Architecture

Chris Wood & Jana Iyengar
This client looks suspicious...

GET /gustav.jpg

Client

Server

Challenge
Generate proof based on this challenge

GET /gustav.jpg

Challenge

Proof

Server
GET /gustav.jpg
Challenge
Proof
OK, I trust you
GET /gustav.jpg

Client

Challenge

Proof

Server
Can you solve a CAPTCHA?
Can you solve a CAPTCHA?

Proof that I solved a CAPTCHA
Client

Can you attest to some property?

Proof that I attested to this property

Server
Can you attest to some property?

Proof that I attested to this property

Attest to property

Proof of attestation

Can you attest to some property?

Proof that I attested to this property
Can you attest to some property?

Proof that I attested to this property

Attest to property

Proof of attestation

Trust
Attester

Can you attest to some property?

Proof

Issuer

Proof, please

Proof, please

Attest to property

Proof

Client

Proof

Server
Can you attest to some property?

Proof, please

Attest to property

Proof
Privacy Pass Architecture

Client

Server

Issuer

Trust

Attester

Proof, please

Trust

Attest to property

Proof

Can you attest to some property?

Proof
Deployment Variations

The architecture can be instantiated in various ways

*Combined origin/attester/issuer* ("single verifier")

*Combined attester/issuer*

*All separate*
Architecture describes two parts of the protocol, which are detailed in two separate documents:

**Redemption** is a consistent/unified API for redeeming tokens, along with the ability to challenge.

**Issuance** can support multiple types (VOPRF, publicly verifiable, etc). This is the exchange that can be extended or replaced for new deployment models.
Big Picture

Architecture

Some function *attests* to certain state or properties associated with a client

- Has this person solved a CAPTCHA?
- Does this person have a subscriber account?

Issuers that trust attesters produce proof -- tokens -- bound to these properties

Redeemers, or *origins*, consume tokens from trusted issuers
Why rework the architecture?

Current architecture tightly couples *issuance* and *redemption*

Issuer and redeemer may be the same (as in Privacy Pass) but don’t need to be

Separate roles allow for new deployment models and are more compatible with features like public verifiability

New architecture separates these functions and *shifts extensibility to issuance*

New extensions or features can be solved by new issuance protocols

Redemption is unchanged

Makes *attestation* explicit, but deployment specific
Proposal

Define architecture in terms of functional roles (Client, Origin, Attester, Issuer)

Define protocols in terms of Redemption and Issuance

Merge PR into architecture document

https://github.com/ietf-wg-privacypass/base-drafts/pull/86
Challenge & Redemption

Tommy Pauly
Challenge

Client

"Let me do/access foo"

Origin

"Give me a token to prove you’re not a bot"

Redemption

Client

"Here’s a token, let me do/access foo"

Origin

"Token validated, go ahead"
Challenge & Redemption

All token schemes involve token redemption

Token redemption is when a client presents a token to gain access, anonymously

Challenges are optional

Allows a server to indicate that it needs tokens

Indicates types of tokens and token issuers that are trusted

Allows for interactive tokens
What was missing?

Previous design required Javascript APIs (W3C) work to functionally drive token interactions.

No clear way to support new token types (POPRF vs publicly verifiable, etc).

HTTP authentication method allows a more standard definition.

- Explicit support for different types of tokens, defined in their own contexts.
- Works both in Javascript (W3C) and non-Javascript contexts.

Authors proposing that this work replaces the HTTP API document.
Features

Define an IANA registry of token types, indicate in challenge & redemption

Indicate Issuer name(s) (who does the Origin trust to vend tokens?)

Allow for “interactive tokens” with a one-time nonce to prevent farming

Allow for binding tokens to an origin to prevent cross-origin spending
Origin considerations

Make it easy for origins to adopt!

