Internet Engineering Task Force
Internet-Draft
Intended status: Informational
Expires: September 11, 2021

R. Belchior
M. Correia
INESC-ID, Instituto Superior Tecnico
T. Hardjono
MIT

March 10, 2021

DLT Gateway Crash Recovery Mechanism
draft-belchior-gateway-recovery-01

Abstract

This memo describes the crash recovery mechanism for the Open Digital Asset Protocol (ODAP), entitled ODAP-2PC. ODAP-2PC assures that gateways running ODAP are crash-fault tolerant, meaning that the atomicity of asset transfers are assured even if gateways crash. This protocol includes the description of the messaging and logging flow necessary for gateways to keep track of current state, the crash recovery protocol, and a rollback protocol.

Table of Contents

1. Introduction

Gateway systems that perform virtual asset transfers among DLTs must possess a degree of resiliency and fault tolerance in the face of possible crashes. A key component of crash recovery is maintaining logs that enable either the same or other backup gateways to resume partially completed transfers. Another key component is an atomic commit protocol (ACP) that guarantees that the source and target DLTs are modified consistently (atomicity) and permanently (durability), e.g., that assets that are taken from the source DLT are persisted into the recipient DLT.

This memo proposes: (i) the parameters that a gateway must retain in the form of logs concerning message flows within asset transfers; (ii) a JSON-based format for logs related to asset transfers.

2. Terminology

There following are some terminology used in the current document:

- Gateway: The nodes of a DLT system that are functionally capable of handling an asset transfer with another DLT. Gateway nodes implement the gateway-to-gateway asset transfer protocol.
- Primary Gateway: The node of a DLT system that has been selected or elected to act as a gateway in an asset transfer.
- Backup Gateway: The node of a DLT system that has been selected or elected to act as a backup gateway to a primary gateway.
- Message Flow Parameters: The parameters and payload employed in a message flow between a sending gateway and receiving gateway.
- Source Gateway (or G1): The gateway that initiates the transfer protocol. Acts as a coordinator of the ACP and mediates the message flow.
- Recipient Gateway (or G2): The gateway that is the target of an asset transfer. It follows instructions from the source gateway.
- Source DLT: The DLT of the source gateway.
- Target DLT: The DLT of the recipient gateway.
- Log: Set of log entries such that those are ordered by the time of its creation.
- Public (or Shared) Log: log where several nodes can read and write from it.
- Private Log: log where only one node can read and write from it.
- Log data: The log information is retained by a gateway connected to an exchanged message within an asset transfer protocol.
- Log entry: The log information generated and persisted by a gateway regarding one specific message flow step.
- Log format: The format of log-data generated by a gateway.
- Atomic commit protocol (ACP): A protocol that guarantees that assets that are taken from a DLT are persisted into the other DLT. Examples are two and three-phase commit protocols (2PC, 3PC, respectively) and non-blocking atomic commit protocols.

- Fault: A fault is an event that alters the expected behavior of a system.

- Crash-fault tolerant models: models allowing a system to keep operating correctly despite having a set of faulty components.

- Digital asset: a form of digital medium recordation that is used as a digital representation of a tangible or intangible asset.

3. Logging Model

Logs are associated to a process running operations on a certain gateway, and they can be stored in several supports: 1) off-chain storage (with the possibility of a hash of the logs being stored on-chain), where logs are stored on the hard-drive of the computer system performing the role of a gateway; 2) cloud storage; 3) on-chain storage, either storing the logs on the blockchains that gateways are connected, or to a third blockchain.

To manipulate the log, we define a set of log primitives, that translate log entry requests from a process into log entries, realized by the log storage API, later presented:

- writeLogEntry(l,L) - writes a log entry l in the log L

- getLogLength - obtains the number of log entries

- getLogEntry(l) - retrieves a log entry l.

A log entry request typically comes from a single event in a given protocol. Log entry requests have the format (phase, step, operation, gateways), where the field operation corresponds to an arbitrary command, and the field gateways correspond to the parties involved in the protocol. We define four operations types to provide context to the protocol being executed. Operation type (init-) states the intention of a gateway to execute a particular operation, and operation (exec-) expresses that the gateway is excecuting an operation. The operation type (done-) states when an agent successfully executed a step of the protocol, while (ack-) refers to when a gateway acknowledges a message received from another. Conversely, we use the type (fail-) to refer to when an agent fails to execute a specific step.

