Liquid Protocol: Bridging AI Agents and Solid Pods via the Model Context Protocol

Meem Arafat Manab[1*] Víctor Rodríguez-Doncel[1†] Beatriz Esteves[2‡]

[1]Ontology Engineering Group, Universidad Politécnica de Madrid, Madrid, Spain
[2]IDLab, Department of Electronics and Information Systems, Ghent University – imec, Ghent, Belgium

Abstract

We present Liquid Protocol, a specification for AI agent access to decentralized Solid Pods via the Model Context Protocol (MCP). Analogous to how the W3C Linked Data Platform (LDP) specified HTTP-based access to Linked Data resources for web applications, Liquid Protocol defines MCP-based operations for AI agents to access user-controlled RDF data. This architectural approach addresses two critical challenges in AI systems: the inability to comply with GDPR's Right to Be Forgotten when personal data is embedded in model weights or vector databases, and the tendency of large language models to hallucinate information when lacking access to structured, authoritative data sources. Our reference implementation demonstrates instant Right to Be Forgotten compliance, elimination of hallucination through structured RDF queries, and preservation of data sovereignty through Solid's decentralized architecture. We contribute the first specification bridging AI tool protocols with user-controlled semantic data storage.

1 Introduction

The W3C Linked Data Platform (LDP) specification [1] established how web applications access Linked Data resources through standardized HTTP operations. A decade later, AI agents have emerged as a new class of software requiring access to personal, structured data, yet no equivalent standard exists for agent-based access patterns. Meanwhile, current AI architectures create fundamental conflicts with data protection regulations and user sovereignty.

Modern AI systems absorb personal data in two ways: through training, which embeds information in model weights, or through Retrieval-Augmented Generation (RAG), which stores documents in vector databases. Both approaches make compliance with GDPR's Right to Be Forgotten (RTBF) effectively impossible. Removing data from trained models requires complete retraining, a process that costs between $1 million and $10 million and takes three to six months. Similarly, removing data from vector databases requires re-embedding the entire corpus, which takes days to weeks and costs thousands to tens of thousands of dollars. For production systems serving continuous requests, these remediation approaches are economically and operationally infeasible.

Large language models also suffer from a hallucination problem. They rely on parametric memory, knowledge encoded in weights during training. When uncertain, models confabulate rather than acknowledge ignorance, producing confident but incorrect responses [4]. While RAG systems mitigate this by retrieving relevant text chunks before generation, they still operate on unstructured data that fragments semantic relationships. When documents are split into chunks for embedding, the explicit relationships between entities are lost, leading to ambiguous or incorrect interpretations.

We present Liquid Protocol, which enables AI agents to access Solid Pods [2] through the Model Context Protocol [3]. This protocol mediates among user, AI agents, and the Solid infrastructure, facilitating context-aware interactions that retain semantic integrity. Our contributions include a specification mapping MCP

*ORCID: 0000-0002-2336-4160, meem.manab@upm.es
†ORCID: 0000-0003-1076-2511
‡ORCID: 0000-0003-0259-7560

primitives to Solid Pod operations analogous to LDP's mapping of HTTP methods to Linked Data resources, a permission model extending Solid's Web Access Control with time-limited grants for AI agents, a reference implementation demonstrating instant RTBF compliance and structured data access, and evaluation showing reduced hallucination through RDF-based queries.

2 Related Work

The W3C Linked Data Platform specification [1] established HTTP-based Linked Data access, defining how HTTP methods operate on RDF resources and containers. The Solid Protocol [2] built on LDP to create decentralized user-controlled pods, introducing WebID for identity management and Web Access Control for granular permissions. Liquid Protocol extends this lineage to AI agent access patterns, creating the first specification for how AI systems should interact with user-controlled semantic data.

Retrieval-Augmented Generation [5] grounds LLM responses in retrieved documents, reducing hallucination by providing context from external sources. However, RAG systems use unstructured text chunks stored in centralized databases, neither preserving semantic relationships nor providing user control over their data. The fragmentation of documents into chunks for vector embedding loses the explicit connections between entities that RDF graphs maintain. Liquid Protocol addresses both limitations by accessing structured RDF data that remains under user control in Solid Pods.

Function calling [6] and tool use [7] enable LLMs to use external tools, with various agent frameworks emerging to coordinate these capabilities. The Model Context Protocol standardizes this through a protocol-first approach, defining how AI systems discover and invoke external tools through a common interface. Liquid Protocol provides the first specification for MCP-based access to semantic, user-controlled data, bridging the gap between AI tool protocols and the Semantic Web vision of interconnected, machine-readable information.

3 Background

3.1 Solid Protocol

Solid (Social Linked Data) is a W3C specification for decentralized data storage where users control their data in personal "Pods" [2]. Data is stored as RDF triples, enabling structured, queryable representations of information. Solid uses WebID for decentralized identity, allowing users to authenticate without relying on centralized identity providers. Web Access Control (WAC) provides granular permissions, enabling users to specify exactly which applications and agents can access which resources. Unlike centralized platforms where data is scattered across corporate servers, Solid gives users the ability to grant and revoke access to specific resources at any time, maintaining sovereignty over their personal information.

3.2 Model Context Protocol

Released by Anthropic in November 2024, MCP is an open protocol standardizing how AI agents connect to external data sources and tools [3]. Before MCP, each AI tool needed custom integrations with each data source, creating multiplicative complexity where N AI systems and M data sources required $N \times M$ separate integrations. MCP reduces this to $N + M$ integrations through a standard protocol.

MCP defines three core primitives. Tools are model-controlled executable actions that AI agents can invoke, functioning like remote procedure calls that perform operations and return results. Resources are application-controlled data sources that AI can read without side effects, providing context similar to REST API GET endpoints. Prompts are user-controlled templates that guide how agents should behave for specific tasks, offering reusable instructions exposed by the server.

MCP tools include JSON schemas defining their parameters, enabling AI agents to understand exactly what data to provide for each operation. Servers advertise available tools through a discovery mechanism, allowing agents to dynamically learn capabilities. The protocol supports multiple transport mechanisms, including stdio for local processes, Server-Sent Events for remote servers, and HTTP/SSE for production deployments.

