Network Working Group
Internet-Draft
Intended status: Informational
Expires: 18 March 2026

B. Newbold
D. Holmgren
Bluesky Social

14 September 2025

Authenticated Transfer: Architecture Overview
draft-newbold-at-architecture-00

Abstract

Authenticated Transfer (AT) is a collection of protocol components that together provide a generic framework for interoperable social web applications, using global aggregations of interlinked, self-certifying data records.

This informational document provides an overview of the entire system, as implemented in late 2025. Some of those components may be in scope as work for the IETF, while other components may not. Many components are general-purpose and may find use outside of the context of AT. The intent of this document is to provide context for how all the components can fit together for certain use cases.

About This Document

This note is to be removed before publishing as an RFC.

Status information for this document may be found at https://datatracker.ietf.org/doc/draft-newbold-at-architecture/.

Source for this draft and an issue tracker can be found at https://github.com/bluesky-social/ietf-drafts.

Status of This Memo

This Internet-Draft will expire on 18 March 2026.

Table of Contents

1. Introduction and Background

Authenticated Transfer (AT) is a generic protocol framework for interoperable social web applications, using global aggregations of interlinked, self-certifying data records. Authority and authenticity of user-generated content are rooted in persistent user-controlled account identifiers, as opposed to network locations, which allows end users to migrate between hosting providers without impacting social graph links.

Applications interoperate by reusing data records hosted in public user repositories. Records often include references to records in other repositories. In aggregate, they form a global graph of interconnected structured data. Records conform to machine-readable data schemas (Lexicons). Lexicon schemas are also published as records and can be resolved as such. A composable annotation layer (labels) separates many aspects of content moderation from the operation of the underlying network infrastructure.

The design goals for the system are to enable “big world” social web applications with features and user experience competitive with modern commercial platforms, but with stronger end-user control over public identity and published data, and with no single party in exclusive control of any aspect of the overall network. As a principle, it should be possible to substitute providers for each service role in the network with minimal disruption, and barriers to entry for new providers should be minimized.

Users participate in the network by authenticating a client application to their current account hosting provider (Personal Data Server, or PDS). They publish content (including social interactions, for example, a “follow” or “emoji reaction”) by creating typed data records in their public data repository. Signed repository operations are broadcast by the PDS over a firehose event stream (a WebSocket). Repository operations are cryptographically authenticated, which allows intermediate services to aggregate and redistribute event streams. Indexing services (” AppViews”) receive event streams and provide opinionated, application-specific views and aggregations of the network, such as reaction counts or full text search. Client applications make authenticated API requests to AppViews and other network services on behalf of users. Moderation services publish signed annotations (labels), which may be aggregated by AppViews and included in API responses based on request headers provided by client applications.

The AT framework itself does not specify common social media conventions like "follows" or "avatars", leaving these to application-level schemas. Client developers can choose to reuse

existing schemas (and data) or declare their own. Use cases include traditional web document publishing (articles and pages), short-form video, collaborative coding, reviews, photo sharing, microblogging, business listings, etc.

2. Network Identity

Individual accounts in the AT network are referenced by both a permanent account identifier and a mutable "handle". The permanent identifier is intended for machine use and enables persistent references and social graph relationships. It is controlled by the end user and does not change if the account migrates between hosting providers. The handle is intended for display and human recognition.

Network identity may be pseudonymous and unlinked to offline identity. Users may have a single account and identity used across all applications, or multiple identities for distinct use-cases or personas.

2.1. Permanent Account Identifiers (DIDs)

Permanent account identifiers are Decentralized Identifier (DID), as specified in [W3CDID]. For the purpose of AT, DIDs are string identifiers (URIs) that be universally resolved to a DID document, which is a JSON document with a standard structure. The DID document contains identifier references ("alsoKnownAs"), public cryptographic key declarations ("verificationMethod"), and service endpoint declarations ("service"). The DID framework allows for many distinct resolution mechanisms ("methods"), each with their own design trade-offs around cost, latency, consistency, etc. Method types are indicated in the string identifier itself. To ensure interoperability, in the AT framework only a small number of "blessed" DID methods may be used; however new methods may be supported in the future as an evolvability mechanism.

The two supported DID methods are:

* did:web (eg, did:web:example.com), a W3C draft specification ([DIDWEB]) based on an HTTPS well-known endpoint. In the context of AT, path segments are not allowed.

* did:plc (eg, did:plc:ewvi7nxzyoun6zhxrhs64oiz), an independent specification ([DIDPLC]) based on self-authenticating operation logs, with authority rooted in rotatable cryptographic keys.

Control of an AT network account is rooted in control of the DID, over the full lifetime of the account. DID document contents can be changed, but the identifier (and thus DID method) can not be changed, merged, or split.

