Workgroup: Network Working Group
Internet-Draft:
draft-ietf-privacypass-key-consistency-00
Published: 24 October 2022
Intended Status: Informational
Expires: 27 April 2023

Authors: A. Davidson M. Finkel M. Thomson
Brave Software The Tor Project Mozilla
C. A. Wood
Cloudflare

Key Consistency and Discovery

Abstract

This document describes the key consistency and correctness requirements of protocols such as Privacy Pass, Oblivious DoH, and Oblivious HTTP for user privacy. It discusses several mechanisms and proposals for enabling user privacy in varying threat models. In 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

Copyright Notice

Table of Contents

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 for validating privately issued tokens for anonymous session resumption. Oblivious DoH and HTTP both use public keys to encrypt messages to a particular server.

User privacy in these systems depends on users receiving a key that many, if not all, other users receive. If a user were to receive a public key that was specific to them, or restricted to a small set of users, then use of that public key could be used to learn targeted information about the user. Users also need to receive the correct public key.

In this document, we elaborate on these core requirements, and survey various system designs that might be used to satisfy them. The purpose of this document is to highlight challenges in building and deploying solutions to this problem.

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:

Key Consistency and Correctness System (KCCS): A mechanism for providing clients with a consistent view of cryptographic key material within a period of time.

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

The KCCS's consistency model is dependent on the implementation and reliant system's threat model.

3. Core Requirements

Privacy-focused protocols which rely on widely shared public keys typically require keys be consistent and correct. Informally, key consistency is the requirement that all users who communicate with an entity share the same view of the key associated with that entity; key correctness is that the key's secret information is controlled by the intended entity and is not known to be available to an external attacker.

Some protocols depend on large sets of users with consistent keys for privacy reasons. Specifically, all users with a consistent key represent an anonymity set wherein each user of the key in that set is indistinguishable from the rest. An attacker that can actively cause inconsistent views of keys can therefore compromise user privacy.

An attacker that can cause a user to use an incorrect key will likely compromise the entire protocol, not just privacy.

Reliant systems must also consider agility when trying to satisfy these requirements. A naive solution to ensuring consistent and correct keys is to only use a single, fixed key pair for the entirety of the system. Users can then embed this key into software or elsewhere as needed, without any additional mechanics or controls to ensure that other users 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, users need some mechanism to ensure that newly rotated keys are consistent and correct.

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 and Correctness at Key Acquisition

There are a variety of ways in which reliant systems may build key consistency and correct systems (KCCS), ranging in operational complexity to ease-of-implementation. 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.

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, users would directly query servers for their corresponding public key, as shown below.

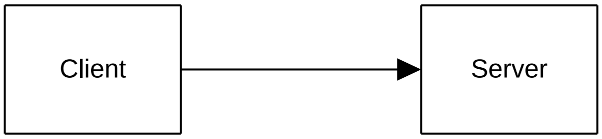

Figure 1: Direct Discovery Example

The properties of this solution depend on external mechanisms in place to ensure consistency or correctness. Absent any such mechanisms, servers can produce unique keys for users without detection. External mechanisms to ensure consistency here might include, though are not limited to:

- Presenting a signed assertion from a trusted entity that the key is correct.

- Presenting proof that the key is present in some tamper-proof log, similar to Certificate Transparency ([RFC6962]) logs.

- User communication or gossip ensuring that all users have a shared view of the key.

The precise external mechanism used here depends largely on the threat model. If there is a trusted external log for keys, this may be a viable solution.

4.2. Trusted Proxy Discovery

In this model, there exists a trusted proxy that fetches keys from servers on behalf of multiple users, as shown below.