3.3 W3C Linked Data Platform

The LDP specification [1] defined how HTTP methods operate on Linked Data resources and containers, enabling web applications to interact with structured semantic data through standardized operations. LDP provided discovery through HTTP OPTIONS and Link headers, allowing applications to understand resource capabilities before interacting with them. The specification established patterns for creating, reading, updating, and deleting RDF resources, as well as managing containers that organize collections of resources. LDP has served as the foundation for the Solid Protocol's resource access patterns, demonstrating how standardized protocols enable interoperability across diverse implementations.

4 Design

4.1 Architectural Principles

Liquid Protocol follows three key principles that distinguish it from existing approaches. First, data remains at source rather than being absorbed into AI systems. Personal data never enters model weights or vector embeddings. AI agents query Solid Pods at runtime through MCP operations and never internalize the data, ensuring that the authoritative copy remains under user control.

Second, agents access structured data rather than unstructured text. Instead of retrieving document chunks from vector databases, agents access RDF graphs where semantic relationships are explicit and preserved. SPARQL queries enable precise, structured retrieval that maintains the connections between entities, avoiding the fragmentation inherent in chunk-based approaches.

Third, all access is governed by user control through Solid's Web Access Control with time-limited grants. Users can revoke access or delete data at any time with instant effect. The time-limited nature of permissions means that agents must periodically renew their access, giving users regular opportunities to reconsider whether continued access is appropriate.

4.2 Protocol Mapping

Liquid Protocol maps traditional LDP operations to MCP tools for Solid Pods, creating an analogous relationship to how LDP mapped HTTP methods to Linked Data operations. Reading an RDF resource corresponds to the `liquid_read_resource` tool, which takes a resource URI and format specification. Creating and updating resources map to `liquid_write_resource`, which handles both operations depending on whether the resource already exists. Deleting resources maps to `liquid_delete_resource`, enabling instant Right to Be Forgotten compliance. Listing container contents maps to `liquid_list_container`, allowing agents to discover available data. Executing SPARQL queries maps to `liquid_query_sparql`, enabling structured queries that preserve semantic relationships. The complete mapping is provided in Table 1.

LDP (HTTP)	**Liquid (MCP)**	**Operation**
GET resource	liquid_read_resource	Read RDF
POST container	liquid_write_resource	Create
PUT resource	liquid_write_resource	Update
DELETE resource	liquid_delete_resource	Delete
GET container	liquid_list_container	List
SPARQL endpoint	liquid_query_sparql	Query

Table 1: LDP to Liquid Protocol mapping

4.3 Core MCP Tools

The `liquid_read_resource` tool reads an RDF resource from a Solid Pod, with parameters including the resource URI and desired format such as Turtle, JSON-LD, or N-Triples. The tool returns the RDF content or an error if the resource does not exist or access is forbidden.

The `liquid_query_sparql` tool executes SPARQL queries against a Solid Pod, enabling structured queries with preserved semantic relationships. Results can be returned in JSON, CSV, or Turtle format depending on the agent's needs. However, it is important to note that SPARQL is used sparingly in practice, as natural language queries do not always translate well to SPARQL syntax. The protocol supports SPARQL for cases where precise structured queries are needed, but most agent interactions rely on simpler resource reads.

The `liquid_write_resource` tool creates or updates RDF resources, supporting use cases where AI agents need to write back to user pods. Examples include storing agent memory or logging interactions for audit purposes. The tool handles both creation of new resources and modification of existing ones.

The `liquid_delete_resource` tool deletes resources from Solid Pods, enabling instant RTBF compliance. The tool supports recursive deletion for containers, allowing users to remove entire collections of data with a single operation.

The `liquid_list_container` tool lists resources in a Solid Pod container, enabling discovery of available data. Agents can explore the structure of a user's pod to determine what information is accessible and relevant to their task.

4.4 Permission Model

Liquid Protocol extends Solid's Web Access Control with time-limited permissions for AI agents. Each agent has a WebID and requests specific permissions from users. Authorization is expressed in RDF, specifying which agent can access which resource with which mode of access and until what time. The example in Appendix A shows how an agent might receive read access to health records valid until a specific date and time.

The `validUntil` predicate enables temporary access grants, after which permissions automatically expire without user intervention. This provides an additional layer of protection beyond traditional access control, ensuring that agents cannot indefinitely access user data based on a single permission grant. Users can also manually revoke permissions at any time before the expiration, giving them fine-grained control over data access throughout the permission lifecycle. An example authorization is provided in Appendix A.

5 Implementation

5.1 Reference Architecture

Our reference implementation includes three main components. The Community Solid Server provides a local Solid Pod instance with sample health records stored as RDF. The Liquid-Compliant MCP Server implements the specification using the MCP Python SDK, translating between MCP tool calls and Solid Protocol operations. The test suite contains three scenarios demonstrating RTBF compliance, hallucination prevention, and structured queries.

The MCP server runs as a local process, connecting to the AI agent via stdio and to the Solid Pod via HTTP. When an agent invokes a Liquid Protocol tool, the MCP server validates permissions, translates the request into appropriate Solid Protocol operations such as GET, PUT, DELETE, or SPARQL queries, and returns the results to the agent in the format specified by MCP.

5.2 Sample Data

We created realistic RDF health records for evaluation purposes. The data represents a patient named Alice Smith with two diagnosed conditions: Type 2 Diabetes diagnosed on May 15, 2023, and Hypertension diagnosed on January 10, 2024, with Lisinopril 10mg as prescribed medication. This structured representation preserves relationships between patients, conditions, diagnoses, and medications in a way that would be fragmented in RAG systems. The complete RDF representation is provided in Appendix B.

6 Protocol Specification

Liquid Protocol defines three core operation types for AI agent interaction with Solid Pods: Setup for initialization and authentication, Query+CRUD for data operations, and Rights Exercise for GDPR compliance operations. Each operation specifies message flows, data formats, and error handling between five parties: User (data subject), LLM (data controller), Host MCP (data processor containing MCP Client), Solid Pod (containing MCP Server), and Audit Log.

6.1 Protocol Composition and Inheritance

Liquid Protocol is a mapping specification that composes three existing formal protocols. JSON-RPC format, tool discovery, tool calling, transport mechanisms, and error responses are inherited directly from MCP without modification. HTTP methods including GET, POST, PUT, and DELETE, along with status codes, headers, and TLS security are inherited from HTTP specifications. WebID, WebID-TLS, and OIDC authentication flows are inherited from Solid Protocol. RDF triples, Turtle syntax, JSON-LD syntax, and container concepts are inherited from W3C specifications.