See Appendix A for AT-specific details.

2.2. Handles

Account handles are DNS hostnames. Handles must be bidirectionally validated: the handle must resolve to the DID, and the same DID must resolve to the handle. Accounts can have at most one valid handle at a time, but they can change. Handles are human-readable, intended for display in end-user applications, and may contain additional semantics, such as implying a degree of affiliation or control between an account and the party controlling a registered domain name. Handles are sometimes displayed with the @ prefix, like @handle.example.com.

DID documents reference a handle with a URI entry in the alsoKnownAs array. The format is the string at:// followed by the handle. Only the first value with the at:// prefix is considered.

Handles can resolve to DIDs using either of the following two mechanisms, both demonstrating control of the DNS hostname:

* a DNS TXT record at the hostname with the additional prefix part _atproto (for example, _atproto.handle.example.com for the handle example.com). The TXT record value is in key/value syntax with the key name did, like did=did:web:example.com.

* an HTTPS well-known endpoint at /.well-known/atproto-did. The response should have Content-Type text/plain, and include the full DID with no additional prefix or suffix.

If an account’s handle fails to validate for any reason, it should not be displayed to end users. Instead, the handle.invalid pseudo-handle may be used, or the account DID may be displayed.

3. Data Repositories

Public AT data records are stored in per-account data repositories. The repository data structure is a key/value content-addressed Merkle Search Tree (MST, originally described in [MST]), which functions as a key/value store. The top of the tree is referenced by a signed commit object, which includes a revision code to prevent accidental out-of-order processing. Commits, tree nodes, and data records are all serialized with a deterministic flavor of CBOR. Updates to

repositories, including a new commit, can be expressed as verifiable diff operations. Records may be deleted, leaving no long-term observable state in the repository, and deletion events are broadcast the same as other repository operations. Entire repositories can be serialized in a simple appended-blocks file format (CAR files).

Details of the repository structure and mechanisms for synchronizing repositories across the network are expanded in [ATREPO].

Media files, including images, audio, video, and long-form text, are not included directly in repositories but are instead stored as separate "blobs" and referenced by content hash, size, and content type.

Repositories can contain records of many schemas and for many distinct applications and use-cases. In live network use, they can contain anywhere from a handful to millions of distinct records, and have overall sizes of kilobytes to hundreds of megabytes. Individual CBOR records are usually hundreds to thousands of bytes in size. Service operators may enforce limits on the rate of repo operations and the size of individual records.

3.1. Data Model

Records are structured data objects that can be represented in both JSON and a deterministic profile of CBOR. The AT data model defines a subset of values and data structures that can be reliably round-tripped between the two serialization formats. JSON representation is used in web applications and HTTP APIs, while CBOR is used for hashing, signatures, and storage in repositories.

The deterministic CBOR encoding used is specified in Section 4.2 of [CBOR], with map key ordering following the original specification in Section 3.9 of [RFC7049] for historical compatibility. More encoding details are described in [ATREPO].

Some data types that have no direct representation in JSON are encoded as compound objects. For example, binary data is encoded in base64 using a reserved $bytes property:

```
{
  "exampleBytesField": {
    "$bytes": "nFERjvLLiw9qm45JrqH9QTzyC2Lu1Xb4ne6+sBrCzI0"
  }
}
```

Content references by hash can be expressed as strings without any special structure, or encoded as a binary link using a reserved $link property:

```
{
  "exampleLinkField": {
    "$link": "bafyreiff4hvcmjwrrhg7477umwkl7p2bwbppqxftfjw6sk56mp2lrk6cse"
  }
}
```

References to media files (blobs) are represented in both JSON and CBOR using a reserved blob object structure. Note that blob references can be parsed and identified out of generic record data without knowing the overall record schema.

```
{
  "exampleBlobField": {
    "$type": "blob",
    "ref": {
      "$link": "bafkreibjfgx2gprinfvicegelk5kosd6y2frmqpqzwqkg7usac74l3t2v4"
    },
    "mimeType": "image/jpeg",
    "size": 54321
  }
}
```

3.2. AT References (URIs)

In the context of an AT repository, keys (or paths) are simple strings with the format <collection>/<record-key>. For example, app.example.blog.post/3kghpsza2uu2j. The collection part describes the data schema of the record, and the record key identifies the specific record. Depending on the data schema, the record key may have a fixed value if only a single instance is expected (eg, self), a compact sortable timestamp format (called Timestamp Identifier, or TID), or a more open-ended or application-specific string syntax (which is still URL-safe).