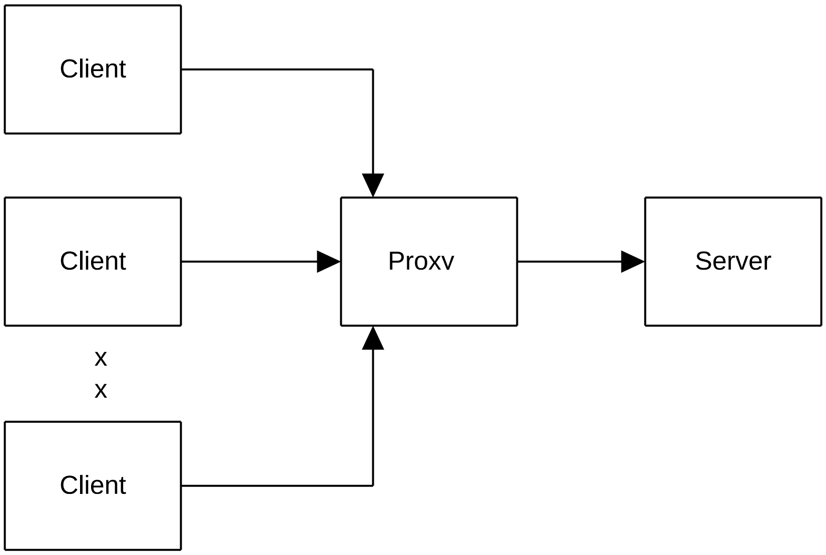

Figure 2: Single Proxy Discovery Example

If this proxy is trusted, then all users which request a key from this server are assured they have a consistent view of the server key. However, if this proxy is not trusted, operational risks may arise:

- The proxy can collude with the server to give per-user keys to clients.

- The proxy can give all users a key owned by the proxy, and either collude with the server to use this key or retroactively use this key to compromise user privacy when users later make use of the key.

Mitigating these risks can be done in a variety of ways. For example, clients may demand tamper-proof proof evidence that the key is consistent and correct for the server, using techniques described in Section 4.1. Clients may gossip amongst themselves to determine if they are being served different keys. Alternatively, the clients may attempt to confirm the key provided by the proxy, as described in Section 4.3.

4.3. Shared Proxy with Key Confirmation

Clients that retrieve keys through a single proxy can directly confirm the correctness of this key provided by the proxy by "checking" with the server. One variant of this checking mechanism is described in [DOUBLECHECK]. Briefly, clients connect directly to the server through some proxy (so as to hide their identity) and ask for the key. If this key does not match that provided by the shared proxy, the clients conclude that the key is malicious. This is shown in Figure 3.

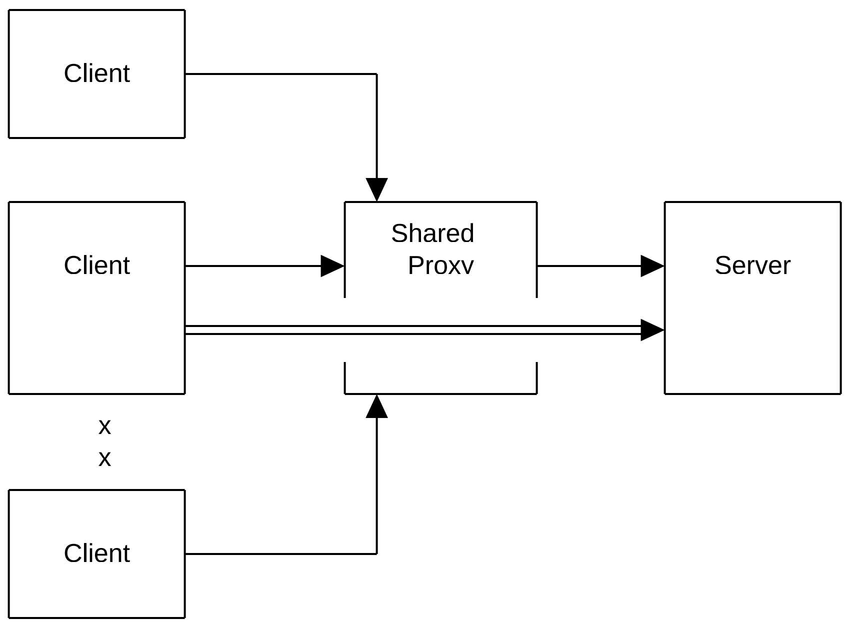

Figure 3: Shared Proxy with Confirmation Discovery Example

4.4. Multi-Proxy Discovery

In this model, users leverage multiple, non-colluding proxies to fetch keys from servers, as shown below.