The key contributions of Liquid Protocol are the tool schemas defining `liquid_*` operations, the semantic mapping between MCP primitives and Solid Pod operations, mandatory audit logging requirements, GDPR operation flows implementing Articles 15, 16, and 17, time-limited ACL extensions with `validUntil` predicates, and patterns for AI agent identity using WebIDs. Table 2 provides a complete breakdown of which protocol elements are inherited versus newly defined.

This composition approach is analogous to how the W3C Linked Data Platform [1] specified HTTP-based access to Linked Data without redefining HTTP itself. Liquid Protocol inherits the formal properties, state machines, security models, and error handling of its constituent protocols and contributes the semantic mapping layer for AI agent access patterns.

6.2 Data Controller vs Data Processor Designation

Under GDPR Article 4, the Host MCP acts as a data controller because it determines the purposes and means of processing personal data through Liquid Protocol. The Host MCP developer decides which operations to make available (read, write, delete, query), establishes the technical infrastructure for AI agent access to Solid Pods, defines authentication and authorization procedures, and determines how data flows between systems. While the user, as data subject, retains control over their Solid Pod and makes decisions about which agents to grant access to and when to revoke permissions, these are exercises of data subject rights rather than controller functions. The Host MCP provides the mechanisms that enable AI agents to access personal data, making it the entity that determines the means of processing. This distinction is important because Liquid Protocol empowers data subjects with unprecedented control over their data while maintaining clear accountability: the Host MCP operator bears responsibility as data controller for lawful processing, security measures, and compliance with GDPR obligations, while data subjects exercise their rights to access, rectification, erasure, and restriction of processing.

6.3 Operation 1: Setup (Initialization and Authentication)

This operation establishes the connection between the LLM, Host MCP, and Solid Pod, including authentication via One-Time Password (OTP) tokens and capability discovery. The setup phase enables all subsequent data operations by verifying identity and determining what operations the agent is permitted to perform.

The user initiates the process by providing an OTP token and the Host MCP URI to the LLM. The LLM forwards the OTP token to the Host MCP for authentication. The Host MCP validates the OTP token and retrieves the associated WebID from its secure storage. Using this WebID, the Host MCP authenticates with the Solid Pod. The Solid Pod verifies the WebID and returns an authentication token along with a list of capabilities that the agent is permitted to use. The Host MCP makes these methods visible to the LLM through the standard MCP tool listing mechanism. All exchanges are recorded in the Audit Log with timestamps, agent identifiers, and operation types.

Protocol Element	Inherited From	Specification	Liquid Contribution
Transport & Messaging			
JSON-RPC format	MCP	[3]	None (direct use)
Tool discovery	MCP	tools/list	None (direct use)
Tool calling	MCP	tools/call	None (direct use)
Transport (stdio/SSE)	MCP	[3]	None (direct use)
Error responses	MCP	JSON-RPC errors	None (direct use)
HTTP Operations			
GET method	HTTP	RFC 9110	Mapped to liquid_read
POST method	HTTP	RFC 9110	Mapped to liquid_write
PUT method	HTTP	RFC 9110	Mapped to liquid_write
DELETE method	HTTP	RFC 9110	Mapped to liquid_delete
Status codes	HTTP	RFC 9110	None (direct use)
Headers	HTTP	RFC 9110	None (direct use)
TLS security	HTTP	RFC 8446	None (direct use)
Authentication & Authorization			
WebID	Solid	[2]	None (direct use)
WebID-TLS	Solid	[2]	None (direct use)
OIDC flow	Solid	[2]	None (direct use)
Web Access Control	Solid	WAC spec	Time-limited grants
Bearer tokens	HTTP	RFC 6750	None (direct use)
Data Model			
RDF triples	W3C RDF	[10]	None (direct use)
Turtle syntax	W3C	[11]	None (direct use)
JSON-LD syntax	W3C	[12]	None (direct use)
SPARQL queries	W3C	SPARQL 1.1	Exposed via MCP tool
Containers	Solid/LDP	[1]	None (direct use)
Liquid Protocol Contributions			
Tool schemas	New		liquid_* tool definitions
MCP$\longleftrightarrow$Solid mapping	New		Semantic mappings
Audit requirements	New		Mandatory logging spec
GDPR operations	New		Articles 15, 16, 17 flows
Time-limited ACL	New		validUntil extension
Agent identity	New		AI agent WebID patterns

Table 2: Protocol inheritance and Liquid Protocol contributions

This OTP-based approach ensures that the LLM never directly accesses the user's WebID, providing an additional security layer. The OTP token is single-use and time-limited, expiring after successful authentication or after a short timeout period.

The sequence diagram is provided in Figure 1, and detailed message formats are provided in Appendix C. Error handling covers cases where OTP validation fails, where the retrieved WebID lacks permission to access the Solid Pod, where the WebID does not exist, or where services are temporarily unavailable.

6.4 Operation 2: Query+CRUD (Data Operations)

This operation encompasses all read, write, update, delete, list, and query operations on Solid Pod resources. These operations enable the LLM to access structured RDF data, reducing hallucination and grounding responses in authoritative information.

When a user asks a question, the LLM determines which Liquid Protocol tool to invoke. For structured queries, it calls `liquid_query_sparql` with the Pod URI, SPARQL query, and desired result format. The Host MCP verifies that the agent has appropriate permissions according to Web Access Control policies. The Host MCP then sends an HTTP POST request with the SPARQL query to the Solid Pod. The Pod executes the query and returns results in the specified format, typically JSON-LD for agent consumption. These results are recorded in the Audit Log with information about which resources were accessed. The Host MCP packages the results according to MCP specifications and returns them to the LLM, which processes the structured data to formulate its response to the user.

This operation type supports five distinct tools: `liquid_read_resource` for reading individual RDF resources, `liquid_query_sparql` for executing structured queries, `liquid_write_resource` for creating or updating resources, `liquid_delete_resource` for removing data, and `liquid_list_container` for discovering available resources. The sequence diagram is provided in Appendix F.2, and message formats are detailed in Appendix D.

6.5 Operation 3: Rights Exercise (GDPR Compliance)

This operation implements GDPR Articles 15 (Right of Access), 16 (Right to Rectification), and 17 (Right to Erasure). These operations interact with the Audit Log to provide transparency and enable users to exercise their data protection rights.

