Key Consistency and Discovery

Abstract

This document describes the consistency requirements of protocols such as Privacy Pass, Oblivious DoH, and Oblivious HTTP for user privacy. It presents definitions for consistency and then surveys mechanisms for providing consistency in varying threat models. It concludes with discussion of open problems in this area.

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://chris-wood.github.io/key-consistency/draft-ietf-privacypass-key-consistency.html. Status information for this document may be found at https://datatracker.ietf.org/doc/draft-ietf-privacypass-key-consistency/.

Source for this draft and an issue tracker can be found at https://github.com/chris-wood/key-consistency.

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 11 January 2024.
1. Introduction

Several proposed privacy-enhancing protocols such as Privacy Pass [PRIVACY-PASS], Oblivious DoH [ODOH], and Oblivious HTTP [OHTTP] require clients to obtain and use a public key for execution. For example, Privacy Pass public keys are used by clients when issuing and redeeming tokens for anonymous authorization. Oblivious DoH and HTTP both use public keys to encrypt messages to a particular server.

Privacy in these systems depends on clients using an authenticated key that many, if not all, other clients use. If a client were to receive a public key that was specific to them, or restricted to a small set of clients, then use of that public key could be used to
learn targeted information about the client. Informally, using the same key is referred to as key consistency. The degree to which clients use consistent keys determines the extent to which use of a particular key can compromise their individual privacy. This document provides definitions for key consistency that captures this concept.

Depending on the type of consistency, the design space for building key consistency solutions can be large. This document surveys several common approaches to solving this problem and describes the consistency properties they purport to provide under various threat models.

The purpose of this document is twofold: (1) provide a foundation upon which technical solutions can be specified and evaluated, and (2) highlight challenges in building and deploying key consistency solutions in practice.

1.1. Requirements

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.

2. Terminology

This document defines the following terms:

**Reliant System**: A system that embeds one or more key consistency systems.

**Key**: A cryptographic object used by a reliant system.

**Key Identifier (Key ID)**: A unique identifier for a key.

**Key Set**: A set of one or more keys.

**Key Set Identifier (Set ID)**: A unique identifier for a key set.

**Client**: An entity that uses a key in a reliant system.

**Source**: An entity that provides key material for use by clients.

The key consistency model is dependent on the implementation and reliant system's threat model.
3. **Consistency Requirements**

Privacy-focused protocols which rely on widely shared public keys typically require keys be consistent. Informally, key consistency is the requirement that all clients who use a source-provided key in some reliant system share the same view of the key. Some protocols depend on large sets of clients with consistent keys for privacy reasons. Specifically, all clients with a consistent key represent an anonymity set wherein each client of the key in that set is indistinguishable from the rest. An attacker that can actively cause inconsistent views of keys can therefore compromise client privacy.

3.1. **Consistency Definitions**

Formally, consistency is a predicate defined based on key sets. Typically, clients try to assess consistency of one key against one or more keys, but there are no restrictions on whether the clients holding those keys are the same.

There are two different predicates for consistency, defined below.

*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.*

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 checks can happen within a reliant system, i.e., as part of the protocol in which consistency is preferred, or out of it, i.e., a separate protocol run alongside the reliant system. We refer to these two paths as in-band and out-of-band verification. In-band verification is a check which is invoked as part of a reliant system. This type of verification is only achieved by participants of the reliant system. In contrast, out-of-band verifiability is a check that happens outside of a reliant system, i.e., by entities that may not be participants of the reliant system. Consistency verification is typically public, meaning that any entity with two key sets can verify (global) consistency without requiring knowledge of a cryptographic secret.

Reliant systems must also consider agility when trying to achieve consistency. A naive solution to ensuring consistent keys is to only use a single, fixed key pair for the entirety of the system. Clients can then embed this key into software or elsewhere as needed, without
any additional mechanics or controls to ensure that other clients have a different key. However, this solution clearly is not viable in practice. If the corresponding key is compromised, the system fails. Rotation must therefore be supported, and in doing so, clients need some mechanism to ensure that newly rotated keys are consistent.

Operationally, servers rotating keys may likely need to accommodate distributed system state-synchronization issues without sacrificing availability. Some systems and protocols may choose to prioritize strong consistency over availability, but this document assumes that availability is preferred to total consistency.

4. Consistency Mechanisms

There are a variety of ways in which reliant systems may build key consistency solutions, with different trade-offs for operational and implementation complexity. In this section, we survey a number of possible solutions. The viability of each varies depending on the applicable threat model, external dependencies, and overall reliant system's requirements.

In each mechanism, the client has as input a candidate key and seeks to determine if it has a (globally) consistent version of the key.

We do not include the fixed public key model from Section 3, as this is likely not a viable solution for systems and protocols in practice. In all scenarios, the server corresponding to the desired key is considered malicious.

4.1. Direct Discovery

In this model, clients would directly query servers for their corresponding key, as shown below.

![Figure 1: Direct Discovery Example](image)

The properties of this mechanism depend on external mechanisms in place to ensure consistency and whether or not the server colludes with the key source. If the server and source collude, both can present unique per-client keys without detection.
4.2. Shared Cache Discovery

In this model, there exists a shared cache that provides keys from servers on behalf of multiple clients, as shown below.

![Diagram of shared cache discovery](image)

**Figure 2: Shared Cache Discovery Example**

The validity window of the cache's response can impact the overall consistency guarantees. In particular, a system needs to ensure that a server cannot rotate its keys too often in order to divide clients into smaller groups based on when keys are acquired. Such considerations are already highlighted within the Privacy Pass ecosystem, more discussion can be found in [PRIVACY-PASS-ARCH]. Setting a minimum validity period limits the ability of a server to rotate keys, but also limits the rate of key rotation.

