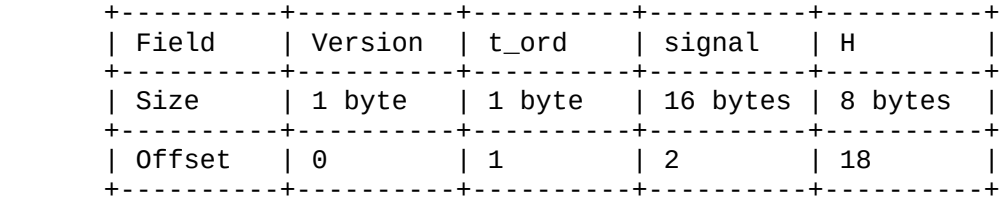
The public key pk_e is shared as a JSON Web Key (JWK) [JWK]:

```
{
  "kty": "EC",
  "crv": "P-256",
  "x": "DROn9TZojl70_6lhtcLxItT2qskNDGk97wjz0N5qdiE",
  "y": "k6EtdGm_jW3b7Le9zM2LgcO7b9Q_qwjS2jL0MFn6V4"
}
```

4.3.3. Token Structure

The Issuer receives pk_e and generates an epoch-scoped secret S_e (32 bytes) that ensures only this issuer can mint tokens.

When a browser requests a batch of N tokens, the Issuer constructs each token message with the following fixed structure:



Field	Version	t_ord	signal	H
Size	1 byte	1 byte	16 bytes	8 bytes
Offset	0	1	2	18

Field definitions:

* Version: MUST be set to 1 for this specification

* t_ord: Sequential ordinal from 1 to N (assigned before shuffling)

* signal: Either 16 zero bytes (NULL) or the actual signal

* H: First 8 bytes of HMAC-SHA256(S_e, Version || t_ord || signal)

4.3.4. Token Generation Algorithm

When a browser requests a batch of N tokens, the Issuer:

1. Observes the signal (e.g., IP address) from the browser’s connection

2. Determines N_signal = floor(N * p_reveal) and N_NULL = N - N_signal

3. For i = 1 to N: a. Set t_ord = i b. If i <= N_signal: set signal to the actual signal value c. If i > N_signal: set signal to a number of zero bytes which matches the signal size d. Construct message M_i as defined in Section 4.3.3 e. Compute HMAC-SHA256(S_e, Version || t_ord || signal) f. Set H to the first 8 bytes of the HMAC result

4. For each message M_i: a. Interpret the 26-byte message as a big-endian integer b. Left-shift the message by three bytes and increment the value of the three-byte padding until the (29-byte) padded message is a valid x-coordinate on the curve. c. Encrypt the point using ElGamal: (u, e) = (r_G, M_point + r_Y) where r is a random scalar d. Serialize u and e using compressed point encoding

5. Shuffle the encrypted tokens with cryptographically secure randomness.

6. Provide the shuffled tokens to the browser along with pk_e, t_epoch_end, and t_next_epoch_start

4.4. Token Re-randomization and Spending

4.4.1. Re-randomization Algorithm

To prevent linkability attacks, the browser MUST re-randomize each token before use. Re-randomization exploits the malleable property of ElGamal encryption to produce a new ciphertext that decrypts to the same plaintext but is unlinkable to the original ciphertext.

Given an ElGamal ciphertext (u, e) where:

* u is a compressed elliptic curve point

* e is a compressed elliptic curve point

The browser re-randomizes by:

1. Parse u and e from their compressed encodings to curve points U and E

2. Generate a random scalar z uniformly from [1, n-1] where n is the order of the secp256r1 curve

3. Compute the re-randomized ciphertext:

    * U′ = U + z * G

    * E′ = E + z * Y

      where G is the generator point and Y is the public key point

4. Serialize U′ and E′ using compressed point encoding

The browser MUST re-randomize each token before it is used and each time the token is re-used to defend against linkability attacks.

The browser maintains a local database with the following schema:

Field Name	u	e	t_epoch_end	epoch_id
Field Type	bytes	bytes	timestamp	bytes

Additional implementation-specific fields for token management:

Field Name	version	public_key	num_signal _tokens	context_id
Field Type	integer	text	integer	string

The context_id field defaults to NULL for new tokens and is set to an identifier of the context (e.g., the top-level domain name) when the token is spent.