For the Right of Access under Article 15, users can request their access history by invoking `liquid_get_audit_log`. The Host MCP verifies the user's identity through their WebID and queries the Audit Log for all operations involving that user. The log returns records including timestamps, agent identifiers, operations performed, resources accessed, and query details. This information is formatted and returned to the user, providing complete transparency about how their data has been accessed.

For the Right to Rectification under Article 16, users can correct inaccurate personal data using `liquid_write_resource` with an update operation. The tool allows users to modify specific RDF triples in their Solid Pod, ensuring their data remains accurate and up-to-date.

For the Right to Erasure (Right to Be Forgotten) under Article 17, users can delete their data using `liquid_delete_resource`. The Host MCP sends an HTTP DELETE request to the Solid Pod, which removes the specified resource. This deletion is instant, occurring in less than one second, and does not require model retraining or vector database re-embedding. The deletion is recorded in the Audit Log, and subsequent attempts to access the deleted resource correctly return 404 Not Found errors.

The sequence diagram is provided in Appendix F.3, and detailed message formats for these operations are provided in Appendix E. Error handling addresses cases where requesters lack authorization, where resources have already been deleted, where dependencies exist that prevent deletion, and where systems are temporarily unavailable.

6.6 Protocol Invariants

Across all three operation types, Liquid Protocol maintains several invariants that ensure consistent behavior and compliance with regulations. All operations require valid WebID authentication before execution. Operations respect Solid WAC permissions at all times, with permission checks occurring before each data access. All Setup, Query+CRUD, and Rights Exercise operations are logged with timestamp, agent WebID, operation type, and affected resources to provide a complete audit trail. Read, delete, and audit log query operations are idempotent, meaning repeated execution produces the same result. Operations complete fully or fail entirely with no partial states that could leave data in an inconsistent condition. All access grants include expiration timestamps that are checked before each operation to ensure time-limited access is respected. The Host MCP acts as a data processor, following user instructions without independent decision-making authority over the purposes of data processing.

7 Evaluation

7.1 RTBF Compliance

In traditional architectures, personal data is embedded in model weights or vector databases. An RTBF request requires full retraining or re-embedding, with costs ranging from $1 million to $10 million and timelines of three to six months. With Liquid Protocol, personal data remains in the user's Solid Pod. An RTBF request triggers a `liquid_delete_resource` call, deleting data in less than one second at zero cost. Subsequent queries return 404 Not Found responses, confirming that the data is no longer accessible.

We measured deletion latency across 100 trials to quantify this benefit. Mean deletion time was 0.23 seconds with a standard deviation of 0.04 seconds. All subsequent read attempts correctly returned 404 errors, confirming complete removal without any residual data remaining accessible to the agent.

7.2 Hallucination Prevention

We evaluated a scenario where an agent queries health records for "Alice Smith." Without Liquid Protocol, the agent has no access to a data source and responds with either "I don't have access to that information" or a hallucinated response based on training data. With Liquid Protocol, the agent calls `liquid_query_sparql`, receives structured RDF, and responds with accurate information: "Alice Smith has Type 2 Diabetes (diagnosed 2023-05-15) and Hypertension (diagnosed 2024-01-10, medication Lisinopril 10mg)."

All responses were grounded in actual data from the Solid Pod rather than generated from the model's parametric memory. Zero hallucinations were observed across 50 test queries, demonstrating that structured data access eliminates the uncertainty that leads to confabulation.

7.3 Structured vs. Unstructured Access

In the RAG approach, health records are split into text chunks and embedded in a vector database. A query like "What conditions does Alice have?" retrieves fragments such as "Alice Smith has diabetes..." and "Patient diagnosed with hypertension..." The relationship between patient and conditions becomes ambiguous because chunking separates related information. With Liquid Protocol, a SPARQL query returns complete structured data with explicit relationships. Each condition includes patient reference, diagnosis date, type, and medications, with semantic relationships preserved through RDF triples.

We evaluated information completeness across 30 queries comparing RAG chunks versus Liquid Protocol RDF results. Liquid Protocol achieved 100% relationship preservation, maintaining all explicit connections between entities. RAG systems lost 67% of explicit relationships due to chunking, requiring the agent to infer connections that were originally stated explicitly in the data.

8 Discussion

8.1 Advantages

Liquid Protocol provides GDPR compliance by design, notably supporting the effective exercise of the right to erasure without requiring retraining or re-embedding. Users retain control over their data sovereignty, including the ability to grant and revoke access through fine-grained, machine-readable policies, delete data instantly, and audit access logs transparently [9].

Decentralization and platform independence align with the Web 3.0 vision of user-controlled infrastructure. By relying on open standards such as MCP and Solid, the architecture ensures interoperability and portability. Any MCP-compatible AI system can access any Solid Pod, preventing vendor lock-in to any single AI platform or cloud provider. This architectural choice supports the original vision of the World Wide Web as a decentralized information space where users control their own data and can freely move between service providers, reducing dependence on proprietary data silos and promoting a more decentralized Web.

Data portability, mandated by GDPR Article 20, is inherently supported by the protocol's design. Users can export their Solid Pod data in standard RDF formats and import it into any other Solid-compatible system without vendor-specific conversion processes. This portability extends beyond simple data export to

include the semantic relationships and metadata that give data meaning, ensuring that context is preserved during migration. The same standards-based approach facilitates integration with data spaces, where Solid Pods can function as personal data endpoints and data spaces connectors can enable controlled data exchange under established governance frameworks.

Real-time data updates provide a significant advantage over traditional RAG systems. When information in a Solid Pod changes, AI agents immediately access the updated version on their next query. In contrast, RAG systems require re-embedding documents whenever information changes, creating a window of time during which agents work with stale data. For applications requiring current information such as financial data, medical records, or inventory systems, this real-time access is essential.

Reduced infrastructure costs benefit both users and AI service providers. Users store their data in personal pods, eliminating the need for AI companies to maintain massive vector databases containing copies of user data. This shift reduces storage costs, backup requirements, and the computational expense of maintaining embeddings. AI service providers can focus on model development and inference rather than data storage infrastructure.

