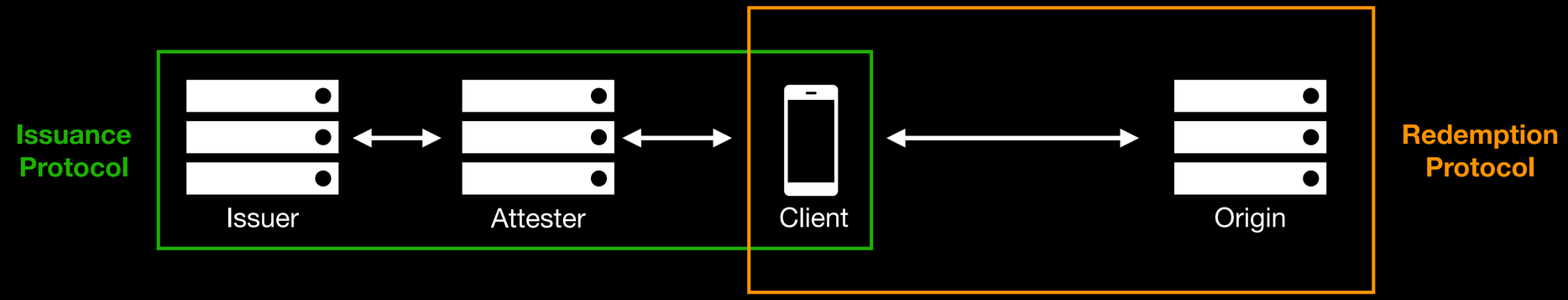
Architecture
draft-ietf-privacypass-architecture

Chris Wood

Protocol Structure

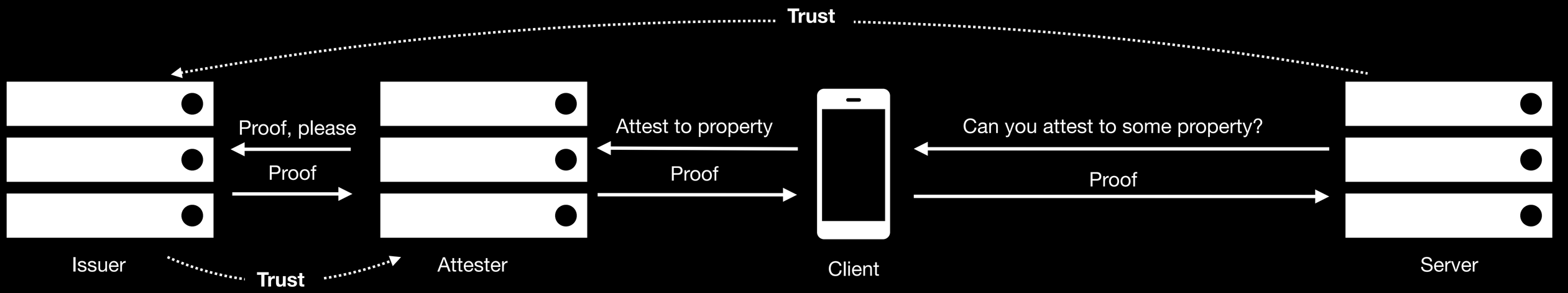


Architecture describes two parts of the protocol, which are detailed in two separate documents:

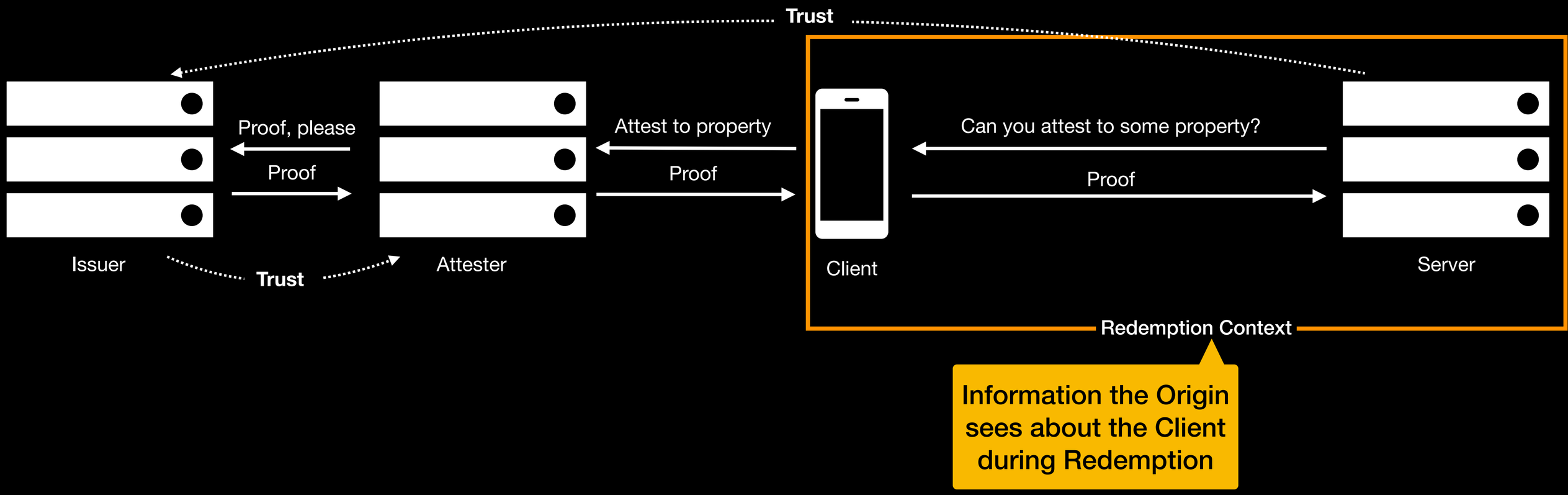
Redemption is a unified protocol for redeeming tokens, along with the ability to challenge.

Issuance can support multiple token types. This is the exchange that can be extended or replaced for new deployment models.

