Privacy Preserving Measurement
Internet-Draft
Intended status: Informational
Expires: 9 March 2026

S. Wang
Apple Inc.
C. Patton
Cloudflare

5 September 2025

Task Binding and In-Band Provisioning for DAP
draft-ietf-ppm-dap-taskprov-03

Abstract

An extension for the Distributed Aggregation Protocol (DAP) is specified that cryptographically binds the parameters of a task to the task's execution. In particular, when a client includes this extension with its report, the servers will only aggregate the report if all parties agree on the task parameters. This document also specifies an optional mechanism for in-band task provisioning that builds on the report extension.

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

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction 3
1.1. Change Log 4
2. Conventions and Definitions 5
3. The Taskprov Extension 6
3.1. Task Encoding 7
3.2. VDAF config 8
3.2.1. Prio3Count 9
3.2.2. Prio3Sum 9
3.2.3. Prio3SumVec 9
3.2.4. Prio3Histogram 9
3.2.5. Prio3MultihotCountVec 9
3.2.6. Poplar1 9
3.3. Extensions 9
4. In-band Task Provisioning with the Taskprov Extension 10
4.1. Overview 11
4.2. Task Advertisement 12
4.3. Deriving the VDAF Verification Key 12
4.4. Opting into a Task 12
4.5. Client Behavior 13
4.6. Leader Behavior 14
4.6.1. Upload Protocol 14
4.6.2. Aggregation Protocol 14
4.6.3. Collection Protocol 14
4.7. Helper Behavior 15
4.8. Collector Behavior 15

5. Security Considerations 16
6. Operational Considerations 17
7. IANA Considerations 17
7.1. Report Extension 18
7.2. Registry for Taskprov Extensions 18
7.3. DAP Sub-namespace for DAP 18
7.4. HTTP Field Name Registration 18
8. Extending this Document 19
9. Normative References 19
Contributors 20
Authors' Addresses 20

1. Introduction

(RFC EDITOR: Remove this paragraph.) This draft is maintained in https://github.com/ietf-wg-ppm/draft-ietf-ppm-dap-taskprov.

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. See Section 4.2 of [DAP] for a complete listing.

In order to execute a task securely, it is required that all parties agree on all parameters associated with the task. However, the core DAP specification does not specify a mechanism for accomplishing this. In particular, it is possible that the parties successfully aggregate and collect a batch, but some party does not know the parameters that were enforced.

A desirable property for DAP to guarantee is that successful execution implies agreement on the task parameters. On the other hand, disagreement between a Client and the Aggregators should prevent reports uploaded by that Client from being processed.

Section 3 specifies a report extension (Section 4.5.3 of [DAP]) that endows DAP with this property. First, it specifies an encoding of all task parameters that are relevant to all parties. This excludes cryptographic assets, such as the secret VDAF verification key (Section 5 of [VDAF]) or the public HPKE configurations [RFC9180] of the aggregators or collector. Second, the task ID is computed by hashing the encoded parameters. If a report includes the extension, then each aggregator checks if the task ID was computed properly: if not, it rejects the report. This cryptographic binding of the task to its parameters ensures that the report is only processed if the Client and Aggregator agree on the task parameters.

One reason this task-binding property is desirable is that it makes the process by which parties are provisioned with task parameters more robust. This is because misconfiguration of a party would manifest in a server's telemetry as report rejection. This is preferable to failing silently, as misconfiguration could result in privacy loss.

Section 4 specifies one possible mechanism for provisioning DAP tasks that is built on top of the extension in Section 3. Its chief design goal is to make task configuration completely in-band, via HTTP request headers. Note that this mechanism is an optional feature of this specification; it is not required to implement the DAP report extension in Section 3.

1.1. Change Log

(RFC EDITOR: Remove this section.)

(*) Indicates a change that breaks wire compatibility with the previous draft.

03:

* Handle repeated extensions in the TaskprovExtension field of the TaskConfig as an error.

* Go back to calling the extension "Taskprov". The name "Taskbind" didn't stick.

* Add task enumeration attacks to security considerations.