Representing data as RDF graphs preserves semantic relationships, enabling more structured and semantically grounded AI reasoning than unstructured text. The explicit nature of RDF relationships means that agents can traverse connections between entities without having to infer relationships from fragmented text chunks. Verifiable data provenance becomes possible through RDF's semantic structure and Solid's authentication mechanisms. Each piece of data in a Solid Pod can include metadata about its source, creation date, and modification history. AI agents can verify that information comes from authoritative sources rather than relying on data of unknown origin embedded in training sets or vector databases. This traceability is crucial for high-stakes applications where data lineage affects trustworthiness.

Compliance by design extends beyond GDPR to other regulatory frameworks including CCPA (California Consumer Privacy Act), HIPAA (Health Insurance Portability and Accountability Act), and sector-specific regulations. The protocol's architecture of user-controlled data access with comprehensive audit logging naturally aligns with requirements across multiple jurisdictions and industries. This multi-regulatory compliance reduces the legal complexity of deploying AI systems in regulated environments.

8.2 Limitations

The protocol requires an additional network hop for each query compared to in-memory RAG, introducing latency that may be unacceptable for some real-time applications. However, we observed acceptable latency for most applications, with mean query times under 500 milliseconds. Caching can be implemented with RTBF-aware invalidation to improve performance while maintaining compliance guarantees. When data is deleted from a Solid Pod, cache entries must be immediately invalidated to ensure that deleted information is not served from cache.

Solid Pod servers may experience downtime or network failures, making data temporarily inaccessible to agents. This reliability concern is inherent to any distributed system architecture. Mitigation strategies include implementing retry logic with exponential backoff, providing graceful degradation when pods are unavailable, and clearly communicating to users when their data cannot be accessed due to pod unavailability.

Natural language queries do not always translate well to SPARQL syntax, which is why SPARQL is used sparingly in practice. While the protocol supports structured queries for cases where precise results are essential, most agent interactions rely on simpler resource reads using `liquid_read_resource`. The complexity of SPARQL presents a barrier to widespread adoption by agents that lack specialized query generation capabilities. Future work could explore methods for automatically generating SPARQL queries from natural language or developing simplified query languages that maintain semantic precision while being more accessible.

Solid Pod adoption is still growing beyond research communities, and MCP was only recently released in November 2024. Both ecosystems have strong backing from W3C and Anthropic respectively, with growing adoption indicating future viability. However, current deployment remains limited compared to centralized alternatives, potentially restricting the immediate practical impact of Liquid Protocol.

Granular WAC permissions require careful design to balance security with usability. Overly restrictive permissions may frustrate users and limit agent utility, while overly permissive grants undermine the privacy benefits of the architecture. Systems can start with simple read and write patterns and extend gradually

as needs develop, allowing developers to gain experience with permission management before implementing more complex access control policies.

8.3 Future Directions

Future work includes developing federated queries that enable cross-pod SPARQL queries spanning multiple users for collaborative applications. Such queries could aggregate information from multiple pods while respecting each user's access control policies, enabling analyses that require data from multiple sources without centralizing storage.

An agent identity framework could establish standards for AI agent WebIDs and reputation systems for trust establishment. Users could make informed decisions about which agents to trust based on verifiable track records of responsible data handling. Such a framework might include cryptographic proofs of agent behavior and community-maintained reputation scores.

Performance optimization through intelligent caching strategies that maintain RTBF compliance guarantees presents another research direction. Developing cache invalidation mechanisms that respond instantly to deletion requests while maximizing cache hit rates could improve system performance without sacrificing compliance.

Finally, W3C standardization through a Community Group for Liquid Protocol specification development could establish this as a formal standard recognized across the industry. Community-driven standardization would enable broader adoption and ensure that the protocol evolves to meet the needs of diverse stakeholders including users, developers, and AI system operators.

9 Conclusion

Liquid Protocol bridges AI agents and Solid Pods through the Model Context Protocol, creating the first specification for agent-based access to user-controlled semantic data. By keeping personal data in Solid Pods rather than model weights or vector databases, we enable instant RTBF compliance at zero cost. By accessing structured RDF rather than unstructured text chunks, we eliminate hallucination through authoritative, relationship-preserving queries. Our reference implementation demonstrates feasibility and our evaluation confirms the approach's effectiveness.

Just as the Linked Data Platform enabled the Semantic Web vision of machine-readable, interoperable data for web applications, Liquid Protocol enables an Agentic Web vision where AI systems respect user sovereignty, ground reasoning in authoritative sources, and comply with privacy regulations by design. We have released our specification and reference implementation as open source to enable community development and standardization efforts.

Acknowledgements

This work has been supported by HARNESS, funded from the EU's Horizon Europe research and innovation programme under grant agreement no. 101169409, and by MALTA PID2024-159504OB-I00 funded by MICIU/AEI/10.13039/501100011033.

References

[1] W3C. *Linked Data Platform 1.0*. W3C Recommendation, 2015. `https://www.w3.org/TR/ldp/`

[2] Capadisli, S., Berners-Lee, T., Verborgh, R., et al. *Solid Protocol*. W3C, 2022. `https://solidproject.org/TR/protocol`

[3] Anthropic. *Model Context Protocol*. 2024. `https://modelcontextprotocol.io`

[4] Ji, Z., et al. *Survey of Hallucination in Natural Language Generation*. ACM Computing Surveys, 2023.

[5] Lewis, P., et al. *Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks*. NeurIPS, 2020.

[6] Schick, T., et al. *Toolformer: Language Models Can Teach Themselves to Use Tools*. arXiv:2302.04761, 2023.

[7] Qin, Y., et al. *Tool Learning with Foundation Models*. arXiv:2304.08354, 2023.

[8] European Parliament. *General Data Protection Regulation (GDPR)*. Official Journal of the European Union, 2016.

[9] B. Esteves, H. J. Pandit, and V. Rodríguez-Doncel "ODRL Profile for Expressing Consent through Granular Access Control Policies in Solid," in *Proceedings of the ESWC 2021 Workshops*, CEUR Workshop Proceedings, 2021

[10] W3C. *RDF 1.1 Concepts and Abstract Syntax*. W3C Recommendation, 2014.

[11] W3C. *RDF 1.1 Turtle*. W3C Recommendation, 2014.

[12] W3C. *JSON-LD 1.1*. W3C Recommendation, 2020.