Privacy Contexts

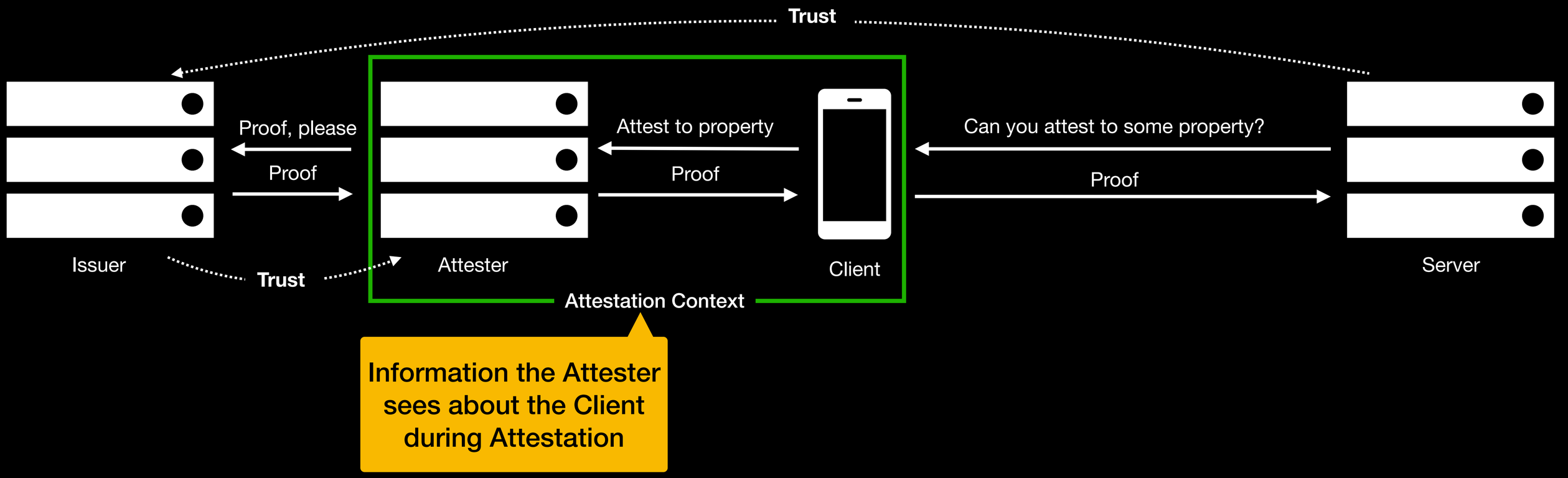


Privacy Contexts

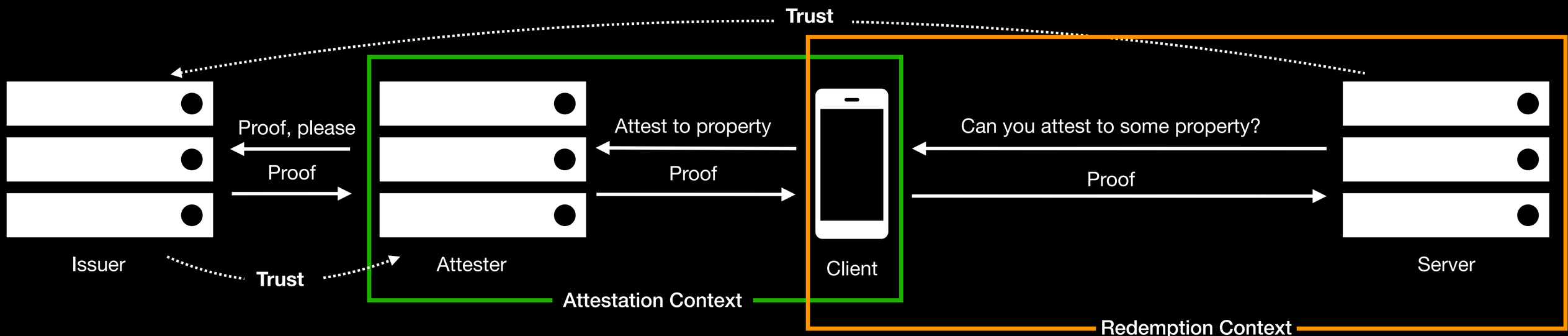


Information the Origin sees about the Client during Redemption

Privacy Contexts



Privacy Contexts

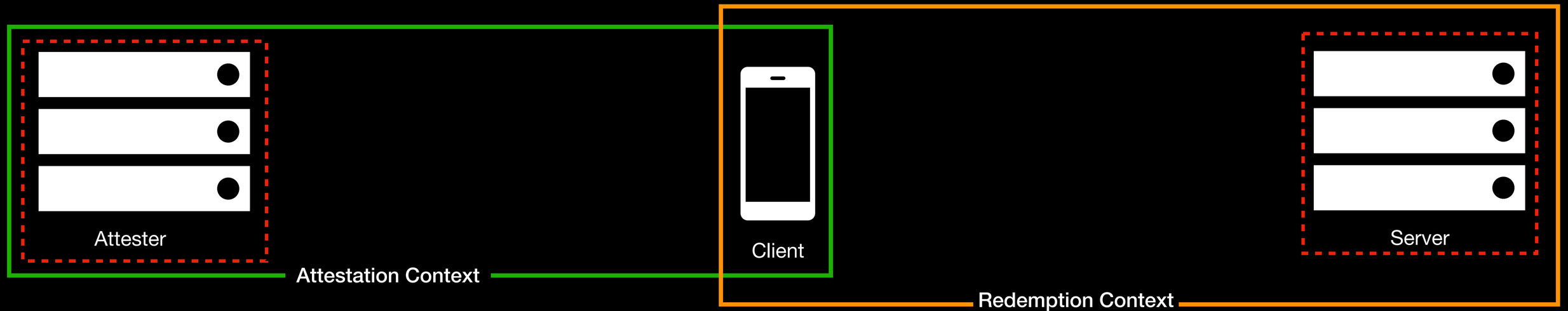


What is meaningful privacy for Privacy Pass?