* Add registration of the "DAP-Taskprov" to IANA considerations.

* Bump draft-ietf-ppm-dap-13 to 16 [DAP].

* Bump draft-irtf-cfrg-vdaf-13 to 15 [VDAF].

02:

* Don't specify a lower limit for vector bounds.

* Update normative references.

* Recommend including the report extension in the public extensions list.

01:

* Add an extension point to the TaskConfig structure and define rules for processing extensions. (*)

* Remove DP mechanisms. (*)

* Add guidelines for extending this document to account for new VDAFs or DAP batch modes. Improve the extension points for these in TaskConfig in order to make this easier. (*)

* Add a salt to the task ID computation. (*)

* Harmonize task lifetime parameters with [DAP] by adding a task start time and replacing the task end time with a task duration. (*)

* Harmonize batch mode parameters with [DAP] by removing the deprecated max_batch_query_count and max_batch_size parameters. (*)

* Task provisioning: Remove guidance for per-task HPKE configurations, as this feature was deprecated by DAP.

* Bump draft-ietf-ppm-dap-12 to 13 [DAP]. (*)

* Bump draft-irtf-cfrg-vdaf-12 to 13 [VDAF].

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 4.4

Table 1

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

Task configuration: The non-secret parameters of a task.

Task author: The entity that defines a task's configuration in the provisioning mechanism of Section 4.

3. The Taskprov Extension

To use the Taskprov extension, the Client includes the following extension in the report extensions for each Aggregator as described in Section 4.5.3 of [DAP]:

```
enum {
    taskprov(0xff00),
    (2^16-1)
} ExtensionType;
```

The payload of the extension MUST be empty. If the payload is non-empty, then the Aggregator MUST reject the report.

When the client uses the Taskprov extension, it computes the task ID (Section 4.2 of [DAP]) as follows:

```
task_id = SHA-256(SHA-256("dap-taskprov task id") || task_config)
```

where task_config is a TaskConfig structure defined in Section 3.1. Function SHA-256() is as defined in [SHS].

The task ID is bound to each report share (via HPKE authenticated and associated data, see Section 4.5.2 of [DAP]). Binding the parameters to the ID this way ensures, in turn, that the report is only aggregated if the Client and Aggregator agree on the parameters. This is accomplished by the Aggregator behavior below.

During aggregation (Section 4.6 of [DAP]), each Aggregator processes a report with the Taskprov extension as follows.

First, it looks up the ID and parameters associated with the task. Note the task has already been configured; otherwise the Aggregator would have already aborted the request due to not recognizing the task.

Next, the Aggregator encodes the parameters as a TaskConfig defined in Section 3.1 and computes the task ID as above. If the derived task ID does not match the task ID of the request, then it MUST reject the report with error "invalid_message".

During the upload interaction (Section 4.5 of [DAP]), the Leader SHOULD abort the request with "unrecognizedTask" if the derived task ID does not match the task ID of the request.

3.1. Task Encoding

The task configuration is encoded as follows:

```
uint32 VdafType; /* As defined in Section 10 of [VDAF] */

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

    /* Leader API endpoint. */
    Url leader_aggregator_endpoint;

    /* Helper API endpoint. */
    Url helper_aggregator_endpoint;

    /* Time precision. */
    Duration time_precision;

    /* Minimum batch size. */
    uint32 min_batch_size;

    /* The batch mode and its parameters. */
    BatchMode batch_mode;
    opaque batch_config<0..2^16-1>;

    /* The earliest timestamp that will be accepted for this task. */
    Time task_start;

    /* The duration of the task. */
    Duration task_duration;

    /* Determines the VDAF type and its config parameters. */
    VdafType vdaf_type;
    opaque vdaf_config<0..2^16-1>;

    /* Taskprov Extensions. */
    TaskprovExtension extensions<0..2^16-1>;
} TaskConfig;
```

The purpose of TaskConfig is to define all parameters that are necessary for configuring each party. It includes all parameters listed in Section 4.2 of [DAP] as well as two additional fields:

* task_info is an opaque field whose contents are specific to the deployment. For example, this might be a human-readable string describing the purpose of this task.