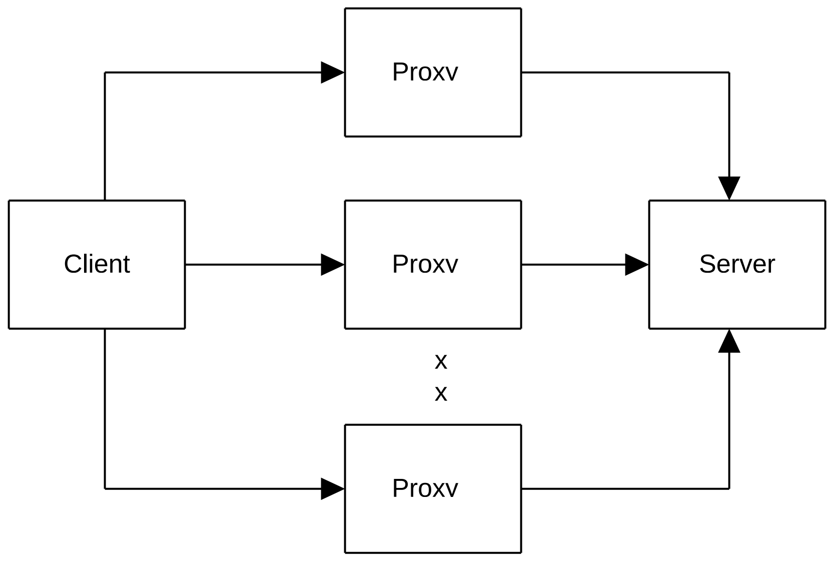

Figure 4: Multi-Proxy Discovery Example

These proxies are ideally spread across multiple vantage points. 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. A large set of 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 users with unique keys without detection. This is especially true as the number of users of these anonymity networks increases. However, beyond Tor, there does not exist a special-purpose anonymity network for this purpose.

Note that connecting to Tor proxies may not be a viable option (indeed, could even be dangerous) for clients operating in managed networks which scrutinize and/or ban Tor traffic.

4.5. Database Discovery

In this model, servers publish keys in an external database and clients fetch keys from the database, as shown below.

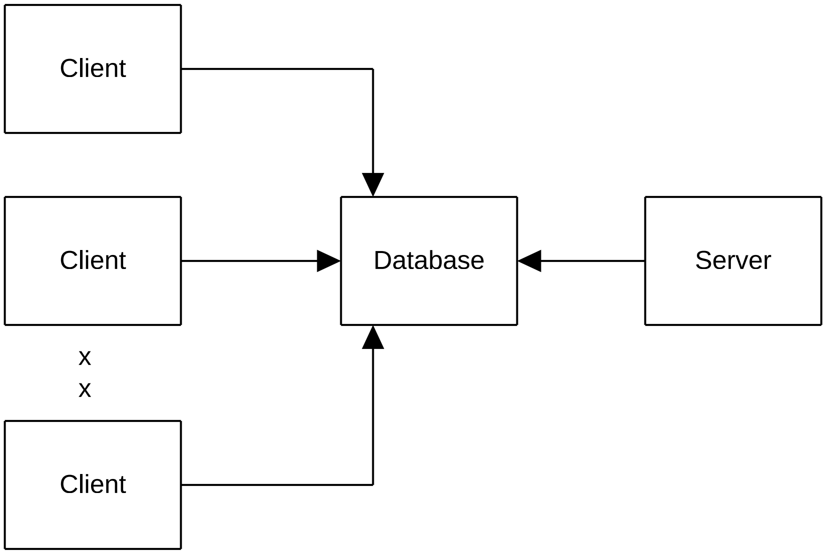

Figure 5: Database Discovery Example

The database is expected to have a table that asserts mappings between server names and keys. Examples of such databases are as follows:

*An append-only, audited table 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 users. Any reliant system which depends on this type of KCCS requires the log be audited or users 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 table 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.

