Workgroup: Privacy Preserving Measurement
Internet-Draft:
draft-dcook-ppm-dap-interop-test-design-06
Published: 19 September 2023
Intended Status: Informational
Expires: 22 March 2024
Authors: D. Cook
ISRG

DAP Interoperation Test Design

Abstract

This document defines a common test interface for implementations of the Distributed Aggregation Protocol for Privacy Preserving Measurement (DAP) and describes how this test interface can be used to perform interoperation testing between the implementations. Tests are orchestrated with containers, and new test-only APIs are introduced to provision DAP tasks and initiate processing.

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://divergentdave.github.io/draft-dcook-ppm-dap-interop-test-design/draft-dcook-ppm-dap-interop-test-design.html. Status information for this document may be found at https://datatracker.ietf.org/doc/draft-dcook-ppm-dap-interop-test-design/.

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/divergentdave/draft-dcook-ppm-dap-interop-test-design.

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 22 March 2024.

Copyright Notice

Table of Contents

1. Introduction

This document defines a common test interface for implementations of the Distributed Aggregation Protocol for Privacy Preserving Measurement [DAP]. This test interface facilitates interoperation tests between different participating DAP implementations. As DAP has four distinct protocol roles, (Client, Leader, Helper, and Collector) manual interoperation testing between all combinations of even a small number of DAP implementations could be taxing. The goal of this document's common test interface is to enable automation of these interoperation tests, so that different participating implementations can be exchanged for each other, and the same test suite can be re-run on different combinations of implementations. Simplifying interoperation testing will aid in identifying errors in implementations, identifying ambiguities in the protocol specification, and reducing regressions in implementations.

Taking inspiration from QuicInteropRunner [SI2020], each participating implementation provides one or more container images adhering to a common interface. A test runner will start one container for each protocol participant, configure networking between the containers, and send various HTTP API requests. As part of this common testing interface, the HTTP servers in the containers will support some new test-only HTTP APIs, which will allow the test runner to provision shared task parameters and secrets, as well as trigger the start of different sub-protocols.

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.

3. Container Interface

Each participating DAP implementation may provide one or more container images, one for each protocol role it implements. (Client, Leader, Helper, and Collector) A list of available container images will be maintained for each role. Implementations may want to submit a single Aggregator image in both the Leader list and Helper list.

The test runner will fetch each container using the given repository, image name, and tag.

When the container's entry point executable is run, it **SHALL** start up an HTTP server listening on port 8080. In all cases, the container will serve the endpoints described in Section 4 (particularly, the subsection or subsections appropriate to its protocol role). In the case of a Helper or Leader container, it **SHALL** also serve the endpoints specified by [DAP] on a port (which **MAY** be the same port 8080 as used to serve the interoperation test API) at some relative path. The container should run indefinitely, and the test runner will terminate the container on completion of the test case. (While DAP requires HTTPS connections, only using HTTP between containers simplifies test setup. Putting TLS client/server interop out-of-scope for these tests is acceptable, as it's not of interest.)

Log output **SHOULD** be captured into the directory "/logs" inside the container. This will be copied out to the host for inspection on completion of the test case.

No environment variables or volume mounts will be provided to the containers.

4. Interoperation Test API

Each container will have an HTTP server listening on port 8080 for commands from the test runner. All requests **MUST** use the HTTP method POST. Requests and responses for each endpoint listed below **SHALL** be encoded JSON objects [RFC8729], with media type application/json. All binary blobs (i.e. task IDs, batch IDs, HPKE configurations, and VDAF verification keys) **SHALL** be encoded as strings with base64url [RFC4648], inside the JSON objects. Any integer values in the parameters, measurement, or aggregate result of a [VDAF] will be encoded as strings in base 10 instead of as numbers. This avoids incompatibilities due to limitations on the range of JSON numbers that different implementations can process.