* extensions is a list of extensions to this document. The format and semantics of extensions are describe in Section 3.3.

This structure does not include cryptographic assets shared by only a subset of the parties, including the secret VDAF verification key [VDAF] or public HPKE configurations [RFC9180].

The batch_mode field indicates the DAP batch mode and corresponds to a codepoint in the Batch Modes Registry.

TODO Add a reference to the IANA registry for batch modes created by [DAP].

The batch_config field contains any parameters that are required for configuring the batch mode. For the time-interval and leader-selected batch modes specified in [DAP], the payload is empty. Batch modes defined by future documents may specify a non-empty payload; see Section 8 for details. The length prefix of the batch_config ensures that the batch config can be decoded even if the batch mode is unrecognized.

The vdaf_type field indicates the VDAF for the task and corresponds to a codepoint in the VDAF Identifiers registry.

TODO: Add a reference to the IANA registry created by [VDAF].

The vdaf_config field contains parameters necessary to configure an instance of the VDAF. Section 3.2 defines a suitable encoding of the configuration for each VDAF specified in [VDAF]. VDAFs defined by future documents may also use this field as well; see Section 8 for details.

The length prefix of the vdaf_config ensures that VDAF config can be decoded even if the VDAF type is not recognized.

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

3.2. VDAF config

This section defines the payload of TaskConfig.vdaf_config for each VDAF specified in [VDAF]. In some cases, the VDAF supports more than two Aggregators; but since DAP only supports two Aggregators, we do not include the number of Aggregators in the encoding.

3.2.1. Prio3Count

The payload is empty.

3.2.2. Prio3Sum

```
struct {
    uint32 max_measurement; /* largest summand */
} Prio3SumConfig;
```

3.2.3. Prio3SumVec

```
struct {
    uint32 length;       /* length of the vector */
    uint8 bits;          /* bit length of each summand */
    uint32 chunk_length; /* size of each proof chunk */
} Prio3SumVecConfig;
```

3.2.4. Prio3Histogram

```
struct {
    uint32 length;       /* number of buckets */
    uint32 chunk_length; /* size of each proof chunk */
} Prio3HistogramConfig;
```

3.2.5. Prio3MultihotCountVec

```
struct {
    uint32 length;       /* length of the vector */
    uint32 chunk_length; /* size of each proof chunk */
    uint32 max_weight;   /* largest vector weight /
} Prio3MultihotCountVecConfig;
```

3.2.6. Poplar1

```
struct {
    uint16 bits; /* bit length of the input string */
} Poplar1Config;
```

3.3. Extensions

The TaskConfig structure includes a list of extensions. In general, extensions can be used to bind additional, application-specific information to the task. For example, an extension might be used to encode the identity of the Collector. (Only the Aggregators are identified in TaskConfig.)

Each extension is structured as follows:

```
struct {
  TaskprovExtensionType extension_type;
  opaque extension_data<0..2^16-1>;
} TaskprovExtension;

enum {
  reserved(0),
  (2^16-1)
} TaskprovExtensionType;
```

The extension_type identifies the extension and extension_data is structured as specified by the extension.

Extensions are treated as mandatory-to-implement in the protocol described in Section 4. In particular, protocols participants MUST opt-out of tasks containing unrecognized extensions. See Section 4.4.

If the same extension type appears more than once among the Taskprov extensions in the TaskConfig structure, then protocols participants MUST opt-out of tasks with repeated extensions.

Note that Taskprov extensions are semantically distinct from DAP report extensions and do not share the same codepoint registry (Section 7.2). Future documents may want to define both a Taskprov extension and a report extension, but there may also be situations where a document defines one but not the other.

4. In-band Task Provisioning with the Taskprov Extension

Before a task can be executed, it is necessary to first provision the Clients, Aggregators, and Collector with the task's configuration. The core DAP specification does not define a mechanism for provisioning tasks. This section describes a mechanism whose key feature is that task configuration is performed completely in-band, via HTTP request headers.

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) are carried by HTTP headers piggy-backed on the protocol flow.

