Workgroup: Privacy Preserving Measurement
Internet-Draft: draft-wang-ppm-dap-taskprov-02
Published: 16 November 2022
Intended Status: Informational
Expires: 20 May 2023
Authors: S. Wang   C. Patton
         Apple Inc.   Cloudflare

In-band Task Provisioning for DAP

Abstract

An extension for the Distributed Aggregation Protocol (DAP) is specified that allows the task configuration to be provisioned in-band.

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://wangshan.github.io/draft-wang-ppm-dap-taskprov/draft-wang-ppm-dap-taskprov.html. Status information for this document may be found at https://datatracker.ietf.org/doc/draft-wang-ppm-dap-taskprov/.

Discussion of this document takes place on the Privacy Preserving Measurement Working Group mailing list (mailto:ppm@ietf.org), which is archived at https://mailarchive.ietf.org/arch/browse/ppm/. Subscribe at https://www.ietf.org/mailman/listinfo/ppm/.

Source for this draft and an issue tracker can be found at https://github.com/wangshan/draft-wang-ppm-dap-taskprov.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

The DAP protocol [DAP] enables secure aggregation of a set of reports submitted by Clients. This process is centered around a "task" that determines, among other things, the cryptographic scheme to use for the secure computation (a Verifiable Distributed Aggregation Function [VDAF]), how reports are partitioned into batches, and privacy parameters such as the minimum size of each batch. Before a task can be executed, it is necessary to first provision the Clients, Aggregators, and Collector with the task's configuration.

However, The core DAP specification does not define a mechanism for provisioning tasks. this document describes a mechanism designed to fill this gap. Its key feature is that task configuration is performed completely in-band. It relies solely on the upload channel and the metadata carried by reports themselves.

This method presumes the existence of a logical "task author" (written as "Author" hereafter) who is capable of pushing configurations to Clients. All parameters required by downstream entities (the Aggregators and Collector) are encoded in an extension field of the Client's report. There is no need for out-of-band task orchestration between Leader and Helpers, therefore making adoption of DAP easier.

The extension is designed with the same security and privacy considerations of the core DAP protocol. The Author is not regarded as a trusted third party: It is incumbent on all protocol participants to verify the task configuration disseminated by the Author and opt-out if the parameters are deemed insufficient for privacy. In particular, adopters of this extension should presume the Author is under the adversary's control. In fact, we expect in a real-world deployment that the Author may be implemented by one of the Aggregators or Collector.

Finally, the DAP protocol requires configuring the entities with a variety of assets that are not task-specific, but are important for establishing Client-Aggregator, Collector-Aggregator, and Aggregator-Aggregator relationships. These include:

* The Collector's HPKE [RFC9180] configuration used by the Aggregators to encrypt aggregate shares.

* Any assets required for authenticating HTTP requests.

This specification does not specify a mechanism for provisioning these assets; as in the core DAP protocol, these are presumed to be configured out-of-band.

Note that we consider the VDAF verification key [VDAF], used by the Aggregators to aggregate reports, to be a task-specific asset. This document specifies how to derive this key for a given task from a pre-shared secret, which in turn is presumed to be configured out-of-band.

2. Conventions and Definitions

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.

This document uses the same conventions for error handling as [DAP]. In addition, this document extends the core specification by adding the following error types:

Type	Description
invalidTask	An Aggregator has opted out of the indicated task as described in Section 3.3

Table 1

The terms used follow those described in [DAP]. The following new terms are used:

Task configuration: The non-secret parameters required to create a task in task provision.

Task author: The entity that defines a task's configuration.

3. The "task_prov" Extension

A new extension is defined:

```
enum {
    task_prov(0xff00),
    (65535)
} ExtensionType;
```

When the Client includes this extension with its report, the body of the extension is a TaskConfig defined follows:

```
struct {
    /* Info specific for a task. */
    opaque task_info<1..2^8-1>;

    /* A list of URLs relative to which an Aggregator's API endpoints
    can be found. Defined in I-D.draft-ietf-ppm-dap-02. */
    Url aggregator_endpoints<1..2^16-1>;

    /* This determines the query type for batch selection and the
    properties that all batches for this task must have. */
    QueryConfig query_config;

    /* Time up to which Clients are allowed to upload to this task.
    Defined in I-D.draft-ietf-ppm-dap-02. */
    Time task_expiration;

    /* Determines the VDAF type and its config parameters. */
    VdafConfig vdaf_config;
} TaskConfig;
```

The purpose of TaskConfig is to define all parameters that are necessary for configuring an Aggregator. It includes all the fields to be associated with a task. (See task configuration in [DAP].) In addition to the Aggregator endpoints, maximum batch query count, and task expiration, the structure includes an opaque task_info field that is specific to a deployment. For example, this can be a string describing the purpose of this task.