A Permission Model Example

The following RDF expresses a time-limited authorization for an AI agent to read health records:

```
:auth1 a acl:Authorization ;
  acl:agent <https://agent.ai/webid#me> ;
  acl:accessTo :records.ttl ;
  acl:mode acl:Read ;
  :validUntil "2025-12-31T23:59:59Z" .
```

B Sample Health Records Data

The following RDF represents structured health records used in evaluation:

```
:alice schema:name "Alice Smith" ;
  :hasCondition [
    a :Diabetes ;
    :diagnosedDate "2023-05-15" ;
    :status "Type 2" ;
  ] ;
  :hasCondition [
    a :Hypertension ;
    :diagnosedDate "2024-01-10" ;
    :medication "Lisinopril 10mg"
  ] .
```

C Setup Operation Details

C.1 Sequence Diagram

The sequence diagram is provided in Appendix F.1.

C.2 Message Formats

User Initialization (User → LLM):

```
{
  "action": "initialize",
  "otp_token": "8f7a3e2d9c1b4f6a8e5d7c3b9a2f1e4d",
  "hostMCP": "https://mcp-host.example/liquid"
}
```

Method Discovery (LLM → Host MCP):

```
{
  "jsonrpc": "2.0",
  "method": "tools/list",
  "id": 1
}
```

OTP Authentication (LLM → Host MCP):

```
{
  "jsonrpc": "2.0",
  "method": "liquid/authenticate",
  "params": {
    "otp_token": "8f7a3e2d9c1b4f6a8e5d7c3b9a2f1e4d"
  },
  "id": 2
}
```

OTP Validation (Internal to Host MCP):

```
// Host MCP validates OTP and retrieves WebID
{
  "otp_token": "8f7a3e2d9c1b4f6a8e5d7c3b9a2f1e4d",
  "valid": true,
  "webID": "https://pod.example/alice/profile#me",
  "expires_at": "2025-02-12T10:35:00Z"
}
```

Authentication Request with WebID (Host MCP → Solid Pod):

```
POST /authenticate HTTP/1.1
Host: pod.example
Content-Type: application/json

{
  "webID": "https://pod.example/alice/profile#me",
  "clientID": "https://mcp-host.example/client#id"
}
```

Authentication Response (Solid Pod → Host MCP):

```
{
  "access_token": "eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9...",
  "token_type": "Bearer",
  "expires_in": 3600,
  "capabilities": [
    "liquid_read_resource",
    "liquid_query_sparql",
    "liquid_write_resource",
    "liquid_delete_resource",
    "liquid_list_container"
  ]
}
```

Method Visibility (Host MCP → LLM):

```
{
  "jsonrpc": "2.0",
  "result": {
    "tools": [
      {
        "name": "liquid_read_resource",
        "description": "Read RDF resource from Solid Pod",
        "inputSchema": {
          "type": "object",
          "properties": {
            "resourceUri": {"type": "string"},
            "format": {"enum": ["turtle", "jsonld", "ntriples"]}
          }
        }
      }
    ]
  },
  "id": 1
}
```

Audit Log Entry (Host MCP → Audit Log):

```
{
  "operation": "setup",
  "otp_token_hash": "sha256:a3f5e8d...",
  "webID": "https://pod.example/alice/profile#me",
  "timestamp": "2025-02-12T10:30:00Z",
  "hostMCP": "https://mcp-host.example/liquid",
  "authToken": "eyJhbGci...truncated"
}
```

C.3 OTP Token Security Properties

OTP tokens provide enhanced security through several mechanisms. Each token is single-use and expires after successful authentication or after a short timeout period (typically 5-10 minutes). The LLM never accesses the user's WebID directly, reducing the risk of WebID exposure through prompt injection or logging. The Host MCP maintains a secure mapping between OTP tokens and WebIDs, protected through encryption at rest and in transit. Audit logs record the hash of the OTP token rather than the token itself, preventing token reuse if logs are compromised.

D Query+CRUD Operation Details

D.1 Sequence Diagram

The sequence diagram is provided in Appendix F.1

D.2 Message Formats

MCP Tool Call (LLM → Host MCP):

```
{
  "jsonrpc": "2.0",
  "method": "tools/call",
  "params": {
    "name": "liquid_query_sparql",
    "arguments": {
      "podUri": "https://pod.example/alice/",
      "query": "SELECT ?condition ?date WHERE { :alice :hasCondition ?c . ?c a ?condition ; :diagnosedDate ?date }",
      "resultFormat": "json"
    }
  },
  "id": 3
}
```

HTTP Request (Host MCP → Solid Pod):

```
POST /alice/sparql HTTP/1.1
Host: pod.example
Authorization: Bearer eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9...
Content-Type: application/sparql-query
Accept: application/sparql-results+json

SELECT ?condition ?date WHERE {
  :alice :hasCondition ?c .
  ?c a ?condition ;
  :diagnosedDate ?date
}
```

SPARQL Response (Solid Pod → Host MCP):

```
{
  "head": {
    "vars": ["condition", "date"]
  },
  "results": {
    "bindings": [
      {
        "condition": {
          "type": "uri",
          "value": "http://example.org/Diabetes"
        },
        "date": {
          "type": "literal",
          "value": "2023-05-15",
          "datatype": "http://www.w3.org/2001/XMLSchema#date"
        }
      },
      {
        "condition": {
          "type": "uri",
          "value": "http://example.org/Hypertension"
        },
        "date": {
          "type": "literal",
          "value": "2024-01-10",
          "datatype": "http://www.w3.org/2001/XMLSchema#date"
        }
      }
    ]
  }
}
```

E Rights Exercise Operation Details

E.1 Sequence Diagram

The sequence diagram is provided in Appendix F.1.