3.1. Example

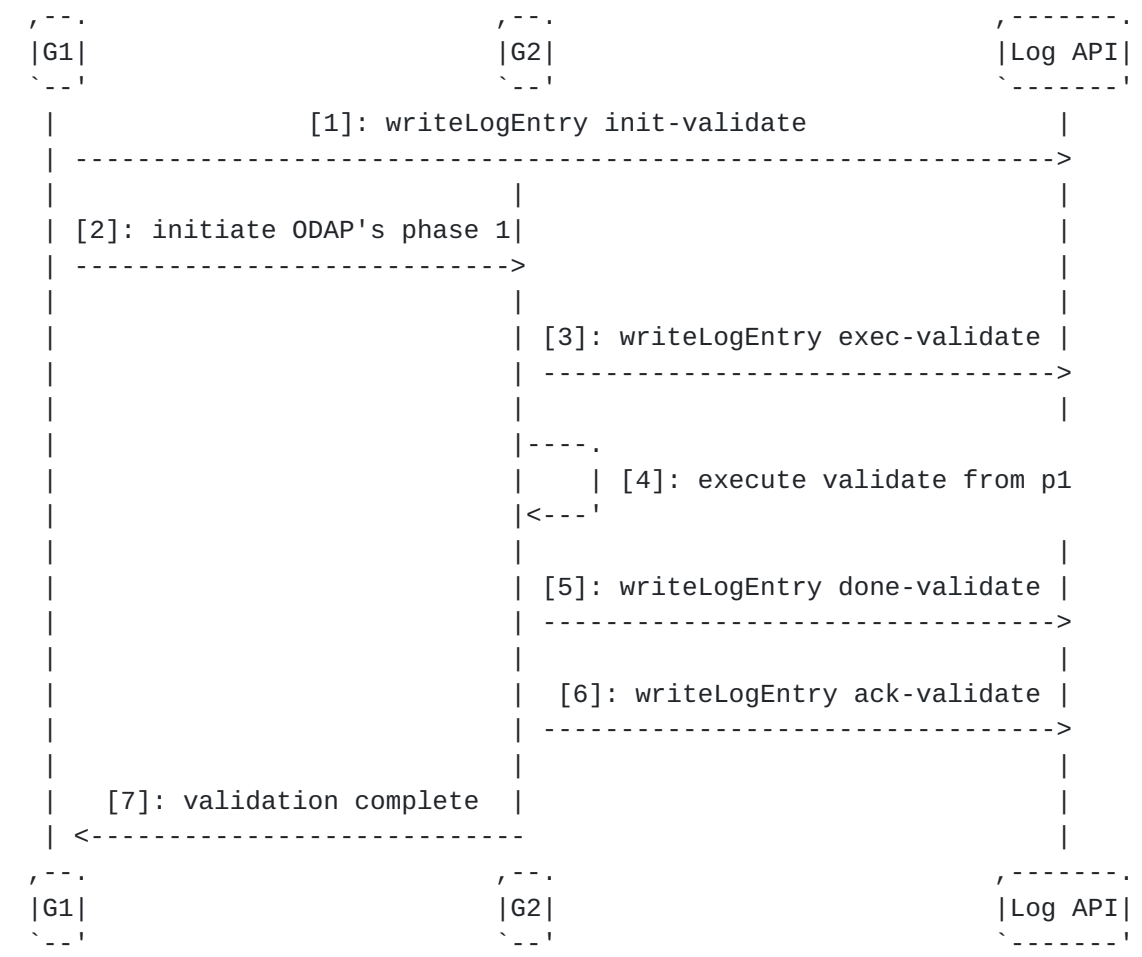

Figure 1

From step 1 to 7, the generated logs are:

At step 1, LOG: <p1, 1, init-validate, (GS->GR)>.

At step 2, GS commands GR to execute validate.

At step 3, LOG: <p1, 2, exec-validate, (GR)>.

At step 4: GR executes validate

At step 5, LOG: <p1,3, done-validate, (GR)>.