Origins don’t need to do complex crypto, just need to verify

- Publicly verifiable types are simple (RSA signatures)
- Privately verifiable requires Issuer key (or a single HTTP request to the issuer)

Interactive tokens mitigate concerns about farming and double-spending

- Shifts server state from *redeemed tokens* (unbounded) to *number of outstanding challenges* (bounded by active sessions)
Challenge

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

```c
struct {
    uint16_t token_type; // Defines Issuance protocol
    opaque issuer_name<1..2^16-1>;
    opaque redemption_nonce<0..32>; // Optional
    opaque origin_name<0..2^16-1>; // Optional
} TokenChallenge;
```

Redemption nonce: If present, token presented must be fresh (interactively minted)

Origin name: If present, token is restricted to the origin, else it’s cross-origin
Redemption

Authorization: PrivateToken token=abc...

struct {
    uint16_t token_type; // Matches challenge
    uint8_t nonce[32]; // Client-generated nonce
    uint8_t context[32]; // Hash of TokenChallenge
    uint8_t token_key_id[Nid];
    uint8_t authenticator[Nk]; // From Issuance protocol
} Token;

Context: SHA256 hash of the corresponding challenge

Authenticator: Signature, POPRF output, etc
Redemption Properties

Security properties

**Redemption unlinkability**: Redeemer cannot link two tokens to the same client
Replace HTTP API document with this HTTP auth scheme

HTTP interactions with Issuers go to the Issuance Protocol document

Update W3C APIs to drive this HTTP API
Issuance
Basic Tokens
Issuance protocols

Origin

\( pk_i \)
challenge

Token

Client

Attester

Issuer
\((sk_i, pk_i)\)
Basic Tokens
Issuance protocols

Origin

$pk_s$

$pk_s$

challenge

token

Client

nonce ← $\{0,1\}^{256}$
context = Hash(challenge)
req, inv = Blind($pk_s$, (nonce, context))

Attester

Issuer

$(sk_s, pk_s)$
Basic Tokens
Issuance protocols

**Origin**

\[ p_k_s \]

\[ \text{challenge} \]

\[ \text{token} \]

**Client**

\[ \text{nonce} \leftarrow \{0,1\}^{256} \]

\[ \text{context} = \text{Hash}(\text{challenge}) \]

\[ \text{req, inv} = \text{Blind}(p_k_s, (\text{nonce, context})) \]

**Attester**

\[ \text{req} \]

\[ \text{resp} \]

**Issuer**

\[ (s_k_s, p_k_s) \]

\[ \text{resp} = \text{BlindSign}(s_k_s, \text{req}) \]
Basic Tokens
Issuance protocols

Origin $\rightarrow$ Client $\leftarrow$ Attester $\rightarrow$ Issuer

Client
nonce $\leftarrow \{0,1\}^{256}$
context = Hash(challenge)
req, inv = Blind($pk_s$, (nonce, context))
authenticator = Finalize($pk_s$, (nonce, context), resp, inv)

Attester
req $\rightarrow$ resp $\leftarrow$ BlindSign($sk_s$, req)

Issuer
(resp, $sk_s$, $pk_s$)
Basic Tokens
Issuance protocols

\[
\text{Client} \quad \text{nonce} \leftarrow \{0,1\}^{256} \\
\text{context} = \text{Hash} (\text{challenge}) \\
\text{req, inv} = \text{Blind} (pk_s, (\text{nonce, context})) \\
\text{authenticator} = \\
\text{Finalize} (pk_s, (\text{nonce, context}), \text{resp, inv})
\]

\[
\text{Attester} \quad \text{req} \\
\text{resp}
\]

\[
\text{Issuer} \quad (sk_s, pk_s) \\
\text{resp} = \text{BlindSign} (sk_s, \text{req})
\]

This could also be a POPRF
## Issuance Registry

### Issuance protocols

<table>
<thead>
<tr>
<th>Value</th>
<th>Name</th>
<th>Publicly Verifiable</th>
<th>Public Metadata</th>
<th>Private Metadata</th>
<th>Authenticator Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0001</td>
<td>POPRF(P-384, SHA-384)</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>48</td>
</tr>
<tr>
<td>0x0002</td>
<td>Blind RSA (4096)</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>512</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