When the browser wishes to assert its willingness to probabilistically reveal a signal:

1. It checks for any token already assigned to the requesting context whose t_epoch_end has not passed

2. If such a token exists, the browser reuses it for this connection

3. If no such token exists, the browser assigns a non-spent, current-epoch token to the context by setting the context_id field

4. The assigned token is re-randomized and sent to the relevant party

4.5. Key Disclosure and Validation

At the end of each epoch and after the embargo period has ended, the Key Coordinator MUST publish (pk_e, sk_e, S_e) as JSON Web Keys (JWKs). The ElGamal key is stored with key type "eg", where "x" and "y" hold the public key (pk_e) and "d" holds the secret key (sk_e). The HMAC secret S_e is stored as "hmac.k". All values are converted to big-endian byte arrays and base64url-encoded [RFC4648].

Example key disclosure format:

```
   {
     "epoch_id": 12,
     "epoch_start_time": "20241125T10:00:00",
     "epoch_end_time": "20241126T12:00:00",
     "invalidated_at": null,
     "eg": {
       "kty": "EC",
       "crv": "P-256",
       "x": "DROn9TZojl70_6lhtcLxItT2qskNDGk97wjz0N5qdiE",
       "y": "k6EtdGm_jW3b7Le9zM2LgcO7b9Q_qwjS2jL0MFn6V4",
       "d": "S7_oLScyL_W2ob71hx6kHFv5nTmAt2CvqzmKeF7lLGA"
     },
     "hmac": {
       "kty": "oct",
       "k": "AyM1SysPpbyDfgZld3umj1qzKObwVMkoqQ-EstJQLr_T-1qS0gZH75aKtMN3Yj0iPS4hcgUuTwjA
zZr1Z9CAow",
       "alg": "HS256"
     }
   }
```


Users and websites can then:

1. Decrypt tokens using sk_e

2. Validate HMAC values using S_e

3. Verify that the distribution of t_ord values is uniformly distributed

4. Confirm that the actual reveal rate matches the expected p_reveal

5. HTTP Integration

5.1. HTTP Header Format

PRTs are transmitted in the "Sec-Probabilistic-Reveal-Token" HTTP header [RFC7231]. The header value is a Structured Header Byte Sequence [RFC8941] containing a TLS Presentation Language [RFC8446] serialized PRTStruct:

```
   struct {
     uint8 version;
     uint16 u_length;
     opaque u[u_length];
     uint16 e_length;
     opaque e[e_length];
     opaque epoch_id[8];
   } PRTStruct;
```

Where:

* version identifies the token format version

* u and e are the ElGamal ciphertext components

* epoch_id identifies the corresponding key material for decryption

For version 1, u and e are each 33 bytes in length (compressed secp256r1 points).

5.2. Token Transmission

Example HTTP header:

```
Sec-Probabilistic-Reveal-Token:
:AQAhA0YcSOPXwN8JkGJz2Rxe349sEOzwLcXnrU0/
e5P1QUEEACECjvPnzEReeDlIkrDocZA5ZtiIptiG02YOOaNMJKyKZTdIXbE63QJtYA==:
```

Browsers SHOULD include this header on requests where:

1. The request is being sent through a proxy for privacy purposes

2. The destination origin has registered to receive PRTs

3. A valid PRT is available for the current epoch

If no valid PRTs are available when composing a proxied request, the browser SHOULD make the request without the header. Well-behaved clients SHOULD only fail to attach a PRT in exceptional circumstances (e.g., Issuer unavailability).

6. Implementation Considerations

6.1. Epoch Length

The epoch length affects both privacy and utility:

* Epochs SHOULD be long enough to prevent leakage through user re-identification within browsing sessions

* Epochs SHOULD be short enough that signals remain valid and users can verify issuer behavior in reasonable time

* A minimum epoch length of four hours is RECOMMENDED

* Typical implementations MAY use epoch lengths of approximately one day

6.2. Privacy Considerations

Tokens from the same browser MUST NOT be joinable by the website. If the receiving party can link multiple tokens from the same browser (e.g., through storing them in partitioned storage), the website's chance of recovering the signal increases beyond p_reveal.