This mechanism 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 mechanism should presume the Author is under the adversary's control. In fact, we expect in a real-world deployment that the Author may be co-located with the Collector.

The DAP protocol also 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 section 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.

4.1. Overview

The process of provisioning a task begins when the Author disseminates the task configuration to the Collector and each of the Clients. When a Client issues an upload request to the Leader (as described in Section 4.5 of [DAP]), it includes in an HTTP header the task configuration it used to generate the report. We refer to this process as "task advertisement". Before consuming the report, the Leader parses the configuration and decides whether to opt-in; if not, the task's execution halts.

Otherwise, if the Leader does opt-in, it advertises the task to the Helper during the aggregation interaction (Section 4.6 of [DAP]). In particular, it includes the task configuration in an HTTP header of each aggregation job request for that task. Before proceeding, the Helper must first parse the configuration and decide whether to opt-in; if not, the task's execution halts.

4.2. Task Advertisement

To advertise a task to its peer, a protocol participant includes the header "DAP-Taskprov" with an HTTP request incident to the task execution. The value is the TaskConfig structure defined Section 3.1, expanded into its URL-safe, unpadded Base 64 representation as specified in Sections 5 and 3.2 of [RFC4648].

4.3. Deriving the VDAF Verification Key

When a Leader and Helper implement this mechanism, 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(
        SHA-256("dap-taskprov"), # salt
        verify_key_init,         # IKM
    ),
    task_id,                     # info
    VERIFY_KEY_SIZE,             # L
)
```

where task_id is as defined in Section 3. Functions HKDF-Extract() and HKDF-Expand() are as defined in [RFC5869]. Both functions are instantiated with SHA-256() as defined in [SHS].

4.4. Opting into 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) 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 config or other parameters are 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:

1. The task has ended.

2. The DAP batch mode or VDAF is not implemented.

3. One of the extensions is not recognized.

4. One of the extension types appear more than once.

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

4.5. Client Behavior

Upon receiving a TaskConfig from the Author, the Client decides whether to opt into the task as described in Section 4.4. If the Client opts out, it MUST NOT attempt to upload reports for the task.

Once the client opts into a task, it may begin uploading reports for the task to the Leader. The extension codepoint taskprov MUST be offered in both the Leader and the Helper’s report extensions. The extension may be included in either the public or private report extensions; it is RECOMMENDED that the extension be included in the public extensions. In addition, each report’s task ID MUST be computed as described in Section 3.

The Client SHOULD advertise the task configuration by specifying the encoded TaskConfig described in Section 3 in the "DAP-Taskprov" HTTP header, but MAY choose to omit this header in order to save network bandwidth. However, the Leader may respond with "unrecognizedTask" if it has not been configured with this task. In this case, the Client MUST retry the upload request with the "DAP-Taskprov" HTTP header.

4.6. Leader Behavior

4.6.1. Upload Protocol

Upon receiving a Client report, if the Leader does not support the Section 4 mechanism, it will ignore the "DAP-Taskprov" HTTP header. In particular, if the task ID is not recognized, then it MUST abort the upload request with "unrecognizedTask".

Otherwise, if the Leader does support this mechanism, it first checks if the "DAP-Taskprov" HTTP header is specified. If not present, that means the Client has skipped task advertisement. If the Leader recognizes the task ID, it will include the client report in the aggregation of that task ID. Otherwise, it MUST abort with "unrecognizedTask". The Client will then retry with the task advertisement.

If the Client advertises the task, the Leader checks that the task ID indicated by the upload request matches the task ID derived from the "DAP-Taskprov" HTTP header as specified in Section 3. 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 4.4. If it opts out, it MUST abort the upload request with "invalidTask".

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

During the upload flow, if the Leader’s report share does not present a taskprov extension type, the Leader MUST abort the upload request with "invalidMessage".