*Ensure no single entity can link **per-Client** and **per-Server** information*

Joint Deployment

Privacy contexts



Useful deployment model for "privacy-friendly" attestation, such as a CAPTCHA

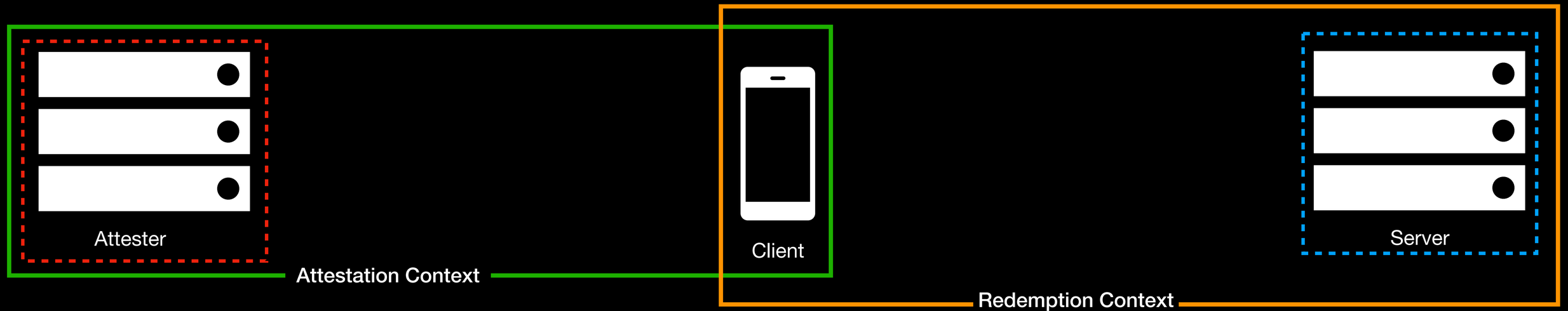
Attester and **Server** are the same entity and share context, including Client IP address, origin name, etc

Meaningful privacy requires context separation ***over time*** or ***over space***

- Time separation: Non-interactive tokens with unlinkable token issuance and redemption
- Space separation: Unlinkable Client identity using a proxy to connect to server

Split Deployment

Privacy contexts



Useful deployment for "privacy-unfriendly" attestation, such as proof of application account ownership

Attester and **Server** are different, non-colluding entities with different contexts

Meaningful privacy requires that Attestation context does not contain per-Server information and that Redemption context does not contain per-Client information

- Attestation constraint: Keep Server (Origin) secret from Attester during issuance (unconditional input secrecy)
- Redemption constraint: Keep Client information (IP address) secret from Server (use of proxy)

Next Steps

Address cross-origin tokens and double spend implications (#104)

Update privacy parameterization (#65)

Address centralization (#45)

WGLC?

Auth Scheme
draft-ietf-privacypass-auth-scheme

Tommy Pauly

Status

Newly adopted!

Minor terminology changes on GitHub

Stabilizing challenge/response format

Terminology updates

Renamed "redemption_nonce" to "redemption_context"

- This really is just a server-chosen context to bind a token to
- Doesn't need to be unique
- Doesn't require that token issuance is "interactive"
- This is not exposed during issuance

Renamed "context" (in Token struct) to challenge_digest

Stabilizing formats

Several implementations have been testing interop

To encourage deployment testing and experimentation, let's stabilize the format of challenges and responses!

Challenge

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

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

Redemption context: If present, token presented must be tied to the context chosen by the server

Origin name: If present, token is restricted to the origin, otherwise 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 challenge_digest[32]; // Hash of TokenChallenge
    uint8_t token_key_id[Nid];
    uint8_t authenticator[Nk]; // From Issuance protocol
} Token;
```

Nonce: Client-chosen nonce, used during issuance

Challenge Digest: Hash of the corresponding challenge

Authenticator: RSA signature, POPRF output, etc.

Origin Behavior

Choose an Issuer & token type

Choose to be per-origin or cross-origin

- For cross-origin, double-spend prevention is only as good as the coordination between origins and the Issuer
- Per-origin allows double-spend prevention to be isolated to a single origin; also prevents the cache of tokens being take up by some other origin

Choose (optional) context

- Empty-context tokens only require state to enforce double-spend prevention
- Context-based tokens can be tied to client session properties (5-tuple, time window, etc) or other state to let the server prevent double-spending more easily

Context Construction Examples

Context-free

```
redemption_context = nil
```

Deterministic, cross-session context

```
redemption_context = SHA256(Client IP address subnet)
```

Per-session context

```
redemption_context = random_bytes(32)
```

Client Behavior

Manage cached tokens

- Cached token needs to match issuer, origin name (if present), and context (if present)
- Empty-context tokens can always be cached. Issuance batch size can be variable, depends on attestation burden
- Context-based tokens may be cached, or can be generated fresh. Context-based tokens should be cleared when cookie state is cleared, or across cookie boundaries.

Verify origin name (if present)

- Needs to match the origin that issued the challenge
- Origin names aren't necessarily seen on the issuance codepath; this is a contract where origins enforce no cross-origin spending

Next Steps

Does anyone see a need to change the formats?

Continue polishing the document

Continue interop testing and experimentation