Browsers MUST enforce reasonable bounds on the epoch length to ensure privacy requirements are met.

Implementations MUST track context assignments to ensure that only one token per context per epoch is allocated. The context_id field in the browser database is REQUIRED to prevent websites from obtaining multiple tokens and exceeding the intended p_reveal rate.

7. Security Considerations

7.1. Token Ordinal Metadata

Token ordinals prevent a malicious client from choosing one token and using it for every session. This would allow an attacker to generate an unbounded set of tokens with the same message. Although the attacker does not know which property this set of tokens has, this re-use would create a high likelihood that the signal would never be revealed.

For each requested batch of tokens, the issuer records the order of the token in the generation sequence as the token's "ordinal" and then shuffles the tokens. This field does not convey information about the client requesting tokens, since each batch of tokens has the full set of ordinal IDs.

Websites can monitor the distribution of token ordinal values and detect spikes that may indicate an attacker re-randomizing the same token across different sessions.

7.2. Ciphertext Re-randomization

The ability to re-randomize a token's ciphertext without changing the underlying contents (a property of the ElGamal encryption scheme) underpins many of the security and privacy properties of PRTs. The issuer cannot link a re-randomized token's ciphertext back to any tokens it issued, and origins that receive re-randomized versions of the same token cannot link them together.

Before sending a token on a connection to a particular site for the first time, the client re-randomizes the ciphertext for that token to make it unlinkable to any other usage of the token for other pairs. The client caches the ciphertexts for each pairing and reuses them on subsequent requests while the underlying token remains valid.

The client does not re-randomize the token when the token is re-used on the same site. This allows the site to distinguish between a client with many visits using the same token and a client that is re-randomizing the same token across multiple visits (see section above).

7.3. IP Address Mismatch

The IP address a client uses to fetch tokens may differ from the IP address used to connect to websites later. This can occur due to network changes, dynamic IP assignment, or other factors. This drift may result in a PRT revealing an IP address that is unexpected from the user’s perspective.

Browsers can reduce the chance of IP address mismatch by fetching tokens close to when they are spent and aligned with user expectations.

8. IANA Considerations

This document has no IANA actions.

9. References

9.1. 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>.

[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>.

9.2. Informative References

[ELGAMAL] "ELGAMAL", WHATWG Living Standard, n.d., <https://scispace.com/pdf/a-public-key-cryptosystem-and-a-signature-scheme-based-on-1d0i870rfp.pdf>.

[JWK] Jones, M., "JSON Web Key (JWK)", RFC 7517, DOI 10.17487/RFC7517, May 2015, <https://www.rfc-editor.org/rfc/rfc7517>.

[RFC2104] Krawczyk, H., Bellare, M., and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication", RFC 2104, DOI 10.17487/RFC2104, February 1997, <https://www.rfc-editor.org/rfc/rfc2104>.

[RFC3526] Kivinen, T. and M. Kojo, "More Modular Exponential (MODP) Diffie-Hellman groups for Internet Key Exchange (IKE)", RFC 3526, DOI 10.17487/RFC3526, May 2003, <https://www.rfc-editor.org/rfc/rfc3526>.

[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>.

[RFC5114] Lepinski, M. and S. Kent, "Additional Diffie-Hellman Groups for Use with IETF Standards", RFC 5114, DOI 10.17487/RFC5114, January 2008, <https://www.rfc-editor.org/rfc/rfc5114>.

[RFC7231] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", RFC 7231, DOI 10.17487/RFC7231, June 2014, <https://www.rfc-editor.org/rfc/rfc7231>.

[RFC8446] 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>.

[RFC8941] Nottingham, M. and P. Kamp, "Structured Field Values for HTTP", RFC 8941, DOI 10.17487/RFC8941, February 2021, <https://www.rfc-editor.org/rfc/rfc8941>.

[SEC1] "SEC1", WHATWG Living Standard, n.d., <https://www.secg.org/sec1-v2.pdf>.

Acknowledgments

We thank Scott Hendrickson for his thoughtful review and constructive criticism of earlier drafts of this proposal.

Authors' Addresses

Philipp Pfeiffenberger
Google

Email: philippp@gmail.com

Jonathan Katz
Google
Email: jkcrypto@google.com

Theodore Olsauskus-Warren
Google
Email: sauski@google.com