For privacy, users should either download the entire database and query it locally, or remotely query the database using privacy-preserving queries (e.g., a private information retrieval (PIR) protocol). In the case where the database 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; see Section 5 for details. Downloading the entire database 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.

5. Minimum Validity Periods

In addition to ensuring that there is one key at any time, or a limited number keys, any 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 at [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.

6. Separate Consistency Verification

The other schemes described here all attempt to directly limit the number of keys that a client might accept. However, by changing how keys are used, clients can impose costs on servers that might discourage key diversity.

Protocols that have distinctly separate processes for acquiring and using keys might benefit from moving consistency checks to the usage part of the protocol. Correctness might be guaranteed through a relatively simple process, such obtaining keys directly from a server. A separate correctness check is then applied before keys are used.

6.1. Independent Verification

Anonymous queries to verify key consistency can be used prior to use of keys. A request for the current key (or limited set of keys) will reveal if the key that was acquired is different than the original. If the key that was originally obtained is not included, the client can abort any use of the key.

It is important that any validation process not carry any information that might tie it to the original key discovery process or that the system providing verification be trusted. A proxy (see Section 4.2) might be sufficient for providing anonymity, though more robust anonymity protections (see Section 4.4) could provide stronger guarantees. Querying a database (see Section 4.5) might provide independent verification if that database can be trusted not to provide answers that change based on client identity.

6.2. Key-Based Encryption

Key-based encryption has 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 if it knows the key configuration. 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.

7. Future Work

The model in Section 4.4 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 4.4), 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).

8. 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.

9. References

9.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/rfc/rfc2119>.

[RFC6962] Laurie, B., Langley, A., and E. Kasper, "Certificate Transparency", RFC 6962, DOI 10.17487/RFC6962, June 2013, <https://www.rfc-editor.org/rfc/rfc6962>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/rfc/rfc8174>.

9.2. Informative References

[ARGON2] Biryukov, A., Dinu, D., Khovratovich, D., and S. Josefsson, "Argon2 Memory-Hard Function for Password Hashing and Proof-of-Work Applications", Work in Progress, Internet-Draft, draft-irtf-cfrg-argon2-13, 11 March 2021, <https://datatracker.ietf.org/doc/html/draft-irtf-cfrg-argon2-13>.

[DOUBLECHECK] Schwartz, B. M., "Key Consistency by Double-Checking via a Semi-Trusted Proxy", Work in Progress, Internet-Draft, draft-schwartz-ohai-consistency-doublecheck-03, 19

October 2022, <https://datatracker.ietf.org/doc/html/draft-schwartz-ohai-consistency-doublecheck-03>.

[ODOH] Kinnear, E., McManus, P., Pauly, T., Verma, T., and C. A. Wood, "Oblivious DNS over HTTPS", Work in Progress, Internet-Draft, draft-pauly-dprive-oblivious-doh-11, 17 February 2022, <https://datatracker.ietf.org/doc/html/draft-pauly-dprive-oblivious-doh-11>.

[OHTTP] Thomson, M. and C. A. Wood, "Oblivious HTTP", Work in Progress, Internet-Draft, draft-ietf-ohai-ohttp-05, 26 September 2022, <https://datatracker.ietf.org/doc/html/draft-ietf-ohai-ohttp-05>.

[PRIVACY-PASS] Celi, S., Davidson, A., Faz-Hernández, A., Valdez, S., and C. A. Wood, "Privacy Pass Issuance Protocol", Work in Progress, Internet-Draft, draft-ietf-privacypass-protocol-06, 6 July 2022, <https://datatracker.ietf.org/doc/html/draft-ietf-privacypass-protocol-06>.

[PRIVACY-PASS-ARCH] Davidson, A., Iyengar, J., and C. A. Wood, "The Privacy Pass Architecture", Work in Progress, Internet-Draft, draft-ietf-privacypass-architecture-08, 17 October 2022, <https://datatracker.ietf.org/doc/html/draft-ietf-privacypass-architecture-08>.

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