Configuration Consistency

draft-ietf-privacypass-key-consistency

draft-group-privacypass-k-check

draft-pw-privacypass-in-band-consistency

IETF 117 - PRIVACYPASS - Consistency Design Team

Problem Statement

- Define a "key consistency" or "configuration consistency" protocol
- Identify at least one major Privacy Pass use case that the protocol aims to support.
- Define a concrete protocol that can be implemented interoperably.
- Describe the consistency assurances achieved by the protocol.

Use Cases

- PrivacyPass
  - Issuer key consistency by client
- OHTTP
  - Configuration consistency

Consistency Definitions

- **Key Identifier** (Key ID): A unique identifier for a key.
- **Key Set**: A set of one or more keys.
- **Client**: An entity that uses a key in a reliant system.
- **Source**: An entity that provides key material for use by clients.
- **Consistency** and **global consistency**
- **In-band** and **out-of-band**

Consistency Definitions

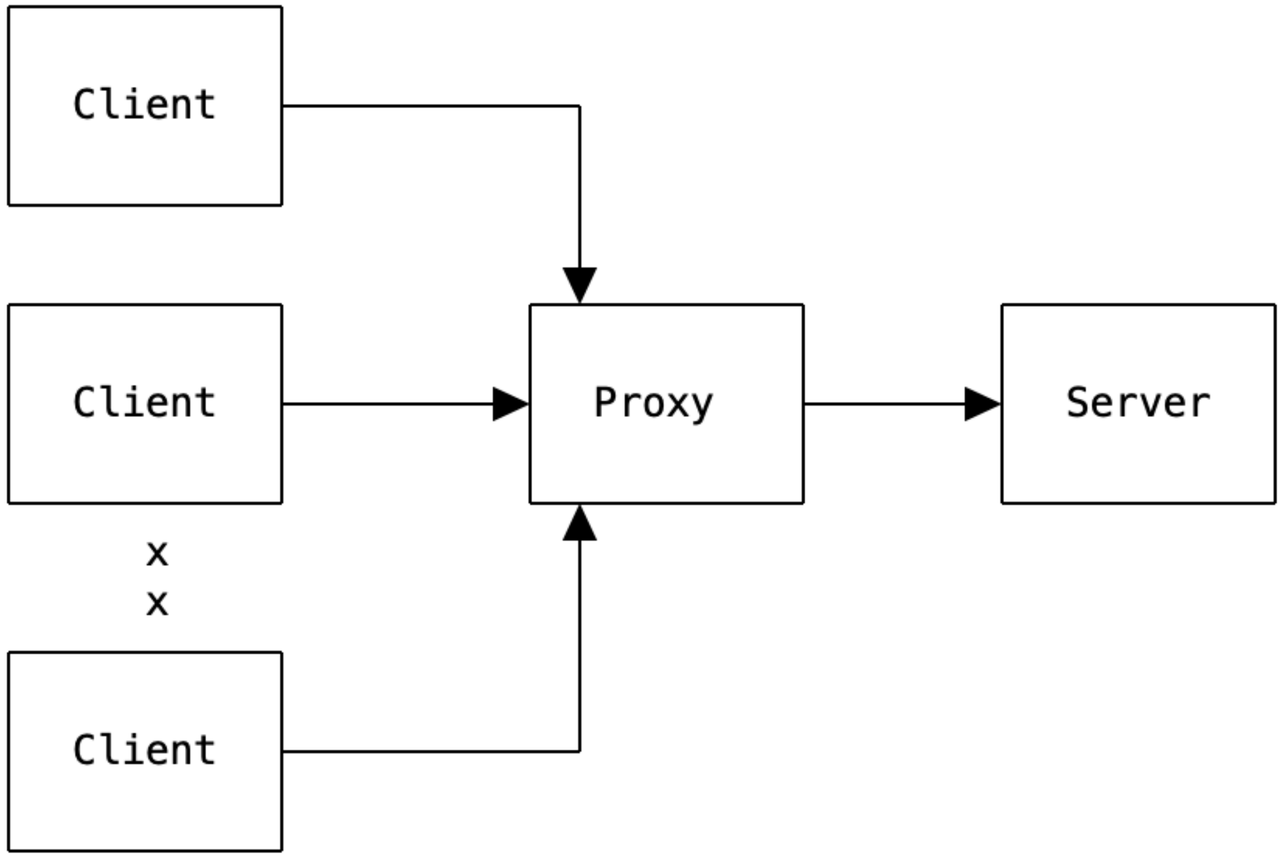
- **Consistency**
  - Two key sets with the same set ID are consistent if and only if (iff) the sets are equal.
- **Global Consistency**
  - A key set X is globally consistent iff, for all key sets Y with the same set ID, the X and Y are consistent.