The query_config field defines the DAP query configuration used to guide batch selection. It is defined as follows:

```
struct {
    QueryType query_type;         /* I-D.draft-ietf-ppm-dap-02 */
    Duration time_precision;      /* I-D.draft-ietf-ppm-dap-02 */
    uint16 max_batch_query_count; /* I-D.draft-ietf-ppm-dap-02 */
    uint32 min_batch_size;
    select (QueryConfig.query_type) {
        case time_interval: Empty;
        case fixed_size:    uint32 max_batch_size;
    }
} QueryConfig;
```

The vdaf_config defines the configuration of the VDAF in use for this task. It is structured as follows (codepoints are as defined in [VDAF]):

```
enum {
    prio3_aes128_count(0x00000000),
    prio3_aes128_sum(0x00000001),
    prio3_aes128_histogram(0x00000002),
    poplar1_aes128(0x00001000),
    (2^32-1)
} VdafType;

struct {
    DpConfig dp_config;
    VdafType vdaf_type;
    select (VdafConfig.vdaf_type) {
        case prio3_aes128_count: Empty;
        case prio3_aes128_sum: uint8; /* bit length of the summand */
        case prio3_aes128_histogram: uint64<8..2^24-8>; /* buckets */
        case poplar1_aes128: uint16; /* bit length of input string */
    }
} VdafConfig;
```

Apart from the VDAF-specific parameters, this structure includes a mechanism for differential privacy (DP). This field, dp_config, is structured as follows:

```
enum {
    reserved(0), /* Reserved for testing purposes */
    none(1),
    (255)
} DpMechanism;

struct {
    DpMechanism dp_mechanism;
    select (DpConfig.dp_mechanism) {
        case none: Empty;
    }
} DpConfig;
```

OPEN ISSUE: Should spell out definition of DpConfig for various differential privacy mechanisms and parameters. See issue #94 for discussion.

The definition of Time, Duration, Url, and QueryType follow those in [DAP].

3.1. Deriving the Task ID

When using the task_prov extension, the task ID is computed as follows:

```
task_id = SHA-256(task_config)
```

where task_config is the TaskConfig structure disseminated by the Author. Function SHA-256() is as defined in [SHS].

3.2. Deriving the VDAF Verification Key

When a Leader and Helper implement the task_prov extension in the context of a particular DAP deployment, they **SHOULD** compute the shared VDAF verification key [VDAF] as described in this section.

The Aggregators are presumed to have securely exchanged a pre-shared secret out-of-band. The length of this secret **MUST** be 32 bytes. Let us denote this secret by verify_key_init.

Let VERIFY_KEY_SIZE denote the length of the verification key for the VDAF indicated by the task configuration. (See [VDAF], Section 5.)

The VDAF verification key used for the task is computed as follows:

```
verify_key = HKDF-Expand(
    HKDF-Extract(
        task_prov_salt,  # salt
        verify_key_init, # IKM
    ),
    task_id,             # info
    VERIFY_KEY_SIZE,     # L
)
```

where task_prov_salt is defined to be the SHA-256 hash of the octet string "dap-taskprov" and task_id is as defined in Section 3.1. Functions HKDF-Extract() and HKDF-Expand() are as defined in [RFC5869]. Both functions are instantiated with SHA-256.

3.3. Configuring a Task

Prior to participating in a task, each protocol participant must determine if the TaskConfig disseminated by the Author can be configured. The participant is said to "opt in" to the task if the derived task ID (see Section 3.1) corresponds to an already configured task or the task ID is unrecognized and therefore corresponds to a new task.

A protocol participant **MAY** "opt out" of a task if:

1. The derived task ID corresponds to an already configured task, but the task configuration disseminated by the Author does not match the existing configuration.

2. The VDAF, DP, or query configuration is deemed insufficient for privacy.

3. A secure connection to one or both of the Aggregator endpoints could not be established.

4. The task lifetime is too long.

A protocol participant **MUST** opt out if the task has expired.

The behavior of each protocol participant is determined by whether or not they opt in to a task.

