Workgroup: Privacy Preserving Measurement
Internet-Draft:
draft-dcook-ppm-dap-interop-test-design-00
Published: 4 August 2022
Intended Status: Informational
Expires: 5 February 2023
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-PPM) 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-PPM 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

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 5 February 2023.

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-PPM]. This test interface facilitates interoperation tests between different participating DAP-PPM implementations. As DAP-PPM has four distinct protocol roles, (client, leader aggregator, helper aggregator, and collector) manual interoperation testing between all combinations of even a small number of DAP-PPM 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 appropriate to its protocol role). In the case of a helper or leader container, it **SHALL** also serve the endpoints specified by [DAP-PPM] on the same port, 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-PPM 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. 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, HPKE configurations, and verification keys) **SHALL** be encoded as strings with base64url [RFC4648], inside the JSON objects.

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-PPM 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

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 "Prio3Aes128Count", "Prio3Aes128Sum", or "Prio3Aes128Histogram"

Key	Value
bits (only present if type is "Prio3Aes128Sum")	The bit width of the integers being summed, (as a number) used to parameterize the Prio3Aes128Sum VDAF.
buckets (only present if type is "Prio3Aes128Histogram")	An array of histogram bucket boundaries, (as numbers) used to parameterize the Prio3Aes128Histogram VDAF.

Table 1: VDAF JSON object structure

4.2. Client

4.2.1. /internal/test/upload

Upon receipt of this command, the client container will construct a DAP-PPM 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
taskId	A base64url-encoded DAP-PPM TaskId.
leader	The leader's endpoint URL.
helper	The helper's endpoint URL.
vdaf	An object, with the layout given in {vdaf-object}. This determines the VDAF to be used when constructing a report.
measurement	If the VDAF's type is "Prio3Aes128Count": 0 or 1. If the VDAF's type is "Prio3Aes128Sum": a number (should be an integer). If the VDAF's type is "Prio3Aes128Histogram": a number (should be an integer).
nonceTime (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, with a value of the number of seconds since the UNIX epoch.

Table 2: 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 3: Response JSON object structure

4.3. Aggregator (Leader or Helper)

4.3.1. /internal/test/endpoint_for_task

Request the base URL for DAP-PPM endpoints for a new task. This API will be invoked immediately before /internal/test/add_task (see Section 4.3.2), to determine the endpoint URLs of the aggregators. If the aggregator uses a common set of DAP-PPM 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 and port at which the aggregator is externally reachable. If the aggregator returns a relative URL, the test runner will combine it with the hostname and port into an absolute URL. Otherwise, the aggregator can incorporate the hostname and port into an absolute URL and return that.

Key	Value
taskId	A base64url-encoded DAP-PPM TaskId
aggregatorId	0 if this aggregator is the leader, or 1 if this aggregator is the helper.
hostnameAndPort	This aggregator's hostname and port in the interoperation test environment. This may optionally be used in constructing the endpoint URL as an absolute URL.

Table 4: 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-PPM 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 5: Response JSON object structure

4.3.2. /internal/test/add_task

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

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

Key	Value
taskId	A base64url-encoded DAP-PPM 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 {vdaf-object}. This determines the task's VDAF.
leaderAuthenticationToken	The authentication bearer token that is shared with the other aggregator, as a string. This string must be safe for use as an HTTP header value.
collectorAuthenticationToken (only present if aggregatorId is 0)	The authentication bearer token that is shared between the leader and collector, as a string. This string must be safe for use as an HTTP header value.
aggregatorId	0 if this aggregator is the leader, or 1 if this aggregator is the helper.
verifyKey	The verification key shared by the two aggregators, encoded with base64url.
maxBatchLifetime	A number, providing the maximum number of times any report can be included in a collect request.
minBatchSize	A number, providing the minimum number of reports that must be in a batch for it to be collected.
minBatchDuration	A number, providing the minimum number of seconds that can be in a batch's interval. The batch interval will always be a multiple of this value.
collectorHpkeConfig	The collector's HPKE configuration, encoded in base64url, for encryption of aggregate shares.

Table 6: 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 7: Response JSON object structure

4.4. Collector

4.4.1. /internal/test/add_task

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

Key	Value
taskId	A base64url-encoded DAP-PPM TaskId.
leader	The leader's endpoint URL.
vdaf	An object, with the layout given in {vdaf-object}. This determines the task's VDAF.

Table 8: 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.
collectorHpkeConfig (if successful)	The collector's HPKE configuration, encoded in base64url, for encryption of aggregate shares.

Table 9: Response JSON object structure

4.4.2. /internal/test/collect_start

Send a collect request to the leader with the provided parameters, and return a handle to the test runner identifying this collect request. The test runner will provide this handle to the collector in subsequent /internal/test/collect_poll requests (see Section 4.4.3).

