Rate-Limited Issuance
Rate-limiting is a common part of fraud prevention and anonymous access. It also often relies on tracking cookies or client IP addresses. A common way to implement this is with "token buckets".
Token buckets

Status quo rate-limiting

Token Bucket
(are there tokens available?)

Resource Request

Yes, service request!

No, don't service request

Token Replenish

Replenish tokens

Consume tokens

3
Token buckets
Status quo rate-limiting

1. Identity request bucket
2. Increment count by number of tokens

Mapping | Token Count
---------|------------
......   | ...
1250123010339 | N (N + T)
......   | ...

No, don't service request

Yes, service request!

Token Replenish

Resource Request
Token buckets
Status quo rate-limiting

1. Identity request bucket
2. Decrement count associated with bucket
3. Process if non-zero, otherwise discard

<table>
<thead>
<tr>
<th>Mapping</th>
<th>Token Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>....</td>
<td>...</td>
</tr>
<tr>
<td>1250123010339</td>
<td>N (N - 1) &gt; 0?</td>
</tr>
<tr>
<td>....</td>
<td>...</td>
</tr>
</tbody>
</table>

No, don’t service request

Yes, service request!
Why Privacy Pass?

Existing rate-limiting schemes break down when clients have more privacy (*shared rate-limiting buckets*)

- Proxies
- VPNs
- Shared IPs on public networks

A basic Privacy Pass token isn't always enough

- Attests to the fact that a device or user passed some check
- Does not prevent that legitimate device or user doing too many actions (click farm, or abuse of metered paywall)
- Degenerates to blocking access
Private Token variants

Basic Token
(OWRF, RSA Blind Signature)

Attest to user/device legitimacy

Replaces captcha for improving confidence in user

Rate-Limited Token
(RSA Blind Signature)

Attest to user/device legitimacy + access rate below threshold

Adds mitigations against a device in a click farm

Allows metered paywall access
Rate-Limited Tokens

Rate-limited tokens extend the basic issuance protocol with new properties:

1. Attester maintains counters for client + anonymized origin

   • Attesters learn stable mapping between per-client secret and per-origin secret, without learning only per-origin information

2. Issuer provides a rate limit to enforce when issuing tokens

   • Issuers learn origin associated with a token challenge, encrypted with HPKE

3. Attesters fail requests if the per-origin rate limit is exceeded
Rate-Limited Tokens

Rate-limited tokens extend the basic issuance protocol with new properties:

1. Attester maintains counters for client + anonymized origin
   • Attesters learn stable mapping between per-client secret and per-origin secret, without learning only per-origin information

2. Issuer provides a rate limit
   • Issuers learn origin associated with a token challenge, encrypted with HPKE

3. Attesters fail requests if the per-origin rate limit is exceeded

This is the main challenge for the protocol
Attester state

The "token buckets" used for rate limiting now are "private token buckets" maintained on the attester.

A stable mapping is a deterministic function between per-client and per-origin information, e.g., \( F(\text{client secret, origin secret}) \)

The mapping is used to enforce rate limits based on individual clients for individual origins.

Attester uses mapping as index into data structure tracking per-client state.

<table>
<thead>
<tr>
<th>Mapping</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \ldots )</td>
<td>( \ldots )</td>
</tr>
<tr>
<td>12311235123</td>
<td>N</td>
</tr>
<tr>
<td>( \ldots )</td>
<td>( \ldots )</td>
</tr>
</tbody>
</table>
Stable Mappings and Rate Limits

Client

$p_k, p_{k_I}$
challenge
origin
$s_{k_C}$
token

req

resp

Attester

Compute stable mapping, decrement count, compare against origin limit, accept or reject response accordingly

Issuer

req

resp, $I_{origin}$

Drop request
Stable Mappings and Rate Limits

**Client**
- $p_k, p_k$
- $s_k$, $C$
- challenge
- origin
- $s_k$
- token

**Attester**
- Compute stable mapping, decrement count, compare against origin limit, accept or reject response accordingly

<table>
<thead>
<tr>
<th>Mapping</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>....</td>
<td>....</td>
</tr>
<tr>
<td>1234</td>
<td>N -&gt; N-1</td>
</tr>
<tr>
<td>....</td>
<td>...</td>
</tr>
</tbody>
</table>

1234 = $F($client, origin$)$

**Issuer**
- req
- resp, $L_{origin}$

Drop request
Current Status

Builds on signature schemes with key blinding for private computing the stable mapping.