3.4. Supporting HPKE Configurations Independent of Tasks

In DAP, Clients need to know the HPKE configuration of each Aggregator before sending reports. (See HPKE Configuration Request in [DAP].) However, in a DAP deployment that supports the task_prov extension, if a Client requests the Aggregator's HPKE configuration with the task ID computed as described in Section 3.1, the task ID may not be configured in the Aggregator yet, because the Aggregator is still waiting for the first Client report with the task_prov extension to arrive.

To mitigate this issue, if an Aggregator wants to support the task_prov extension, it **SHOULD** choose which HPKE configuration to advertise to Clients independent of the task ID. It **MAY** continue to support per-task HPKE configurations for other tasks that are configured out-of-band.

In addition, if a Client wants to include the task_prov extension in its report, it **SHOULD NOT** specify the task_id parameter when requesting the HPKE configuration from an Aggregator.

4. Client Behavior

Upon receiving a TaskConfig from the Author, the Client decides whether to opt in to the task as described in Section 3.3. If the Client opts out, it **MUST** not attempt to upload reports for the task.

> OPEN ISSUE: In case of opt-out, would it be useful to specify how to report this to the Author?

Once the client opts in to a task, it **MAY** begin uploading reports for the task. Each report **MUST** offer the task_prov extension with the TaskConfig disseminated by the Author as the extension payload. In addition, the report's task ID **MUST** be computed as described in Section 3.1.

5. Leader Behavior

5.1. Upload Protocol

Upon receiving a Report from the Client with the task_prov extension, if the Leader does not support the extension, it **MUST** ignore the extension payload and proceed as usual. In particular, if the task ID is not recognized, then it **MUST** abort the upload request with "unrecognizedTask".

> NOTE: This behavior assumes unrecognized extensions are to be ignored. See #334 for discussion.

Otherwise, if the Leader does support the extension, it first attempts to parse the payload. If parsing fails, it **MUST** abort with "unrecognizedMessage".

Next, it checks that the task ID included in the report matches the task ID derived from the extension payload as specified in Section 3.1. If the task ID does not match, then the Leader **MUST** abort with "unrecognizedTask".

The Leader then decides whether to opt in to the task as described in Section 3.3. If it opts out, it **MUST** abort the upload request with "invalidTask".

> OPEN ISSUE: In case of opt-out, would it be useful to specify how to report this to the Author?

Finally, once the Leader has opted in to the task, it completes the upload request as usual.

5.2. Aggregate Protocol

When the Leader opts in to a task, it derives the VDAF verification key for that task as described in Section 3.2.

5.3. Collect Protocol

The Collector might issue a collect request for a task provisioned by the task_prov extension prior to opting in to the task. In this case, the Leader would need to abort the collect request with "unrecognizedTask". When it does so, it **SHOULD** also include a "Retry-After" header in its HTTP response indicating the time after which the Collector should retry its request.

> TODO: Find RFC reference for "Retry-After".

> OPEN ISSUE: This semantics is awkward, as there's no way for the Leader to distinguish between Collectors who support the

extension and those that don't. We should consider adding an extension field to CollectReq.

6. Helper Behavior

Upon receiving of an AggregateInitializeReq from the Leader, If the Helper does not support the task_prov extension, it **MUST** ignore the extension payload and process each ReportShare as usual. In particular, if the Helper does not recognize the task ID, it **MUST** abort the aggregate request with error "unrecognizedTask". Otherwise, if the Helper supports the extension, it proceeds as follows.

First, the Helper checks that all report shares carried by the request pertain to the same task. In particular, it checks that:

1. Either all report shares have the task_prov extension or none do. If not the Helper **MUST** abort with "unrecognizedMessage".

2. All report shares with the task_prov extension have the same extension payload. If not, the Helper **MUST** abort with "unrecognizedMessage".

OPEN ISSUE: This awkward input validation step could be skipped if AggregateInitializeReq had an extension field that we could stick the task configuration in. This would also save significantly in overhead.

Next, the Helper attempts to parse the extension payload. If parsing fails, it **MUST** abort with "unrecognizedMessage".

Next, the Helper checks that the task ID included in the message matches the task ID derived from the extension payload as defined in Section 3.1. If not, the Helper **MUST** abort with "unrecognizedTask".

Next, the Helper decides whether to opt in to the task as described in Section 3.3. If it opts out, it **MUST** abort the aggregate request with "invalidTask".