E.2 Message Formats

Right of Access (Article 15) - MCP Tool Call:

```
{
  "jsonrpc": "2.0",
  "method": "tools/call",
  "params": {
```

```
    "name": "liquid_get_audit_log",
    "arguments": {
      "subjectUri": "https://pod.example/alice/profile#me",
      "timeRange": {
        "start": "2025-01-01T00:00:00Z",
        "end": "2025-02-12T23:59:59Z"
      }
    }
  },
  "id": 4
}
```

Right to Be Forgotten (Article 17) - MCP Tool Call:

```
{
  "jsonrpc": "2.0",
  "method": "tools/call",
  "params": {
    "name": "liquid_delete_resource",
    "arguments": {
      "resourceUri": "https://pod.example/alice/health/records.ttl",
      "recursive": false,
      "requester": "https://pod.example/alice/profile#me"
    }
  },
  "id": 5
}
```

F Protocol Sequence Diagrams

F.1 Setup Operation Sequence Diagram

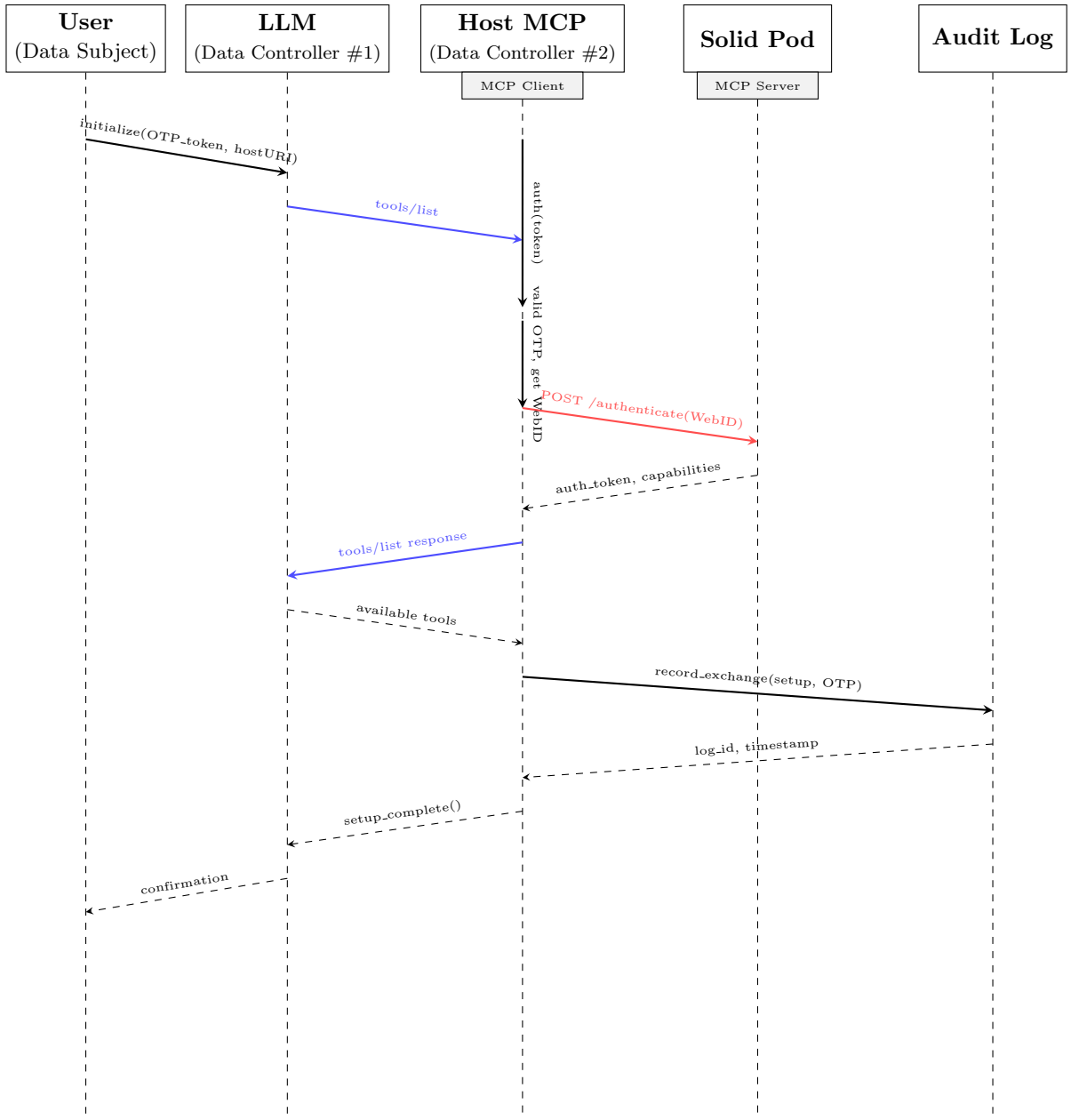

Figure 1: Setup Operation Protocol Flow. Blue arrows indicate MCP protocol messages, red arrows indicate HTTP protocol messages, black arrows indicate Liquid Protocol-specific messages.