Individual records can be globally referenced using a URI schema, at://. The structure of these references is:

at:// <did> / <collection> / <record-key>

For example:

at://did:plc:ewvi7nxzyoun6zhxrhs64oiz/app.example.blog.post/3kjbtzlhayo2p

3.3. Cross-Repository References

Records frequently reference other records through AT-URI references. The referenced record might reside in the same repository or any other repository in the network. Sometimes AT-URI references are complemented with the content hash of the target record to disambiguate the specific version being referenced, or to provide an integrity check.

Verifiable links between records across the network facilitate rich application semantics. To give an example from a comment thread use-case, when one account replies to a post authored by another account, the reply is created in the replying account's repository and includes a cross-repository AT-URI pointing to the original post. Application indexes (Section 4.3) that monitor both repositories can observe these references and construct complete interaction threads by correlating the original post with its replies.

3.4. Data Schemas

AT record data types can be described with a schema definition language named Lexicon. The Lexicon language is outside the scope of this overview document. It has many features and properties in common with JSON Schemas ([JSONSCHEMA]), but includes a global schema namespace system, an "open union" data type for third-party schema extensibility, and rules for evolving schemas over time without breaking existing data or applications.

Lexicon schemas are identified by Namespace Identifiers (NSIDs), which have authority rooted in the global DNS system. NSID syntax is to take a DNS hostname, reverse the order of parts, and then append a single additional name. The set of all NSIDs which differ only by the final part is referred to as a "group". For example, the NSID app.example.blog.post has the group app.example.blog and final name blog; it corresponds to the hostname blog.example.app.

AT specifies a mechanism for resolving NSIDs to public schema documents, using the AT system itself. The hostname corresponding to a group can have a DNS TXT record registered with the part prefix _lexicon, and value indicating a DID (with prefix did=, same as with handle resolution). The account indicated by the DID then publishes in their public repository a record with the collection com.atproto.lexicon.schema, and the record key is the full NSID of the schema. A complete AT Reference may look like:

```
at://did:plc:ewvi7nxzyoun6zhxrhs64oiz/com.atproto.lexicon.schema/app.example.blog.post
```

This provides a mechanism for validating data records of any type in the global network if needed. Most network services are not expected to resolve and validate schemas at runtime. Instead, the schema system allows independent parties to interoperate and reuse public data without explicit permission or prior coordination.

4. Network Architecture

The Network Identity and Data Repository components provide a mechanism for any party to authenticate data repositories that they may have obtained a copy of. This section of the document describes the current network architecture and service roles that facilitate the distribution of repository data. However, note that alternative transports and architectures are possible.

Transfer of repositories between any two hosts in the network uses the same synchronization protocol, regardless of the host roles in the network, with the client or "downstream" host initiating the transfer process (" pull" versus "push"). Details of the synchronization mechanism are described in [ATREPO].

A key principle of AT is that any party can resolve, fetch, and authenticate the full data repository for any account in the network at any time, without prior permission or authentication. Individual data records may be deleted at any time, and entire accounts may be deactivated (pausing redistribution) or deleted.

Most APIs in the network are simple JSON over HTTP. API endpoints can be specified using the Lexicon schema language, in a system called "XRPC". Such an endpoint specified by a schema named com.example.getContent could be requested at the path /xrpc/ com.example.getContent. Some of the endpoints that are part of the AT system itself are specified using XRPC. Using the XRPC system improves interoperability across organizational boundaries, but it is not mandatory for all network services.

4.1. Personal Data Server (PDS)

Each account in the network is hosted on a Personal Data Server (PDS), which may host one or many accounts. The PDS hosts the data repository, holds the signing key used to authenticate repository updates and external API requests, and often has enough control of the account's network identity to facilitate simple changes like handle updates.

PDS instances implement the public repository sync mechanism for all hosted accounts. Blobs, records, and entire repositories can be fetched by any client without authentication (though reasonable rate-

limits may be enforced). Updates to repositories, identities, and account status are all broadcast publicly over a "firehose" WebSocket, which any party can subscribe to.

Account migration between hosts is achieved by uploading a copy of the data repository and any blobs to a new provider, and then updating the network identity (DID document) to reference the new PDS host and newly generated signing key. Accounts are expected to have a semi-trusting relationship with their PDS provider. They may choose to maintain independent control of their network identity and backups of their account data to ensure that they can fully recover their accounts if the provider stops operation suddenly or unexpectedly violates trust.

Accounts on a PDS instance may be "active" or have one of several "inactive" statuses, including account deactivation, deletion, or takendown (by the host). Account status usually propagates "downstream" via broadcast messages on the firehose. Account status can be verified by querying the currently declared PDS instance.

4.2. Relay

Any party that wants to synchronize or observe the entire network could maintain direct connections to all known PDS instances. Maintaining hundreds or thousands of active subscriptions and preventing network abuse can require some operational oversight, so as a convenience subscribers can instead subscribe to a single relay instance.