Each of these test APIs should return a status code of 200 OK if the command was received, recognized, and parsed successfully, regardless of whether any underlying DAP request succeeded or failed. The DAP-level success or failure will be included in the test API response body. If a request is made to an endpoint starting with "/internal/test/", but not listed here, a status code of 404 Not Found **SHOULD** be returned, to simplify the introduction of new test APIs.

4.1. Common Structures

4.1.1. VDAF

In multiple APIs defined below, the test runner will send the name of a [VDAF], along with the parameters necessary to fully specify the VDAF. These will be stored in a nested object, with the following attributes (new type values and new keys will be added as new VDAFs are defined).

Key	Value
type	One of "Prio3Count", "Prio3Histogram", "Prio3Sum", "Prio3SumVec", or "Poplar1"
length (only present if type is "Prio3Histogram" or "Prio3SumVec")	The length of the vectors being summed, encoded in base 10 as a string.
chunk_length (only present if type is "Prio3Histogram" or "Prio3SumVec")	This parameter is required by the parallel sum circuit optimization used in these VDAFs. It is a positive number encoded in base 10 as a string.
bits (only present if type is "Prio3Sum", "Prio3SumVec", or "Poplar1")	In the case of Prio3Sum or Prio3SumVec, the bit width of the integers being summed, encoded in base 10 as a string. In the case of Poplar1, the bit length of the input, encoded in base 10 as a string.

Table 1: VDAF JSON object structure

4.1.2. Query

In multiple APIs defined below, the test runner will need to send a query type, and in one API, it will need to send a query type along with the associated query parameters.

Query types are represented in API requests as numbers, following the values of the QueryType enum in [DAP].

Queries are represented in API requests as a nested object, with the following attributes (new keys will be added as new query types are defined).

Key	Value
type	A number, representing a query type, as described above.
batch_interval_start (only present if type is 1, for time interval queries)	The start of the batch interval, represented as a number equal to the number of seconds since the UNIX epoch.
	The duration of the batch interval in seconds, as a number.

Key	Value
batch_interval_duration (only present if type is 1, for time interval queries)	
subtype (only present if type is 2, for fixed size queries)	0 or 1, representing one of the values of the FixedSizeQueryType enum in [DAP].
batch_id (only present if type is 2, for fixed size queries, and subtype is 0, for "by batch ID" queries)	A base64url-encoded DAP BatchID.

Table 2: Query JSON object structure

4.2. Client

4.2.1. /internal/test/ready

The test runner will POST an empty object (i.e. {}) to this endpoint to check if the Client container is ready to serve requests. If it is ready, it **MUST** return a status code of 200 OK.

4.2.2. /internal/test/upload

Upon receipt of this command, the Client container will construct a DAP report with the given configuration and measurement, and submit it. The Client container will send its response to the test runner once report submission has either succeeded or permanently failed.

Key	Value
task_id	A base64url-encoded DAP TaskId.
leader	The Leader's endpoint URL.
helper	The Helper's endpoint URL.
vdaf	An object, with the layout given in Table 1. This determines the VDAF to be used when constructing a report.
measurement	If the VDAF's type is "Prio3Count": "0" or "1". If the VDAF's type is "Prio3Sum": a string (representing an integer in base 10). If the VDAF's type is "Prio3SumVec": an array of strings, each representing an integer in base 10. If the VDAF's type is "Prio3Histogram": a string (representing an integer in base 10). If the VDAF's type is "Poplar1": an array of Booleans.
time (optional)	If present, this provides a substitute time value that should be used when constructing the report. If not present, the current system time should be used, as per normal. The time is represented as a number,

Key	Value
	with a value of the number of seconds since the UNIX epoch.
time_precision	A number, providing the precision in seconds of report timestamps.

Table 3: Request JSON object structure

Key	Value
status	"success" if the report was submitted to the Leader successfully, or "error" otherwise.
error (optional)	An optional error message, to assist in troubleshooting. This will be included in the test runner logs.

Table 4: Response JSON object structure

4.3. Aggregator (Leader or Helper)

4.3.1. /internal/test/ready

The test runner will POST an empty object (i.e. {}) to this endpoint to check if the Aggregator container is ready to serve requests. If it is ready, it **MUST** return a status code of 200 OK.