At step 6, LOG: <p1, 4, ack-validate, (GR->GS)>.

At step 7: GS receives an acknoledgment from GR.

4. Gateway Crash Recovery

The gateway architecture [ODAP] defines two gateway nodes belonging to distinct DLT systems as a means to conduct a virtual asset transfer in a secure and non-repudiable manner while ensuring the asset does not exist simultaneously on both blockchains.

One of the key deployment requirements of gateways for asset transfers is a high degree of gateways availability. In this document, we consider two common strategies to increase availability: (1) to support the recovery of the gateways and (2) to employ backup gateways with the ability to resume a stalled transfer.

To this end, gateways must retain relevant log information regarding incoming protocol messages (parameters, payloads, etc.) and transmitted messages. In particular, logs are written before operations (write-ahead) to provide atomicity and durability to the asset exchange protocol. The log-data is considered as internal resources to the DLT system, accessible to the backup gateway and possible other gateway nodes.

4.1. Gateway Transfer Model

The Open Digital Asset Protocol (ODAP) is a DLT-agnostic gateway-to-gateway protocol used by a sender gateway and a target gateway to perform a virtual asset's unidirectional transfer [ODAP]. The transfer process is started by a client (application) that interacts with the source gateway or both (source and recipient) gateways to provide instructions regarding actions, related resources located in the source DLT system, and resources located in the remote DLT system. The protocol has two modes, but here we consider only the Relay Mode: Client-initiated Gateway to Gateway asset transfer. When we refer to the ODAP protocol in this document, we refer to the ODAP protocol in Relay Mode, although the logging model specified in this memo can also support the Direct mode., although the logging model specified in this memo can also support the Direct mode.

ODAP has to be instanced with an ACP protocol to guarantee that the source and target DLTs are modified consistently, a property designated Atomicity [BHG87]. ACPs consider two roles: a Coordinator that manages the execution of the protocol and Participants that manage the resources that must be kept consistent. The source gateway plays the ACP role of Coordinator, and the recipient gateway plays the Participant role in relay mode. Gateways exchange messages corresponding to the protocol execution, generating log entries for each one. The message exchange, and corresponding logging procedure is represented in Figure 1.

The simplified message flow format is in the form < ODAP_PHASE, STEP, COMMAND, GATEWAY >, where ODAP_PHASE corresponds to the current phase of ODAP, STEP corresponds to a monotonically increasing integer, COMMAND to the command type being issued by a set of gateways (GATEWAY). However, both two-phase commit and three-phase commit can block in case nodes fail. The protocol being blocking means that if the coordinator crashes, then gateways may not finish transactions. When a crash happens, gateways will be waiting for a confirmation/ abort, and possibly holding the lock regarding a specific digital asset.

4.2. Crash Recovery Model

We assume gateways fail by crashing, i.e., by becoming silent, not arbitrary or Byzantine faults. We assume authenticated reliable channels obtained using TLS/HTTPS [TLS]. To recover from these crashes, gateways store in persistent storage data about the step of their protocol. This allows the system to recover by getting from the log the first step that may have failed. We consider two recovery models:

- Self-healing mode: assumes that after a crash, a gateway eventually recovers;

- Primary-backup mode: assumes that after a crash, a gateway may never recover, but that this failure can be detected by timeout [AD76].

In Self-healing mode, when a gateway restarts after a crash, it reads the state from the log and continues executing the protocol from that point on. We assume the gateway does not lose its long-term keys (public-private key pair) and can reestablish all TLS connections.

In Primary-backup mode, we assume that after a period T of the primary gateway failure, a backup gateway detects that failure unequivocally and takes the role of the primary gateway. The failure is detected using heartbeat messages and a conservative value for T. The backup gateway does virtually the same as the gateway in self-healing mode: reads the log and continues the process. The difference is that the log must be shared between the primary and the backup gateways. If there is more than one backup, a leader-election protocol may be executed to decide which backup will take the primary role.

4.3. Recovery Procedure

Gateways can crash at several points of the protocol.