Consistency Definitions

Checking for consistency or global consistency of **two key sets** (singletons or not) consists in applying a verification function to those sets. If the **two sets** are **consistent** and the **union** of those **two sets** is **equal** to the set of all possible honestly generated values, then the **union is globally consistent**.

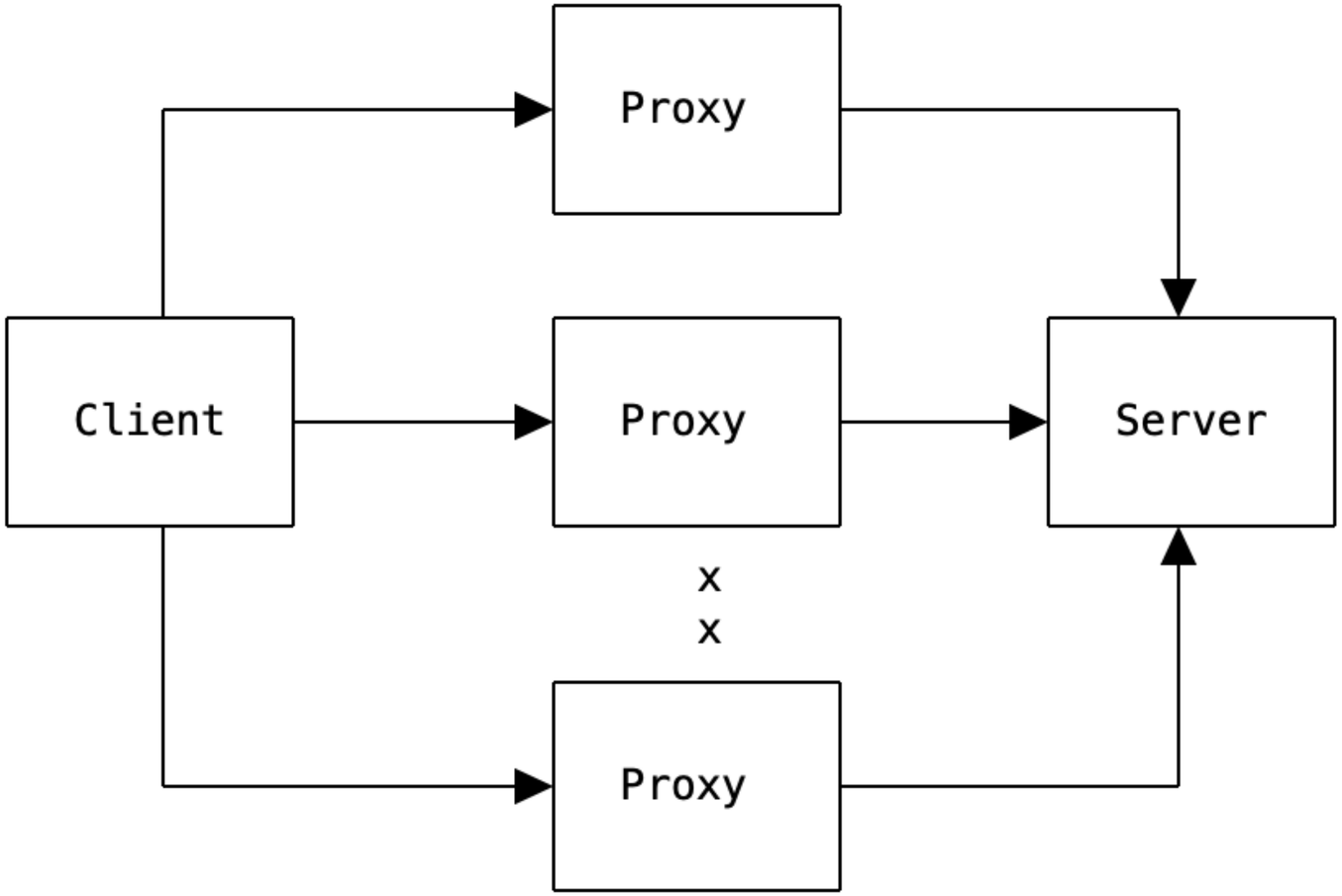
Consistency Mechanism Design Space

Shared Cache



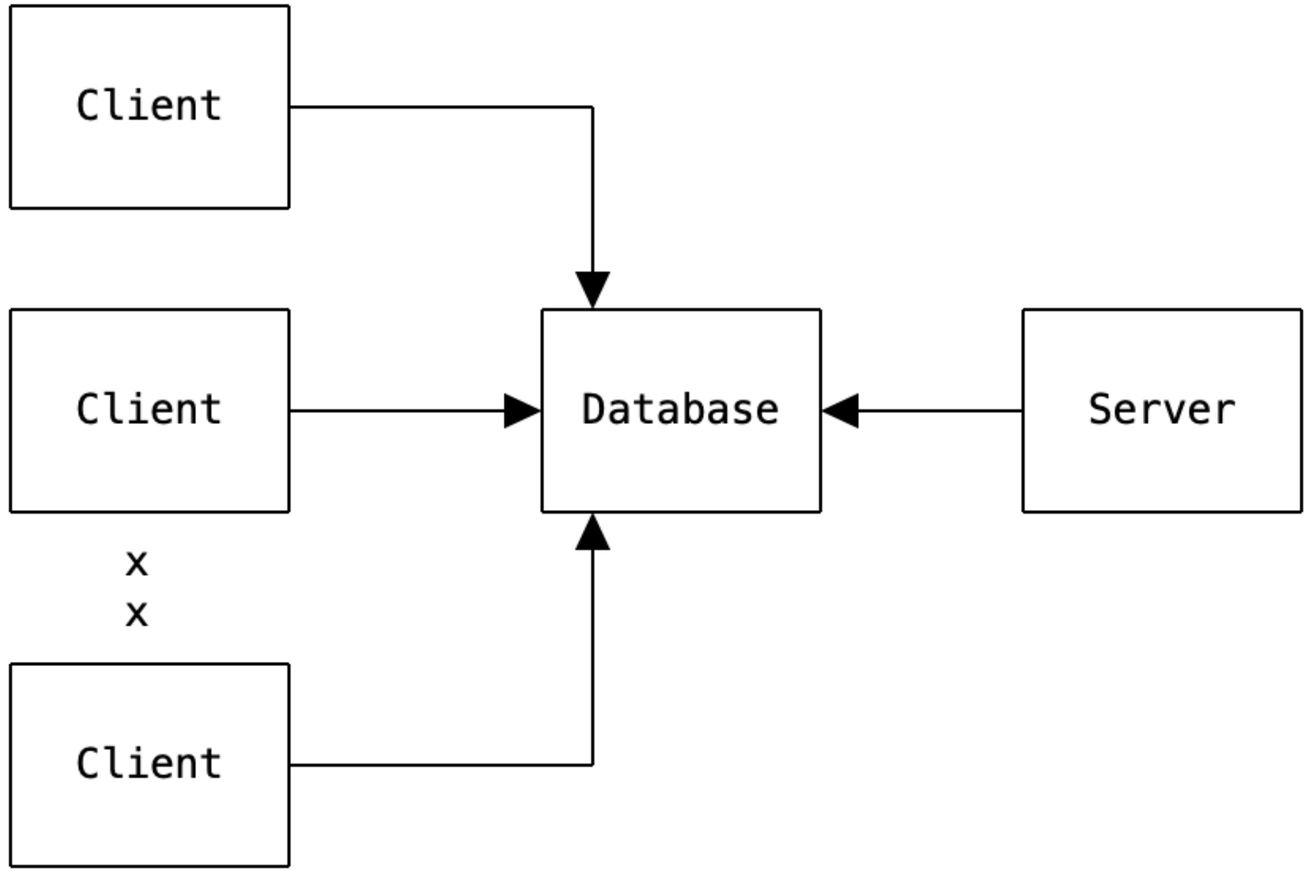
Consistency Mechanism Design Space

Shared Cache with Redundancy



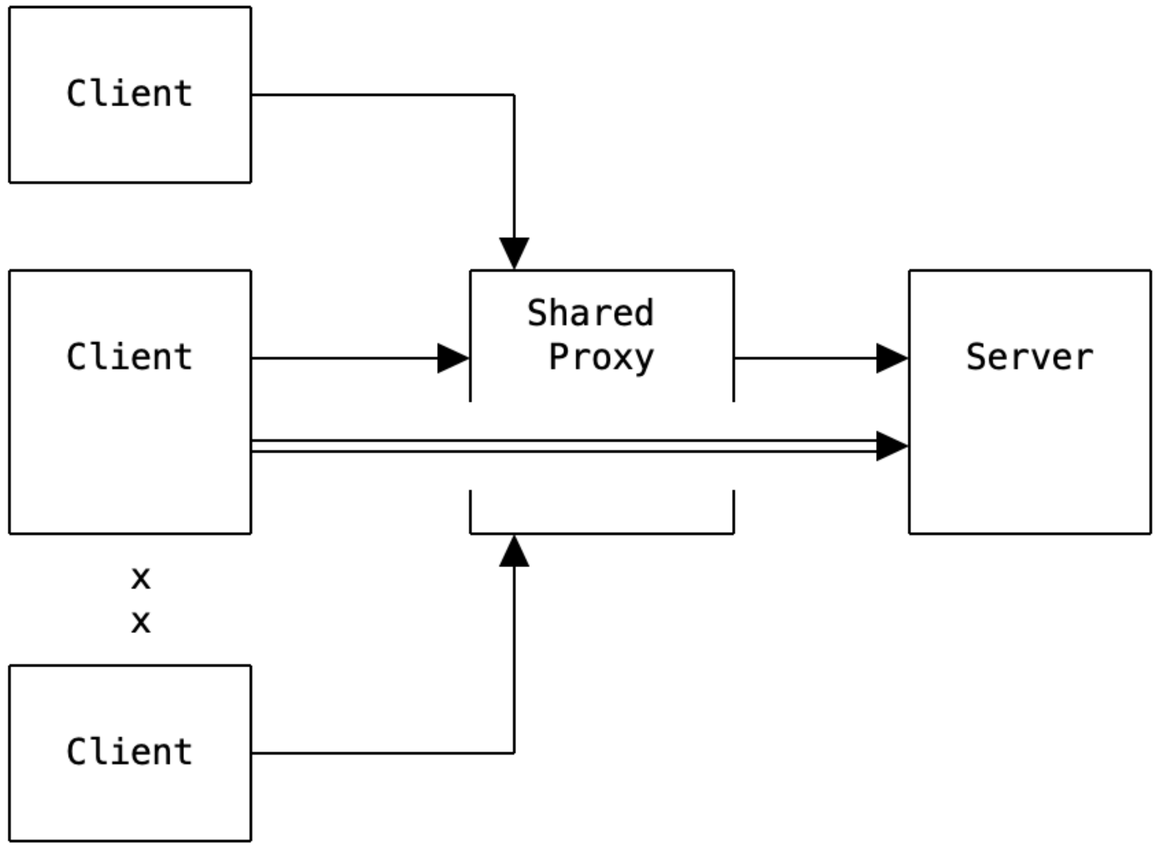
Consistency Mechanism Design Space

Shared Cache with Transparency



Consistency Mechanism Design Space

Shared Cache with Confirmation



In-Band Consistency Check

- Goal: Leverage existing protocol
- Motivation: Simultaneously request tokens and verify consistency
- Only applicable to deployments where the Attester is split from the Origin and Issuer
- Client delegates responsibility to attester
- Open Question: Does this model align with the WG's needs?