Key	Value
taskId	A base64url-encoded DAP-PPM TaskId.

Key	Value
aggParam	A base64url-encoded aggregation parameter.
batchIntervalStart	The start of the batch interval, represented as a number equal to the number of seconds since the UNIX epoch.
batchIntervalDuration	The duration of the batch interval in seconds, as a number.

Table 10: Request JSON object structure

Key	Value
status	"success" if the collect 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 collect request. This must be a string.

Table 11: Response JSON object structure

4.4.3. /internal/test/collect_poll

Upon receiving this command, the collector will poll the leader's collect URL for the collect job associated with the provided handle, and provide the status and result to the test runner.

Key	Value
handle	The handle for a collect request from a previous invocation of /internal/test/collect_start. (see Section 4.4.2)

Table 12: Request JSON object structure

Key	Value
status	Either "complete" if the result was returned, "in progress" if the result was 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.
result (if complete)	The result of the aggregation. If the VDAF is of type Prio3Aes128Count or Prio3Aes128Sum, this will be a number. If the VDAF is of type Prio3Aes128Histogram, this will be an array of numbers.

Table 13: Response JSON object structure

4.4.4. Heavy Hitters

Once Poplar1 reaches a future draft of [DAP-PPM], additional test APIs for collector containers should be introduced to perform an entire Heavy Hitters computation on a given Poplar1 task and

collection interval, encompassing multiple collect flows automatically initiated by the collector.

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_lifetime by making overlapping collect requests.
- Perform an entire aggregation and collect flow, attempt to upload a late report that falls into the same collect interval, and test that performing the collect 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 listen on their ports before sending any commands.

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

Once a future [DAP-PPM] draft solves the issue of retries in the aggregate flow, a reverse proxy could be introduced in front of each aggregator to inject failures when sending requests or responses, to test the protocol's resilience. (It is known such a test would fail based on the current protocol.)

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 opening a TCP connection to each container's port, and retry until it succeeds.
4. Generate a random TaskId, random authentication tokens, and a VDAF verification key.
5. Send a /internal/test/endpoint_for_task request (Section 4.3.1) to the leader.
6. Send a /internal/test/endpoint_for_task request to the helper.
7. Construct aggregator URLs using the above responses.
8. Send a /internal/test/add_task request (Section 4.4.1) to the collector. (the collector generates an HPKE key pair as a side-effect)
9. Send a /internal/test/add_task request (Section 4.3.2) to the leader.
10. Send a /internal/test/add_task request (Section 4.3.2) to the helper.
11. Send one or more /internal/test/upload requests (Section 4.2.1) to the client.
12. Send a /internal/test/collect_start request (Section 4.4.2) to the collector. (this provides a handle for use in the next step)
13. Send /internal/test/collect_poll requests (Section 4.4.3) to the collector, polling until it is completed. (the collector will provide the calculated aggregate result)
14. Stop containers.
15. Copy logs out of each container.
16. Delete containers, and clean up container networking resources.

5. Implementation Status

This document has not yet been implemented at time of writing. There are two open source DAP-PPM implementations, [Janus] from Divvi Up and [Daphne] from Cloudflare Research, which could be extended with this testing interface.

The TypeScript [divviup-ts] library could readily be turned into a client-only test participant.

Both divviup-ts and the VDAF proof of concept implementation [VDAF-POC] could be used as the VDAF core of a non-production DAP-PPM implementation, to provide diversity of VDAF implementations in interoperation tests.

Additional DAP-PPM implementations would be warmly welcomed.

6. Security Considerations

Any DAP-PPM 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-PPM] Geoghegan, T., Patton, C., Rescorla, E., and C. A. Wood, "Distributed Aggregation Protocol for Privacy Preserving Measurement", 11 July 2022, <https://datatracker.ietf.org/doc/html/draft-ietf-ppm-dap-01>.

[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>.

8.2. Informative References

[Daphne] "Implementation of DAP", 27 July 2022, <https://github.com/cloudflare/daphne>.

[divviup-ts] "TypeScript client for https://divviup.org", 27 July 2022, <https://github.com/divviup/divviup-ts>.

[Janus] "Experimental implementation of the DAP-PPM specification", 26 July 2022, <https://github.com/divviup/janus>.

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

[VDAF-POC] "VDAF reference implementations", 11 July 2022, <https://github.com/cfrg/draft-irtf-cfrg-vdaf/tree/main/poc>.

Acknowledgments

Thanks to Brandon Pitman and Christopher Patton for feedback and contributions.

Author's Address

David Cook
ISRG

Email: dcook@letsencrypt.org