**Extensions for consideration:**

- Publicly verifiable anonymous tokens with private metadata bit ([https://eprint.iacr.org/2022/004](https://eprint.iacr.org/2022/004))
Issuance Considerations

Issuance protocols

Issuance protocol is assumed to be stateless on the Issuer

- Blind signature protocols that require multiple rounds and state are possible, but not specified

Compatible with deployment specific key consistency mechanisms

- Issuer keys are discoverable such that applications can build consistency systems on top
Issuance Properties

Security properties

One-more unforgeability: Clients cannot forge tokens

Issuance secrecy: Issuing parties cannot link per-client and per-origin state
Proposal

Replace existing protocol document with new issuance protocol details

Integrates with HTTP-based redemption protocol

Satisfies private and public verifiability (per the charter)

Makes issuance flow in the protocol document explicit and interoperable
Questions for the WG

Wrapping up

1. Are the document proposals clear?
2. Is there consensus in this new direction, which includes:
   1. Updates to draft-ietf-privacypass-architecture and draft-ietf-privacypass-protocol
   2. Adoption of draft-pauly-privacypass-auth-scheme
Rate-Limited Issuance
Rate-Limited Tokens

Issuance protocols

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

1. Issuers learn origin associated with a token challenge

2. Attesters learn stable mapping between per-client secret and per-origin secret, and no per-origin information

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

Challenge: How to reveal only the origin to issuer, and only the mapping to attester?
Rate-Limited Tokens

Issuance protocols

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

1. Issuers learn origin associated with a token challenge

2. Attesters learn **stable mapping** between per-client secret and per-origin secret, and no per-origin information

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

Challenge: How to reveal only the origin to issuer, and only the mapping to attester?
A stable mapping is a deterministic function between per-client and per-origin information, e.g., $F($client secret, origin secret$)$

The mapping is used to enforce per-origin limits

Attester uses mapping as index into data structure tracking per-client state
Detour: Stable Mappings
Issuance protocols

Client
\(p_k, p_{k'}\)
challenge
origin
\(s_{k_C}\)
token

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

Issuer

Drop request
Detour: Stable Mappings

Issuance protocols

Client

$pk_c, pk_i$

callenge

origin

$s_k_c$

token

Attester

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

Issuer

Drop request

$1234 = F(client, origin)$

Mapping

Count

\[
\begin{array}{c|c}
\text{Mapping} & \text{Count} \\
\hline
\ldots & \ldots \\
1234 & \uparrow \\
\ldots & \ldots \\
\end{array}
\]

$N - 1 < L_{\text{origin}}$
Detour: An OPRF Sketch

Issuance protocols

An OPRF protocol computes $F(k, x)$ for per-origin $k$ and per-client $x$
Clients can encrypt the origin identifier under the Issuer’s public key.

\[ F(k, x) \]

\[ \text{Encrypt}(pk_I, \text{origin}) \]
Detour: An OPRF Sketch

Issuance protocols

An Attester can relay the encrypted origin name and complete the OPRF
Detour: An OPRF Sketch

Issuance protocols

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

Issuance protocols

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

1. Issuers learn origin associated with a token challenge

2. Attesters learn stable mapping between per-client secret and per-origin secret, and no per-origin information

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

**Challenge 1**: How to reveal only the origin to issuer, and only the mapping to attester?

**Challenge 2**: How to ensure the attester cannot dictionary attack or replay client requests to learn per-origin information?

Proposed solution to both uses the same mechanism!
Rate-Limited Tokens
Issuance protocols

Cryptographic primitives:

• Blind RSA: Token request

• HPKE: Encrypting origin names from Client to Issuer

• EdDSA with key blinding: Signing Client requests and computing stable mappings

This is the interesting piece!
Detour: EdDSA with Key Blinding
Issuance protocols

Extend RFC8032 EdDSA with two functionalities

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

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

BlindKeySign: Sign message with secret key and secret blind

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

**Rate-Limited Tokens**

**Issuance protocols**

**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

Function of client secret and origin secret

Can’t link two requests to same client

Drop request

- Signing package using blinded key pair
- Finalize token request and output token

**Function of client secret and origin secret**