In 2PC and 3PC, recovery requires that the protocol steps are recorded in a log immediately before sending a message and immediately after receiving a message. Thus, at every step k of the protocol, each gateway writes in the log entry indicating its current state. When a node crashes:

- Self-healing mode: the recovered gateway informs the other party of its recovery and continues the protocol execution;

- Primary-backup mode: if a node is crashed indefinitely, a backup is spun off, using the log storage API to retrieve the most recent version of the log.

Upon recovery, the recovered node attempts to retrieve the most recent log of operations. Based on the latest log entry last(log), it derives the current state of the asset transfer. This can be confirmed by querying all other nodes involved in such transfer by sending a recovery message rm. After the current state is fetched and agreed upon by all parties, the ODAP protocol continues. There are several situations when a crash may occur. The first one is immediately after starting the transfer, as shown below:

Figure 2

The source gateway (G1) crashes right before it issued an init command to the recipient gateway (G2). The gateway eventually

recovers in self-healing mode, querying the last log entry from the log storage API. After that, it sends a recovery message to G2, advertising that the recovery has been completed and asking for an updated version of the log, i.e., the current state. In this case, the latest version of the log corresponds to G1 log. After synchronization has been achieved, the process can continue.

The second scenario requires further synchronization (figure below). At the retrieval of the latest log entry, G1 notices its log is outdated. It updates it upon necessary validation and then communicates its recovery to G2. The process then continues as defined.