4.3.2. /internal/test/endpoint_for_task

Request the base URL for DAP endpoints for a new task. This API will be invoked immediately before /internal/test/add_task (see Section 4.3.3), to determine the endpoint URLs of the Aggregators. If the Aggregator uses a common set of DAP endpoints for all tasks, it could always return the same value, such as the relative URL /. Alternately, implementations may wish to generate new endpoints for each task, derive the endpoint based on the TaskId, etc.

The test runner will provide the hostname at which the Aggregator is externally reachable. If the Aggregator returns a relative URL, the test runner will combine it with the hostname into an absolute URL, assuming that the port is 8080. Otherwise, the Aggregator can incorporate the hostname into an absolute URL and return that.

Key	Value
task_id	A base64url-encoded DAP TaskId.
role	Either "leader" or "helper".
hostname	This Aggregator's hostname in the interoperation test environment. This may optionally be used in constructing the endpoint URL as an absolute URL.

Table 5: Request JSON object structure

Key	Value
status	"success" if the endpoint was successfully selected or set up, or "error" otherwise.
error (optional)	An optional error message, to assist in troubleshooting. This will be included in the test runner logs.
endpoint	A relative or absolute URL, specifying the DAP Aggregator endpoint that should be used for this task. If the test runner receives a relative URL, it will transform it into an absolute URL before performing the next phase of task setup.

Table 6: Response JSON object structure

4.3.3. /internal/test/add_task

Register a task with the Aggregator, with the given configuration and secrets.

At least one of the HPKE keypairs available for this task should use the mandatory-to-implement algorithms in section 6 of [DAP], for broad compatibility.

Key	Value
task_id	A base64url-encoded DAP TaskId.
leader	The Leader's endpoint URL. The test runner will ensure this is an absolute URL.
helper	The Helper's endpoint URL. The test runner will ensure this is an absolute URL.
vdaf	An object, with the layout given in Table 1. This determines the task's VDAF.
leader_authentication_token	The authentication token that is shared with the other Aggregator, as a string. This string **MUST** be safe for use as an HTTP header value. When the Leader sends HTTP requests to the Helper, it **MUST** include this value in a header named DAP-Auth-Token.
collector_authentication_token (only present if role is "leader")	The authentication token that is shared between the Leader and Collector, as a string. This string **MUST** be safe for use as an HTTP header value. When the Collector sends HTTP requests to the Leader, it **MUST** include this

Key	Value
	value in a header named DAP-Auth-Token.
role	Either "leader" or "helper".
vdaf_verify_key	The VDAF verification key shared by the two Aggregators, encoded with base64url.
max_batch_query_count	A number, providing the maximum number of batches any report may be included in, and thus the number of aggregate results it may contribute to.
query_type	A number, representing the task's query type, as described in Section 4.1.2.
min_batch_size	A number, providing the minimum number of reports that must be in a batch for it to be collected.
max_batch_size (only present if query_type is 2, for fixed size queries)	A number, providing the maximum number of reports that may be in a batch for it to be collected.
time_precision	A number, providing the precision in seconds of report timestamps. For tasks using the time interval query type, the batch interval's duration will always be a multiple of this value.
collector_hpke_config	The Collector's HPKE configuration, encoded in base64url, for encryption of aggregate shares.
task_expiration	A number, providing the time when Clients are no longer expected to upload to this task. This is represented as a number of seconds since the UNIX epoch.

Table 7: Request JSON object structure

Key	Value
status	"success" if the task was successfully set up, or "error" otherwise. (for example, if the VDAF was not supported)
error (optional)	An optional error message, to assist in troubleshooting. This will be included in the test runner logs.

Table 8: Response JSON object structure

4.4. Collector

4.4.1. /internal/test/ready

The test runner will POST an empty object (i.e. {}) to this endpoint to check if the Collector container is ready to serve requests. If it is ready, it **MUST** return a status code of 200 OK.