OPEN ISSUE: In case of opt-out, would it be useful to specify how to report this to the Author?

Finally, the Helper completes the aggregate initialize request as usual, deriving the VDAF verification key for the task as described in Section 3.2.

7. Collector Behavior

Upon receiving a TaskConfig from the Author, the Collector first decides whether to opt in to the task as described in Section 3.3.

If the Collector opts out, it **MUST** not attempt to upload reports for the task.

Otherwise, once opted in, the Collector **MAY** begin to issue collect requests for the task. The task ID for each request **MUST** be derived from the TaskConfig as described in Section 3.3.

If the Leader responds to a collect request with an "unrecognizedTask" error, but the HTTP response includes a "Retry-After" header, the Collector **SHOULD** retry its collect request after waiting for the duration indicated by the header.

8. Security Considerations

This document has the same security and privacy considerations as the core DAP specification. In particular, for privacy we consider the Author to be under control of the adversary. It is therefore incumbent on protocol participants to verify the privacy parameters of a task before opting in.

In addition, the task_prov extension is designed to maintain robustness even when the Author misbehaves, or is merely misconfigured. In particular, if the Clients and Aggregators have an inconsistent view of the the task configuration, then aggregation of reports will fail. This is guaranteed by the binding of report metadata to encrypted input shares provided by HPKE encryption.

> OPEN ISSUE: What if the Collector and Aggregators don't agree on the task configuration? Decryption should fail.

A malicious coalition of Clients might attempt to pollute an Aggregator's long-term storage by uploading reports for many (thousands or perhaps millions) of distinct tasks. While this does not directly impact tasks used by honest Clients, it does present a Denial-of-Service risk for the Aggregators themselves.

> TODO: Suggest mitigations for this. Perhaps the Aggregators need to keep track of how many tasks in total they are opted in to?

The HKDF [RFC5869] extraction function is used to derive the task ID. An extractor is not required for security. In fact, a collision-resistant hash function would be sufficient for our application. The choice to use an extractor is conservative in that it allows the derived task ID to be treated as pseudorandom whenever the task configuration itself has high min-entropy from the perspective of the adversary. However, whether this is the case depends on the specific threat model. We note that, in the the threat model for the core DAP protocol, the task configuration is known to the attacker and thus has no entropy.

9. IANA Considerations

NOTE(cjpatton) Eventually we'll have IANA considerations (at the very least we'll need to allocate a codepoint) but we can leave this blank for now.

10. Normative References

[DAP] Geoghegan, T., Patton, C., Rescorla, E., and C. A. Wood, "Distributed Aggregation Protocol for Privacy Preserving Measurement", Work in Progress, Internet-Draft, draft-ietf-ppm-dap-02, 22 September 2022, <https://datatracker.ietf.org/doc/html/draft-ietf-ppm-dap-02>.

[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>.

[RFC5869] Krawczyk, H. and P. Eronen, "HMAC-based Extract-and-Expand Key Derivation Function (HKDF)", RFC 5869, DOI 10.17487/RFC5869, May 2010, <https://www.rfc-editor.org/rfc/rfc5869>.

[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>.

[RFC9180] Barnes, R., Bhargavan, K., Lipp, B., and C. Wood, "Hybrid Public Key Encryption", RFC 9180, DOI 10.17487/RFC9180, February 2022, <https://www.rfc-editor.org/rfc/rfc9180>.

[SHS] "Secure Hash Standard", FIPS PUB 180-4 , 4 August 2015.

[VDAF] Barnes, R., Patton, C., and P. Schoppmann, "Verifiable Distributed Aggregation Functions", Work in Progress, Internet-Draft, draft-irtf-cfrg-vdaf-03, 24 August 2022, <https://datatracker.ietf.org/doc/html/draft-irtf-cfrg-vdaf-03>.

Contributors

CP: Unless the order is meaningful, consider alphabetizing these names.

Junye Chen Apple Inc. junyec@apple.com

Suman Ganta Apple Inc. sganta2@apple.com

Gianni Parsa Apple Inc. gianni_parsa@apple.com

Michael Scaria Apple Inc. mscaria@apple.com

Kunal Talwar Apple Inc. ktalwar@apple.com

Christopher A. Wood Cloudflare caw@heapingbits.net

Authors' Addresses

Shan Wang
Apple Inc.

Email: shan_wang@apple.com

Christopher Patton
Cloudflare

Email: chrispatton+ietf@gmail.com