Querying a cache for its stored copy of a key leaks information to that cache. There are several mitigations for this leak. For example, clients could obtain the contents of a cache and query it locally. Alternatively, clients could remotely query the cache using privacy-preserving queries (e.g., a private information retrieval (PIR) protocol). In the case where the cache is downloaded locally, it should be considered stale and re-fetched periodically. The frequency of such updates can likely be infrequent in practice, as frequent key updates or rotations may affect privacy. Downloading the entire cache works best if there are a small number of entries, as it does not otherwise impose bandwidth costs on each client that may be impractical.
If this cache is trusted, then all clients which request a key from this server are assured they have a consistent view of the server key compared to all other clients of the cache. A malicious cache can introduce the following threats:

* The cache can collude with the server to give per-client keys to clients.

* The cache can give all clients a key owned by the cache, and either collude with the server to use this key or retroactively use this key to compromise client privacy when clients later make use of the key.

Potential mitigations for untrusted caches are described in the following sections.

### 4.2.1. Cache Redundancy

There are several ways the risk of untrusted caches may be mitigated. The first of which is via the use of multiple, non-colluding caches, as shown below.

![Multi-Cache Discovery Example](image)

*Figure 3: Multi-Cache Discovery Example*

This mechanism provides consistency across all clients that share the same set of caches.
4.2.2. Cache Confirmation

If no other caches are available, clients may attempt to confirm the key provided by the cache directly with the server, as shown in the figure below.

![Diagram of Cache Confirmation](Image)

**Figure 4: Shared Proxy with Confirmation Discovery Example**

Ideally, clients confirm with the server via some anonymizing proxy. Examples of proxies include anonymous systems such as Tor. Tor proxies are general purpose and operate at a lower layer, on arbitrary communication flows, and therefore they are oblivious to clients fetching keys. Untrusted proxies that are aware of key fetch requests (Section 4.2) may be used in a similar way. Depending on how clients fetch such keys from servers, it may become more difficult for servers to uniquely target individual clients with unique keys without detection. This is especially true as the number of clients of these anonymity networks increases. However, beyond Tor, there does not exist a special-purpose anonymity network for this purpose.

4.2.3. Cache Transparency

If redundancy is not viable or feasible for a particular deployment, consistency guarantees may also be improved through transparency
systems, i.e., those based on tamper-proof, publicly verifiable data structures. Examples of this type of system are below.

*An append-only, audited log similar to that of Certificate Transparency [RFC6962]. The log is operated and audited in such a way that the contents of the log are consistent for all clients. Any reliant system which depends on this type of KCCS requires the log be audited or clients have some other mechanism for checking their view of the log state (gossiping). However, this type of system does not ensure proactive security against malicious servers unless log participants actively check log proofs. This requirement may impede deployment in practice. Experience with Certificate Transparency shows that most implementations have chosen not to check SignedCertificateTimestamps before using (that is, accepting as valid) a corresponding TLS certificate.

*A consensus-based log whose assertions are created by a coalition of entities that periodically agree on the correct binding of server names and key material. In this model the agreement is achieved via a consensus protocol, but the specific consensus protocol is dependent on the implementation.

4.3. Key Limits

Consistency may also be improved by forcibly limiting the number of keys that an attacker can feasibly use for targeting particular clients. One way to implement this limit is via key-based encryption, which is a procedure where a client encrypt the information that it sends to a server, such as a token or signed object generated with the server keys. This encryption uses a key derived from the key configuration, specifically not including any form of key identifier along with the encrypted information. If key derivation for the encryption uses a pre-image resistant function (like HKDF), the server can only decrypt the information only if it either knows the key configuration or can guess it. As there is no information the server can use to identify which key was used, it is forced to perform trial decryption if it wants to use multiple keys.

These costs are only linear in terms of the number of active keys. This doesn't prevent the use of multiple keys; it only makes their use incrementally more expensive. Adding a nonce with sufficient entropy might be used to force key derivation for every message. Using a time- or memory-hard key derivation function such as [ARGON2] can then be used to increase the cost of trial decryption.

Encrypting this way could provide better latency properties than a separate check.
5. Future Work

The model in Section 4.2.1 seems to be the most lightweight and easy-to-deploy mechanism for ensuring key consistency and correctness. However, it remains unclear if there exists such an anonymity network that can scale to the widespread adoption of and requirements of protocols like Privacy Pass, Oblivious DoH, or Oblivious HTTP. Also, using such a network carries its own set of risks for clients (as described in Section 4.2.1), so in some cases it might be impractical. Existing infrastructure based on technologies like Certificate Transparency or Key Transparency may work, but there is currently no general purpose system for transparency of opaque keys (or other application data).

6. Security Considerations

This document discusses several models that systems might use to implement public key discovery while ensuring key consistency and correctness. It does not make any recommendations for such models as the best model depends on differing operational requirements and threat models.

7. References

7.1. Normative References


7.2. Informative References


[DOUBLECHECK] Schwartz, B. M., "Key Consistency by Double-Checking via a Semi-Trusted Proxy", Work in Progress, Internet-


Authors' Addresses

Alex Davidson
Brave Software
Email: alex.davidson92@gmail.com

Matthew Finkel
The Tor Project
Email: sysrqb@torproject.org

Martin Thomson
Mozilla
Email: mt@lowentropy.net

Christopher A. Wood
Cloudflare
101 Townsend St
San Francisco,
United States of America
Email: caw@heapingbits.net