4.4.2. /internal/test/add_task

Register a task with the Collector, with the given configuration. Returns the Collector's HPKE configuration for this task.

The HPKE keypair generated for this task should use the mandatory-to-implement algorithms in section 6 of [DAP], for broad compatibility.

Key	Value
task_id	A base64url-encoded DAP TaskId.
leader	The Leader's endpoint URL.
vdaf	An object, with the layout given in Table 1. This determines the task's VDAF.
collector_authentication_token	The authentication token that is shared between the Leader and Collector, as a string. This string **MUST** be safe for use as an HTTP header value. When the Collector sends HTTP requests to the Leader, it **MUST** include this value in a header named DAP-Auth-Token.
query_type	A number, representing the task's query type, as described in Section 4.1.2.

Table 9: Request JSON object structure

Key	Value
status	"success" if the task was successfully set up, or "error" otherwise. (for example, if the VDAF was not supported)
error (optional)	An optional error message, to assist in troubleshooting. This will be included in the test runner logs.
collector_hpke_config (if successful)	The Collector's HPKE configuration, encoded in base64url, for encryption of aggregate shares.

Table 10: Response JSON object structure

4.4.3. /internal/test/collection_start

Send a collection request to the Leader with the provided parameters, and return a handle to the test runner identifying this collection job. The test runner will provide this handle to the Collector in subsequent /internal/test/collection_poll requests (see Section 4.4.4).

Key	Value
task_id	A base64url-encoded DAP TaskId.
agg_param	A base64url-encoded aggregation parameter.
query	An object, with the layout given in Table 2. This provides the collection job's query, and in turn determines which reports should be included.

Table 11: Request JSON object structure

Key	Value
status	"success" if the collection request succeeded, or "error" otherwise.
error (optional)	An optional error message, to assist in troubleshooting. This will be included in the test runner logs.
handle (if successful)	A handle produced by the Collector to refer to this collection job. This must be a string.

Table 12: Response JSON object structure

4.4.4. /internal/test/collection_poll

The test runner sends this command to a Collector to poll for completion of the collection job associated with the provided handle. The Collector provides the status and (if available) results to the test runner.

Key	Value
handle	The handle for a collection job from a previous invocation of /internal/test/collection_start. (see Section 4.4.3)

Table 13: Request JSON object structure

Key	Value
status	Either "complete" if the result is ready, "in progress" if the result is not yet ready, or "error" if an error occurred.
error (optional)	An optional error message, to assist in troubleshooting. This will be included in the test runner logs.

Key	Value
batch_id (if the task uses fixed size queries)	The identifier of the batch that was collected, encoded with base64url.
report_count (if complete)	A number, reflecting the count of Client reports included in this aggregated result.
interval_start (if complete)	The start of the collection's interval, represented as a number equal to the number of seconds since the UNIX epoch.
interval_duration (if complete)	The duration of the collection's interval in seconds, as a number.
result (if complete)	The result of the aggregation. If the VDAF is of type Prio3Count or Prio3Sum, this will be a string, representing an integer in base 10. If the VDAF is of type Prio3Histogram, Prio3SumVec, or Poplar1, this will be an array of strings, each representing an integer in base 10.

Table 14: Response JSON object structure

4.5. Test Cases

Test cases could be written to cover the following scenarios.

- Test successful aggregations with each VDAF.
- Test an aggregation over a few hundred or thousand reports, to exercise the Aggregators' division of reports into aggregation jobs.
- Test that uploading a report with a time far in the future is rejected.
- Confirm that Leaders and Helpers reject requests with respective authentication tokens that are incorrect.
- Test enforcement of max_batch_query_count by making overlapping collection requests.
- Perform an entire aggregation and collection flow, attempt to upload a late report that falls into the same batch interval, and test that performing the collection request a second time yields the same result.
- Attempt to upload a canned report from the test runner more than once, and confirm that anti-replay measures were effective by inspecting the aggregation result.