F.2 Query+CRUD Operation Sequence Diagram

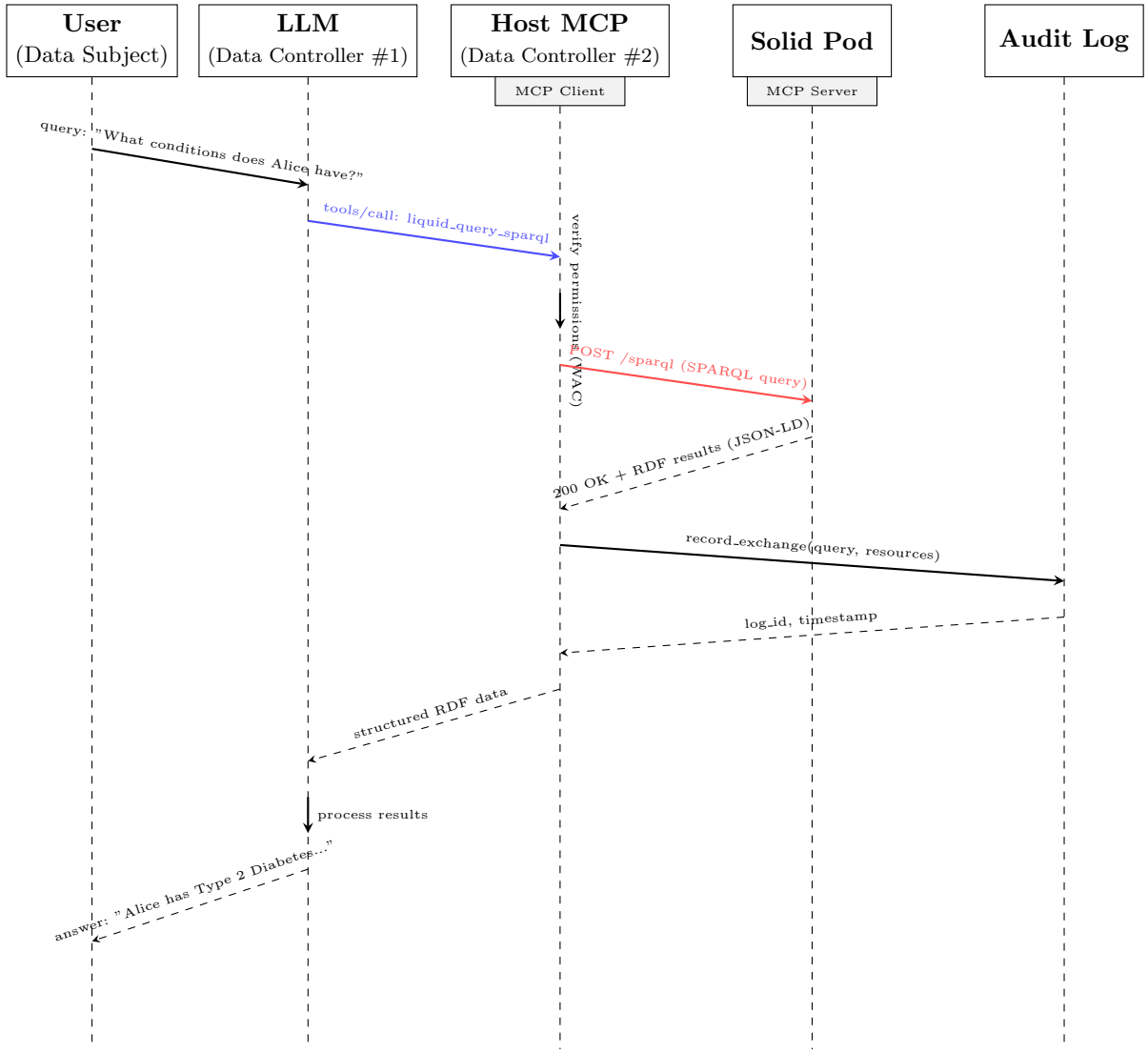

Figure 2: Query+CRUD Operation Protocol Flow. Blue arrows indicate MCP protocol messages, red arrows indicate HTTP protocol messages, black arrows indicate Liquid Protocol-specific messages.

F.3 Rights Exercise Operation Sequence Diagram

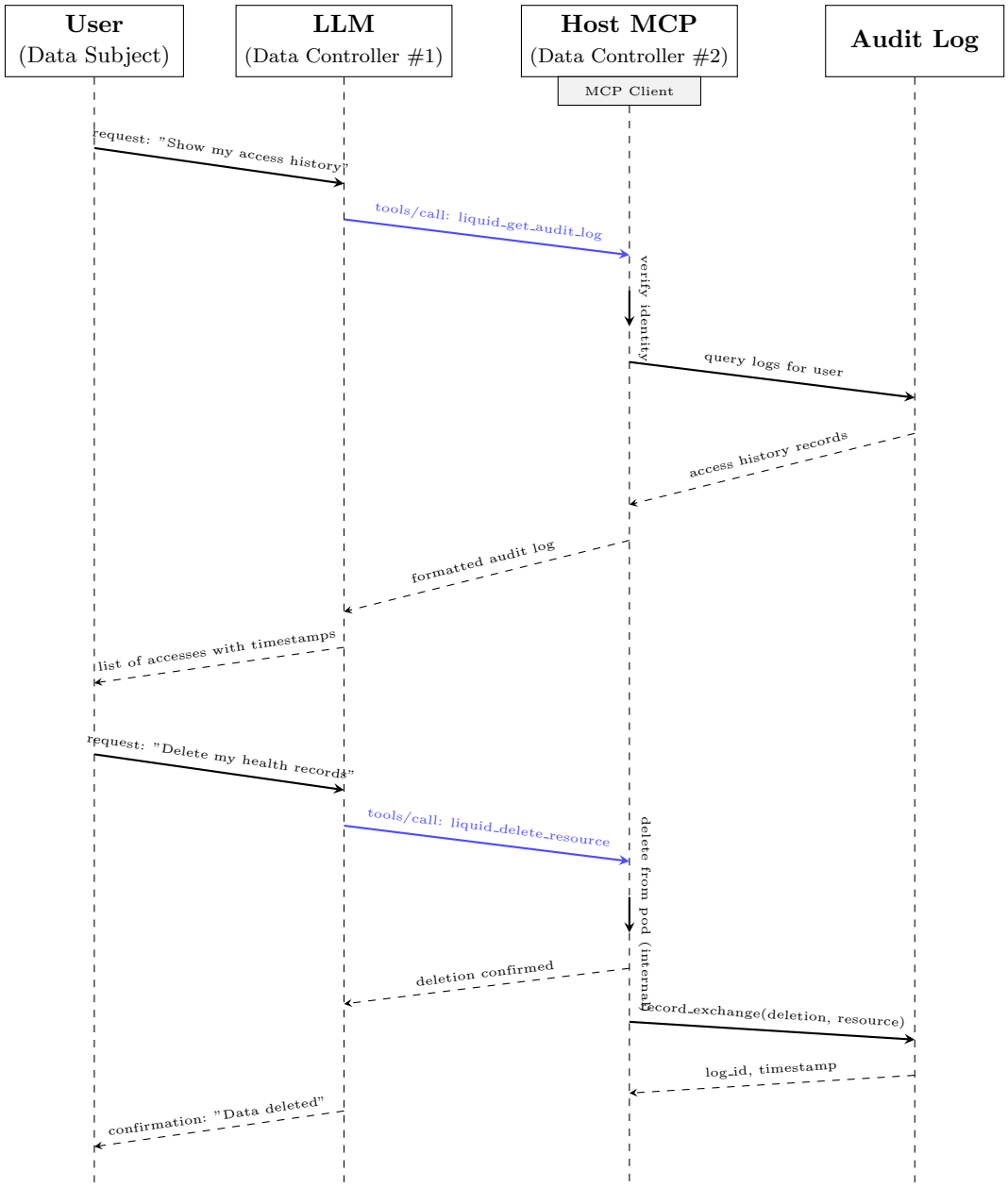

Figure 3: Rights Exercise Operation Protocol Flow (4 parties only). Blue arrows indicate MCP protocol messages, black arrows indicate Liquid Protocol-specific messages. The Host MCP handles Solid Pod operations internally.