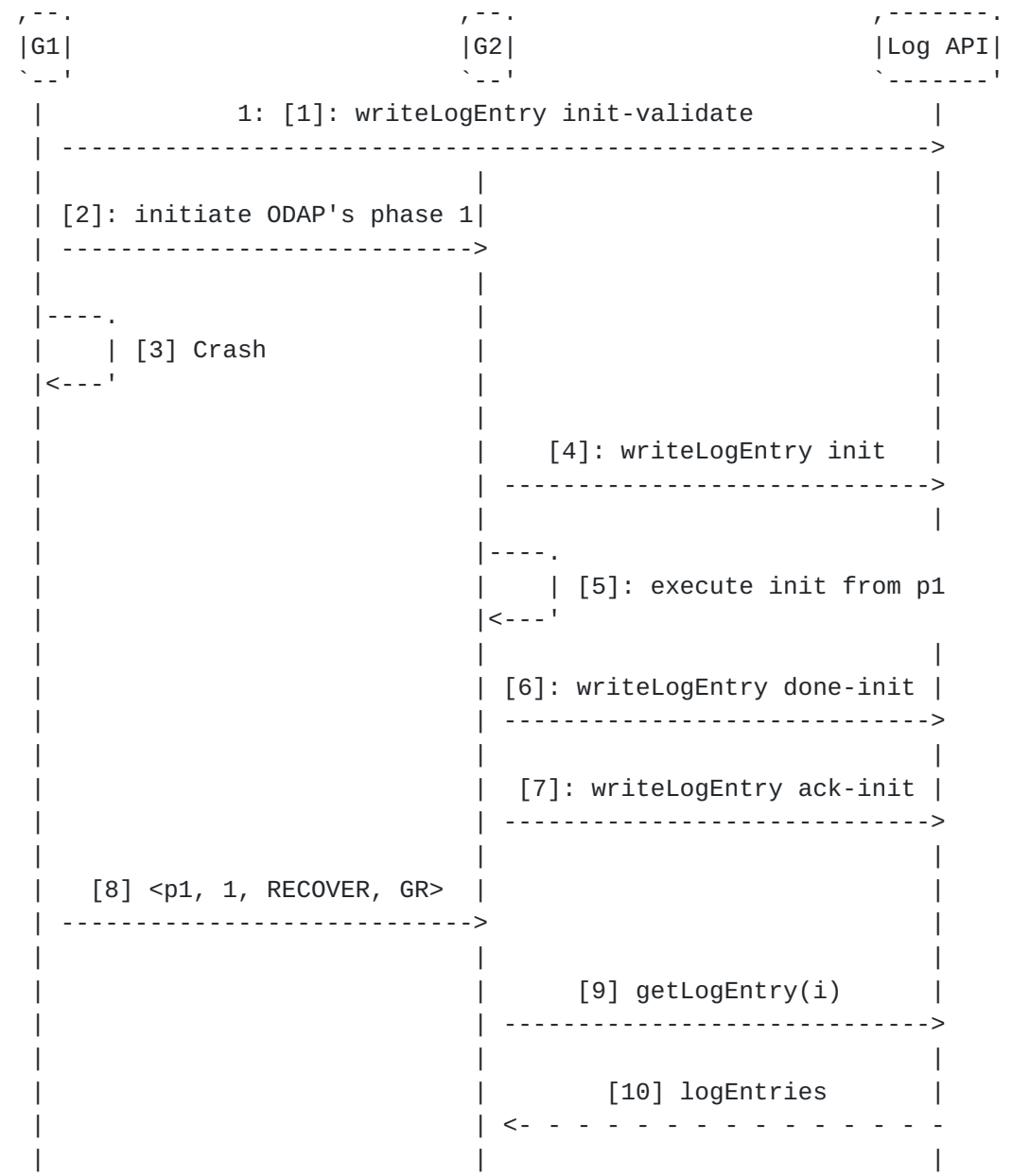

Figure 3

4.4. Log Storage

Log primitives are translated into log entries, persisted by the log storage API in the format <operation, step, phase, gateways>, where the gateway issuing the operation is implicit. For example, when GS initiates ODAP's first phase, by sending a message to GR, a log entry specifying the command init given to G2, in the first operation of the phase p1 is translated to a log entry <p1,1,init-validate,GS-GR)>. After that, the log entry is persisted via the log storage API. Thus, log primitives are also translated into log storage API requests.

We consider the log file to be a stack of log entries. Each time a log entry is added, it goes to the top of the stack (the highest index). Logs can be saved locally (computer?s disk), in an external service (e.g., cloud storage service), or in the DLT the gateway is operating. Saving logs locally is faster than saving them on the respective ledger but delivers weaker integrity and availability guarantees. Saving log entries on a DLT may slow down the protocol because issuing a transaction is several orders of magnitude slower than writing on disk or accessing a cloud service. Self-healing mode is compatible with the three types of logs, but Primary-backup mode

requires storage in an external service or the DLT. For critical scenarios where strong accountability and traceability are needed (e.g., financial institution gateways), blockchain-based logging storage may be appropriate. Conversely, for gateways that implement interoperability between blockchains belonging to the same organization (i.e., a legal framework protects the legal entities involved), local storage might suffice.

We assume the storage service used provides the means necessary to assure the logs' confidentiality and integrity, stored and in transit. The service must provide an authentication and authorization scheme, e.g., based on OAuth and OIDC [OIDC], and use secure channels based on TLS/HTTPS [TLS].

We consider a log storage API that allows developers to abstract from the storage details (e.g., relational vs. non-relational, local vs. cloud) and handles access control if needed. This is API-TYPE 1, as the gateway uses it to store off-chain resources.

4.5. Logging API

The log storage API serves two purposes: 1) it provides a reliable mean to store logs created by all gateways involved in an asset transfer; and 2) promote accountability across parties.

The log storage API MUST respond with return codes indicating the failure (error 5XX) or success of the operation (200). The application may carry out further operation in future to determine the ultimate status of the operation.

4.5.1. POST/saveLogEntry:log

Persists a log entry at the default storage environment, by appending it to the current log. Returns the index of the saved log entry.

Response example:

```
HTTP/1.1 200 OK
Cache-Control: private
Date: Mon, 02 Mar 2020 05:07:35 GMT
Content-Type: application/json

   {
      "success": true,
      "response_data":"2"
   }
```

Figure 4

4.5.2. GET lastEntry

Obtains the latest log entry from the log.

Response example:

```
HTTP/1.1 200 OK
Cache-Control: private
Date: Mon, 02 Mar 2020 05:07:35 GMT
Content-Type: application/json

{
     "success": true,
     "response_data":
           "log_entry": {...}
}
```

Figure 5

4.5.3. GET getLogEntry/:id

Obtains a log entry with specified ID.

Response example:

```
HTTP/1.1 200 OK
Cache-Control: private
Date: Mon, 02 Mar 2020 05:07:35 GMT
Content-Type: application/json

 {
      "success": true,
      "response_data":
            "log_entry": {...}
}
```

Figure 6

4.5.4. GET getLog

Obtains the whole log.