4.6.2. Aggregation Protocol

When the Leader opts in to a task, it SHOULD derive the VDAF verification key for that task as described in Section 4.3. The Leader MUST advertise the task to the Helper in every request incident to the task as described in Section 3.

4.6.3. Collection Protocol

The Collector might create a collection job for a task provisioned by this mechanism prior to opting into the task. In this case, the Leader would need to abort the collect request with "unrecognizedTask". When it does so, it is up to the Collector to retry its request.

OPEN ISSUE: This semantics is awkward, as there’s no way for the Leader to distinguish between Collectors who support this mechanism and those that don’t.

The Leader MUST advertise the task in every aggregate share request issued to the Helper as described in Section 4.2.

4.7. Helper Behavior

The Leader advertises a task to the Helper during each step of an aggregation job and when it requests the Helper’s aggregate share during a collection job.

Upon receiving a task advertisement from the Leader, If the Helper does not support this mechanism, it will ignore the "DAP-Taskprov" HTTP header and process the request as usual. In particular, if the Helper does not recognize the task ID, it MUST abort the request with error "unrecognizedTask". Otherwise, if the Helper supports this mechanism, it proceeds as follows.

First, the Helper attempts to parse payload of the "DAP-Taskprov" HTTP header. If this step fails, the Helper MUST abort with "invalidMessage".

Next, the Helper checks that the task ID indicated in the request matches the task ID derived from the TaskConfig as defined in Section 3. If not, the Helper MUST abort with "unrecognizedTask".

Next, the Helper decides whether to opt in to the task as described in Section 4.4. If it opts out, it MUST abort the request with "invalidTask".

Finally, the Helper completes the request as usual, deriving the VDAF verification key for the task as described in Section 4.3. During an aggregation job, for any report share that does not include the taskprov extension with an empty payload, the Helper MUST mark the report as invalid with error "invalid_message" and reject it.

4.8. Collector Behavior

Upon receiving a TaskConfig from the Author, the Collector first decides whether to opt into the task as described in Section 4.4. If the Collector opts out, it MUST NOT attempt to initialize collection jobs 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 4.4. The Collector MUST advertise the task as described in Section 4.2.

If the Leader responds to a collection request with an "unrecognizedTask" error, the Collector MAY retry its request after waiting an appropriate amount of time.

5. Security Considerations

The Taskprov extension has the same security and privacy considerations as the core DAP protocol. In addition, successful execution of a DAP task implies agreement on the task configuration. This is provided by binding the parameters to the task ID, which in turn is bound to each report uploaded for a task. Furthermore, inclusion of the Taskprov extension in the report means Aggregators that do not implement this extension will reject the report as required by (Section 4.5.3 of [DAP]).

The task provisioning mechanism in Section 4 extends the threat model of DAP by including a new logical role, called the Author. The Author is responsible for configuring Clients prior to task execution. 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.

Another risk is that the Author could configure a unique task to fingerprint a Client. Although Client anonymization is not guaranteed by DAP, some systems built on top of DAP may hope to achieve this property by using a proxy server with Oblivious HTTP [RFC9458] to forward Client reports to the Leader. If the Author colludes with the Leader, the attacker can learn some metadata information about the Client, e.g., the Client IP, user agent string, which may deanonymize the Client. However, even if the Author succeeds in doing so, the Author should learn nothing other than the fact that the Client has uploaded a report, assuming the Client has verified the privacy parameters of the task before opting into it. For example, if a task is uniquely configured for the Client, the Client can enforce the minimum batch size is strictly more than 1.

Another risk for the Aggregators is that 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. This can be mitigated by limiting the rate at which new tasks are configured. In addition, deployments SHOULD

arrange for the Author to digitally sign the task configuration so that Clients cannot forge task creation, e.g., via an extension to Taskprov (Section 3.3).