4.6. Other Test Considerations

All test cases should automatically fail after a generous timeout.

It is the responsibility of the test runner to wait for all containers to start up and respond successfully to a request to /internal/test/ready before sending any further commands.

Aggregator URLs will be constructed by the test runner with hostnames that resolve to the respective containers within the container network.

A reverse proxy could be introduced in front of each Aggregator to inject failures when sending requests or responses, to test round skew recovery stragegies and overall implementation resilience.

4.7. Test Runner Operation

The following sequence outlines how the test runner will use the above APIs on port 8080 of each container to perform a typical integration test, executing a successful aggregation.

1. Create and start containers.
2. Set up networking between containers.
3. Try sending /internal/test/ready requests to each container, and retry until they succeed.
4. Generate a random TaskId, random authentication tokens, and a VDAF verification key.
5. Send a /internal/test/endpoint_for_task request (Section 4.3.2) to the Leader.
6. Send a /internal/test/endpoint_for_task request (Section 4.3.2) to the Helper.
7. Construct Aggregator URLs using the above responses.
8. Send a /internal/test/add_task request (Section 4.4.2) to the Collector. (the Collector generates an HPKE key pair as a side-effect)
9. Send a /internal/test/add_task request (Section 4.3.3) to the Leader.
10. Send a /internal/test/add_task request (Section 4.3.3) to the Helper.

11. Send one or more /internal/test/upload requests (Section 4.2.2) to the Client.

12. Send one or more /internal/test/collection_start requests (Section 4.4.3) to the Collector. (this provides a handle for use in the next step)

13. Send /internal/test/collection_poll requests (Section 4.4.4) to the Collector, polling until each collection is completed. (the Collector will provide the calculated aggregate results)

14. Stop containers.

15. Copy logs out of each container.

16. Delete containers, and clean up container networking resources.

5. Implementation Status

[Janus], [divviup-ts], and [Daphne] currently implement a version of this test interface. [REF-IMPL] is a reference implementation of a test runner using this interface.

Additional DAP implementations would be warmly welcomed.

6. Security Considerations

Any DAP implementation that adopts this testing interface should ensure that the test-only APIs described herein are only present in software used for testing purposes, and not in production systems.

7. IANA Considerations

This document has no IANA actions.

8. References

8.1. 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-07, 14 September 2023, <https://datatracker.ietf.org/doc/html/draft-ietf-ppm-dap-07>.

[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>.

[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>.

[RFC8729] Housley, R., Ed. and L. Daigle, Ed., "The RFC Series and RFC Editor", RFC 8729, DOI 10.17487/RFC8729, February 2020, <https://www.rfc-editor.org/rfc/rfc8729>.

[VDAF] Barnes, R., Cook, D., Patton, C., and P. Schoppmann, "Verifiable Distributed Aggregation Functions", Work in Progress, Internet-Draft, draft-irtf-cfrg-vdaf-07, 31 August 2023, <https://datatracker.ietf.org/doc/html/draft-irtf-cfrg-vdaf-07>.

8.2. Informative References

[Daphne] "Implementation of DAP", 15 December 2022, <https://github.com/cloudflare/daphne>.

[divviup-ts] "TypeScript client for https://divviup.org", 5 October 2022, <https://github.com/divviup/divviup-ts>.

[Janus] "Experimental implementation of the DAP specification", 25 August 2022, <https://github.com/divviup/janus>.

[REF-IMPL] "Reference DAP interoperation test harness", 4 October 2022, <https://github.com/divergentdave/dap-interop-test-runner>.

[SI2020] Seemann, M. and J. Iyengar, "Automating QUIC Interoperability Testing", 10 August 2020, <https://research.protocol.ai/publications/automating-quic-interoperability-testing/seemann2020.pdf>.

Acknowledgments

Thanks to Brandon Pitman, Christopher Patton, and Tim Geoghegan for feedback and contributions.

Author's Address

David Cook
ISRG

Email: dcook@divviup.org