Response example:

```
HTTP/1.1 200 OK
Cache-Control: private
Date: Mon, 02 Mar 2020 05:07:35 GMT
Content-Type: application/json

{
      "success": true,
      "response_data":
            "log": {...}
}
```

Figure 7

4.5.5. POST updateLog

Updates the current log. The log is updated if there are new log entries.

Returns the index of the last common log entry (common prefix).

Response example:

```
HTTP/1.1 200 OK
Cache-Control: private
Date: Mon, 02 Mar 2020 05:07:35 GMT
Content-Type: application/json

 {
    "success": true,
    "response_data":"2"
 }
```

Figure 8

5. Format of log entries

The log entries are stored by a gateway in its log. Entries account for the current status of one of the three ODAP flows: Transfer Initiation flow, Lock-Evidence flow, and Commitment Establishment flow. The recommended format for log entries is JSON [xxx], with protocol-specific mandatory fields, support for a free format field for plaintext or encrypted payloads directed at the DLT gateway or an underlying DLT. Although the recommended format is JSON, other formats can be used (e.g., XML).

The mandatory fields of a log entry are:

- session_ID REQUIRED: unique identifier (UUIDv2) representing an ODAP interaction (corresponding to a particular flow)
- seq_number REQUIRED: represents the ordering of steps recorded on the log for a particular session
- odap_phase REQUIRED: flow to which the logging refers to. Can be Transfer Initiation flow, Lock-Evidence flow, and Commitment Establishment flow.
- source_gateway_pubkey REQUIRED: the public key of the gateway initiating a transfer
- source_gateway_dlt_system REQUIRED: the ID of the gateway initiating a transfer
- recipient_gateway_pubkey REQUIRED: the public key of the gateway involved in a transfer
- recipient_gateway_dlt_system REQUIRED: the ID of the recipient gateway involved in a transfer

- timestamp REQUIRED: timestamp referring to when the log entry was generated (UNIX format)

- payload REQUIRED: Message payload. Contains subfields Votes (optional), Msg, Message type. Votes refers to the votes parties need to commit in the 2PC. Msg is the content of the log entry. Message type refers to the different logging actions (e.g., command, backup). Msg and Message type are specific to the ODAP phase [ODAP].

- payload_hash REQUIRED: hash of the current message payload

Optional log entry fields are:

- logging_profile: contains the profile regarding the logging procedure. If not present, a local store for the logs is assumed.

- source_gateway_uid: the uid of the source gateway involved in a transfer

- recipient_gateway_uid : the uid of the recipient gateway involved in a transfer

- message_signature: Gateway EDCSA signature over the log entry

- last_entry_hash: Hash of previous log entry

- access_control_profile: the profile regarding the confidentiality of the log entries being stored

Example of a log entry created by G1, corresponding to locking an asset (phase 2.3 of the ODAP protocol) :

```
{
    "sessionId": "4eb424c8-aead-4e9e-a321-a160ac3909ac",
    "seqNumber": 6,
    "phaseId": "lock",
    "sourceGatewayId": "5.47.165.186",
    "sourceDltId": "Hyperledger-Fabric-JusticeChain",
    "targetGatewayId": "192.47.113.116",
    "targetDltId": "Ethereum",
    "timestamp": "1606157330",
    "payload": {
        "messageType": "2pc-log",
        "message": "LOCK_ASSET",
        "votes": "none"
 },
 "payloadHash":
"80BCF1C7421E98B097264D1C6F1A514576D6C9F4EF04955FA3AEF1C0664B34E3",
"logEntryHash": "[...]"
}
```


Figure 9

Example of a log entry created by G2, acknowledging G1 locking an asset (phase 2.4 of the ODAP protocol) :


```
{
    "sessionId": "4eb424c8-aead-4e9e-a321-a160ac3909ac",
    "seqNumber": 7,
    "phaseId": "lock",
    "sourceGatewayId": "5.47.165.186",
    "sourceDltId": "Hyperledger-Fabric-JusticeChain",
    "targetGatewayId": "192.47.113.116",
    "targetDltId": "Ethereum",
    "timestamp": "1606157333",
    "payload": {
        "messageType": "2pc-log",
        "message": "LOCK_ASSET_ACK",
        "votes": "none"
    }
    ,
    "payloadHash":
"84DA7C54F12CE74680778C22DAE37AEBD60461F76D381D3CD855B0713BB98D1",
"logEntryHash": "[...]"
}
```


Figure 10

6. Security Considerations

We assume a trusted, secure communication channel between gateways (i.e., messages cannot be spoofed and/or altered by an adversary) using TLS 1.3 or higher. Clients support ?acceptable? credential schemes such as OAuth2.0.