Support for the Taskprov extension may render a deployment of DAP more susceptible to task enumeration attacks (Section 8.6.1 of [DAP]). For example, if the Leader's upload endpoint is unauthenticated, then any HTTP client can learn if a Leader supports a particular task configuration by uploading a report for it with the Taskprov extension. Aggregators can mitigate these kinds of attack by:

1. Requiring authentication of all APIs, including the upload endpoint (see Section 3.3 of [DAP]);

2. Enforcing rate limits on unauthenticated APIs; or

3. Including entropy in the task_info field of the TaskConfig in order to make the task ID harder to predict (e.g., 16 bytes of output of a CSPRNG).

6. Operational Considerations

The Taskprov extension does not introduce any new operational considerations for DAP.

The task provisioning mechanism in Section 4 is designed so that the Aggregators do not need to store individual task configurations long-term. Because the task configuration is advertised in each request in the upload, aggregation, and collection protocols, the process of opting-in and deriving the task ID and VDAF verify key can be re-run on the fly for each request. This is useful if a large number of concurrent tasks are expected. Once an Aggregator has opted-in to a task, the expectation is that the task is supported until it ends. In particular, Aggregators that operate in this manner MUST NOT opt out once they have opted in.

7. IANA Considerations

This document requests a codepoint for the taskprov report extension and for creation of a new registry for Taskprov extensions.

(RFC EDITOR: Replace "XXXX" with the RFC number assigned to this document.)

7.1. Report Extension

The following entry will be (RFC EDITOR: change "will be" to "has been") added to the "Report Extension Identifiers" registry of the "Distributed Aggregation Protocol (DAP)" page created by [DAP]:

Value: 0xff00

Name: taskprov

Reference: RFC XXXX

7.2. Registry for Taskprov Extensions

A new registry will be (RFC EDITOR: change "will be" to "has been") created for the "Distributed Aggregation Protocol (DAP)" page called "Taskprov Extensions". This registry contains the following columns:

Value: The two-byte identifier for the extension

Name: The name of the extension

Reference: Where the mechanism is defined

The initial contents of this registry are listed in the following table.

Value	Name	Reference
0x0000	reserved	Section 3.3 of RFC XXXX

Table 2: Initial contents of the Taskprov Extensions registry.

7.3. DAP Sub-namespace for DAP

TODO Figure out how to ask IANA to register the errors in Table 1. See https://github.com/ietf-wg-ppm/draft-ietf-ppm-dap-taskprov/issues/34

7.4. HTTP Field Name Registration

A new entry to the "Hypertext Transfer Protocol (HTTP) Field Name Registry" will be (RFC EDITOR: change "will be" to "has been") added for the in-band task-provisioning header:

Field Name	Status	Reference
DAP-Taskprov	permanent	Section 4 of RFC XXXX

Table 3: Updates to the Hypertext Transfer Protocol (HTTP) Field Name Registry

8. Extending this Document

The behavior of the taskprov extension may be extended by future documents that define:

1. A new DAP batch mode

2. A new VDAF

3. A new Taskprov extension

Documents defining either a new DAP batch mode or VDAF SHOULD include a section titled "Taskprov Considerations" that specifies the payload of TaskConfig.batch_config or TaskConfig.vdaf_config respectively.

Note that the registry for batch modes is defined by [DAP]; the registry for VDAFs is defined by [VDAF]; and the registry for Taskprov extensions is defined in Section 7.2 of this document.

9. Normative References

[DAP] Geoghegan, T., Patton, C., Pitman, B., Rescorla, E., and C. A. Wood, "Distributed Aggregation Protocol for Privacy Preserving Measurement", Work in Progress, Internet-Draft, draft-ietf-ppm-dap-16, 2 September 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-ppm-dap-16>.

[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>.

[RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, DOI 10.17487/RFC4648, October 2006, <https://www.rfc-editor.org/rfc/rfc4648>.

[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>.

[RFC9458] Thomson, M. and C. A. Wood, "Oblivious HTTP", RFC 9458, DOI 10.17487/RFC9458, January 2024, <https://www.rfc-editor.org/rfc/rfc9458>.

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

[VDAF] Barnes, R., Cook, D., Patton, C., and P. Schoppmann, "Verifiable Distributed Aggregation Functions", Work in Progress, Internet-Draft, draft-irtf-cfrg-vdaf-15, 17 June 2025, <https://datatracker.ietf.org/doc/html/draft-irtf-cfrg-vdaf-15>.

Contributors

Junye Chen Apple Inc. junyec@apple.com

David Cook ISRG divergentdave@gmail.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