K-Check

- Generalizes and extends DoubleCheck
  - HTTP Resource-centric design
- "Out-of-band", (HTTP-based) protocol agnostic, consistency check

Mirror Protocol

- A mirror is similar to a reverse proxy
- The host retrieves, caches, and serves resources
- A mirror is configured with a MIN_VALIDITY_WINDOW parameter
  - an integer indicating the minimum time for resources the mirror will cache according to their "max-age" response directive

Example

- The mirrored resource is identified by a simple URI template
- For example:
    - https://mirror.example/mirror{?target}
    - https://mirror.example/{target}
- Where the *target* value is the percent-encoded URL of a HTTP resource to be mirrored

Mirror Protocol

- Mirror responds with 4xx error if any part of request or validation fail
- Mirror validates that *target* is a valid/allowed resource
- If the resource is already in cache and is still fresh, then respond with:
  - Cache-Control header with appropriate "max-age"
  - The cached response encoded using BHTTP

Mirror Protocol

- If not in cache, then:
  - mirror sends GET to specified *target*
  - validate response
  - cache resource
  - respond with Cache-Control header and BHTTP encoded resource

Mirror Protocol

- Validating response:
  - Persist Vary header
  - The Cache-Control header:
    - is present
    - has a "max-age" response directive that is greater than or equal to MIN_VALIDITY_WINDOW
    - does not have a "no-store" or "private" directive.

K-Check Algorithm

- Mirror discovery is not a primary goal of this protocol
- Client is a priori configured with one or mirror resources
- Inputs to K-Check are:
  - a candidate HTTP resource, *resource*
  - a target URL at which the resource was obtained, *target*
  - a representation of the input resource, *representation*

K-Check Algorithm

- For each configured mirror:
  - Send a mirror request to the mirror for the target URL
    - If request fails, fail this mirror check.
  - Otherwise, compute the first valid, normalized, representation of the resource
  - Compare the computed representation with input *representation*
    - If they do not match, fail this mirror check
    - Otherwise, this mirror check succeeds

K-Check Algorithm

- If all mirror checks succeed, the client outputs success.
- Otherwise, the client has detected an inconsistency and outputs fail.
  - Application-specific behavior on failure

K-Check Algorithm - Normalization

- Application-specific function
  - Structured configurations could require specific fields
- Could use hash or string comparison, by default

K-Check Algorithm - Open Questions

- Can mirrors somehow communicate the number of “active users” to clients?
- How would mirrors determine client uniqueness?
- If mirrors did this accurately, how would clients use this information?

Privacy Pass Example

- Mirror URI Template: "https://mirror.example/mirror{?target}"
- Target URL: https://issuer.example/.well-known/private-token-issuer-directory
- First client sends resource request to mirror, such as:
  - :method = GET
  - :scheme = https
  - :authority = mirror.example
  - :path = /mirror?target=https%3A%2F%2Fissuer.example%2F.well-known%2Fprivate-token-issuer-directory
  - accept = application/private-token-issuer-directory

Privacy Pass Example

- Mirror decodes the *target* parameter
  - https://issuer.example/.well-known/private-token-issuer-directory
- Mirror validates that URL is allowed
- Mirror inspects its cache for a copy of the resource
- Resource is not cached
- Mirror constructs a HTTP request to the target URL to fetch the content
- Request includes the Accept header from the client's request, if present

Privacy Pass Example

- Mirror sends resource request to target, such as:
  - :method = GET
  - :scheme = https
  - :authority = target.example
  - :path = /.well-known/private-token-issuer-directory
  - accept = application/private-token-issuer-directory

Privacy Pass Example

- Target response might looks like:
  - :status = 200
  - content-type = application/private-token-issuer-directory
  - content-length = ...
  - cache-control: max-age=3600

  - <Bytes containing a private token issuer directory>

Privacy Pass Example

- Mirror validates Cache-Control max-age value
- Mirror adds response to its cache
- Mirror encodes response using BHTTP
- Mirror response to client might looks like:
  - :status = 200
  - content-type = application/private-token-issuer-directory
  - content-length = ...
  - cache-control: max-age=3600

  - <Bytes containing the target's BHTTP-encoded response>

Privacy Pass Example


```
{
 "issuer-request-uri": "https://issuer.example.net/request",
 "token-keys": [
  {
   "token-type": 2,
   "token-key": "MI...AB",
   "not-before": 1686913811,
  },
  {
   "token-type": 2,
   "token-key": "MI...AQ",
  }
 ]
}
```

Status

Mirror and K-Check implementations available for reference