The present protocol is crash fault-tolerant, meaning that it handles gateways that crash for several reasons (e.g., power outage). The present protocol does not support Byzantine faults, where gateways can behave arbitrarily (including being malicious). This implies that both gateways are considered trusted. We assume logs are not tampered with or lost.

Log entries need integrity, availability, and confidentiality guarantees, as they are an attractive point of attack [BVC19]. Every log entry contains a hash of its payload for guaranteeing integrity. If extra guarantees are needed (e.g., non-repudiation), a log entry might be signed by its creator. Availability is guaranteed by the usage of the log storage API that connects a gateway to a dependable storage (local, external, or DLT-based). Each underlying storage provides different guarantees. Access control can be enforced via the access control profile that each log can have associated with, i.e., the profile can be resolved, indicating who can access the log entry in which condition. Access control profiles can be implemented with access control lists for simple authorization. The authentication of the entities accessing the logs is done at the Log Storage API level (e.g., username+password authentication in local storage vs. blockchain-based access control in a DLT).

For extra guarantees, the nodes running the log storage API (or the gateway nodes themselves) can be protected by hardening technologies such as Intel SGX [CD16].

7. References

7.1. Normative References

[ODAP] Hargreaves, M. and T. Hardjono, "Open Digital Asset Protocol, October 2020, IETF, draft-hargreaves-odap-00.", October 2020, <https://datatracker.ietf.org/doc/draft-hargreaves-odap/>.

[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/info/rfc2119>.

[TLS] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3?, RFC 8446.", 2018, <https://tools.ietf.org/rfc/rfc8446>.

7.2. Informative References

[AD76] Alsberg, P. and D. Day, "A principle for resilient sharing of distributed resources. In Proc. of the 2nd Int. Conf. on Software Engineering", 1976, <978-0-201-10715-9>.

[BHG87] Bernstein, P., Hadzilacos, V., and N. Goodman, "Concurrency Control and Recovery in Database Systems, Chapter 7. Addison Wesley Publishing Company", 1987, <https://doi.org/10.3389/fbloc.2019.00024>.

[BVC19] Belchior, R., Vasconcelos, A., and M. Correia, "Towards Secure, Decentralized, and Automatic Audits with Blockchain. European Conference on Information Systems", 2019, <https://aisel.aisnet.org/ecis2020_rp/68/>.

[Clar88] Clark, D., "The Design Philosophy of the DARPA Internet Protocols, ACM Computer Communication Review, Proc SIGCOMM 88, vol. 18, no. 4, pp. 106-114", August 1988.

[HS2019] Hardjono, T. and N. Smith, "Decentralized Trusted Computing Base for Blockchain Infrastructure Security, Frontiers Journal, Special Issue on Blockchain Technology, Vol. 2, No. 24", December 2019, <https://doi.org/10.3389/fbloc.2019.00024>.

[OIDC] Sakimura, N., Bradley, J., Jones, M., de Medeiros, B., and C. Mortimore, "OpenID Connect Core 1.0", 2014, <http://openid.net/specs/openid-connect-core-1_0.html>.

[SRC84] Saltzer, J., Reed, D., and D. Clark, "End-to-End Arguments in System Design, ACM Transactions on Computer Systems, vol. 2, no. 4, pp. 277-288", November 1984.

Authors' Addresses

Rafael Belchior
INESC-ID, Instituto Superior Tecnico

Email: rafael.belchior@tecnico.ulisboa.pt

Miguel Correia
INESC-ID, Instituto Superior Tecnico

Email: miguel.p.correia@tecnico.ulisboa.pt

Thomas Hardjono
MIT

Email: hardjono@mit.edu