Requires split deployment model for meaningful privacy.

Several interoperable implementations exist and security analysis is underway.
Is the WG interested in adopting this draft?
Backup
Rate-Limited Tokens

Rate-limited tokens extend the basic issuance protocol with new properties:

1. Attester maintains counters for client + anonymized origin
   - Attesters learn **stable mapping** between per-client secret and per-origin secret, without learning only per-origin information

2. Issuer provides a rate limit
   - Issuers learn origin associated with a token challenge, encrypted with HPKE

3. Attesters fail requests if the per-origin rate limit is exceeded

Can we use an OPRF to compute this?...
An OPRF Sketch

An OPRF protocol computes $F(k, x)$ for per-origin $k$ and per-client $x$. 

![Diagram showing the interaction between Client, Issuer, and OPRF]
An OPRF Sketch

Clients can encrypt the origin identifier under the Issuer’s public key

\[ F(k, x) \]

Encrypt(\(pk_I\), origin)
An OPRF Sketch

An Attester can relay the encrypted origin name and complete the OPRF

\[ F(k, x) \]
An OPRF Sketch

... Attester can perform a dictionary attack to learn $F(k, x)$
Rate-Limited Tokens

Rate-limited tokens extend the basic issuance protocol with new properties:

1. Attester maintains counters for client + anonymized origin
   - Attesters learn \textit{stable mapping} between per-client secret and per-origin secret, without learning only per-origin information

2. Issuer provides a rate limit to enforce when issuing tokens
   - Issuers learn origin associated with a token challenge, encrypted with HPKE

3. Attesters fail requests if the per-origin rate limit is exceeded

An OPRF alone isn't sufficient because of dictionary attacks. Computation of the mapping requires proof of ownership for the per-client secret.
Signature Scheme with Key Blinding

Extend digital signature schemes with two functionalities for signing requests

BlindPublicKey and UnblindPublicKey: Given public key and secret blind, produce blinded public key

\[
UnblindPublicKey(\text{BlindPublicKey}(pkS, skB), skB) = pkS
\]

BlindKeySign: Sign message with secret key and secret blind

\[
\text{Verify}(\text{BlindPublicKey}(pkS, skB), msg, \text{BlindKeySign}(skS, skB, msg)) = true
\]

Draft specification: https://datatracker.ietf.org/doc/draft-dew-cfrg-signature-key-blinding/
Detour: Signature Scheme with Key Blinding

Use signature public key blinding to compute an OPRF*

Let $sk_C$ ($sk_O$) be the per-Client (per-Origin) secret with public key $pk_C$ ($pk_O$), and let $sk_R$ be a random client-generated blind per request

$$F(sk_C, sk_O) = \text{Hash}(pk_C, \text{Blind}(pk_C, sk_O))$$

$$= \text{Hash}(pk_C, \text{Unblind}(\text{Blind}(\text{Blind}(pk_C, sk_R), sk_O), sk_R))$$

Close -- but not identical -- to the OPRF construction in draft-irtf-cfrg-voprf

* Security analysis is underway and results will be published in before IETF 114
Rate-Limited Tokens

Client

- Create token request, encrypt origin name, sign package using blinded key pair, send package and blind to attester

- Finalize token request and output token

Attester

- Verify package using blinded public key, forward request to issuer

- Unblind twice-blinded public key, yielding stable mapping used for rate limit check

Issuer

- Verify package using blinded public key, decrypt origin name, re-blind public key using per-origin secret, evaluate token request

Can’t link two requests to same client

Drop request

Function of client secret and origin secret