Relays subscribe to all known PDS instances (or some subset of instance). They check every inbound message, apply rate-limits, and re-broadcast all valid messages on a aggregated firehose. The firehose stream format is the exact same as subscribing to an individual PDS instance.

Relays can discover new PDS instances by scraping identity directories for service endpoint declarations, or by querying an existing relay for a host list. PDS instances can also announce themselves to relay instances. Relay services may be provided publicly to enable network ecosystem growth, or they might be operated privately.

4.3. Application Indices

Providing featureful and performant web application experiences usually requires application-specific data indexing. Depending on the use-case, this might include full-text search indices, social graphs, numeric aggregations, timeseries, backlink indices, denormalized views, etc.

The AT network services which maintain these indices and provide network APIs to access them, are called "AppViews" because they provide an application-specific view or perspective of the overall network. Such services commonly subscribe to a full-network firehose provided by a relay and filter down to only the record types relevant to the application. In other cases, they may offload full synchronization to an external service which provides a filtered stream of relevant messages. If needed, services can backfill older data by enumerating relevant accounts in the network and fetching full repository exports from each account’s PDS.

AppViews might implement API endpoints with public Lexicon schemas, service authenticated requests from any account, and provide public unauthenticated API access. But they are not required to do any of these things.

4.4. Composable Services

Applications can provide extension points that allow many network services to fill a well-defined interface. The design pattern is for services to advertise their existence using a public declaration record that includes service metadata. The backing service is identified by a DID, and declares a service endpoint in the DID document matching a defined identifier fragment. Client applications indicate which service they would like to fulfill a request by indicating the declaration record by AT-URI in a request to an AppView.

For example, a music service could provide a composable extension point for recommendation services. Clients would make requests to the music AppView indicating which specific recommendation service should be used. The AppView resolves and queries the services for a set of candidates, and then hydrates those results with additional metadata and moderation labels before returning them to the client.

4.5. Client Applications

Clients are central to both the end-user experience and power dynamics in the network. They are downstream of other network services in terms of dataflow, but ultimately control feature sets, service provider defaults, content visibility, moderation features, most forms of monetization, etc. Major clients and the application indices and moderation services they rely on are expected to have mutually sustainable relationships, or even be operated by the same party. In contrast, clients may be relatively independent from PDS service providers.

Clients "write" to the public network by creating records to account repositories, and can "read" from the network either by requests to an application index API, or by directly reading records and blobs from PDS instances.

Clients may authenticate directly to account PDS instances and proxy API requests to other AT network services via the PDS, which adds Service Auth on behalf of the user (see Authentication section). Alternatively, clients may be more tightly integrated with an AppView (application index) and make requests to that service, which would only make requests on to the account’s PDS instance if necessary.

5. Moderation

Infrastructure-layer moderation is an unavoidable part of hosting user-generated content on the web. This means that all AT network services need mechanisms to take down content that is illegal in the physical jurisdiction of the service.

However, modern social web moderation addresses many behavioral patterns and harms beyond simple legality. Individual network services are operated by single parties, who may not be well-positioned to address many or all aspects of moderation. For example, a PDS operator may not be familiar with all of the application types used by hosted accounts. A design principle for the AT system is that, as much as possible, the role of moderation should be unbundled from the operation of infrastructure. This can be achieved using the composable labeling system described below, or via application-specific product features.

Client applications play a central role in network moderation. Clients have the closest relationship to end users, control distribution and display of content, have recognizable brands at state, and are most likely to be in a position to capture value through monetization. Design of product features, selection of infrastructure providers, and default configurations are all

decisions made by application developers. For these reasons, it is client applications that are expected to engage with and support any moderation services they depend on.

5.1. Labeling

Labels are a form of authenticated metadata about specific content and accounts in the AT network. They were originally developed as a composable system for content moderation, but they are not moderation-specific and have found alternative use-cases such as account badging and content curation.

Labels exist as free-standing, self-authenticated data objects, though they are also frequently distributed as part of AppView API responses. Additionally, label "values" may be directly embedded in records themselves ("self-labels"). Labels primarily consist of a source (DID), a subject (URI), and value. The value is a short string, similar to a tag or hashtag, which presumably has pre-defined semantics shared between the creator and consumer of the label. Additional metadata fields can give additional context, but at any point in time there should be only one coherent set of metadata for the combination of source, subject, and value. If there are multiple sets of metadata, the created-at timestamp is used to clarify which label is current. Automated labeling services generate too many labels for them to be published as records in repositories.

Labeling services (identified by a service DID) can provide a message stream of label updates, which can also be used to enumerate and backfill historical labels. Labels are ingested and indexed by AppViews, and can be hydrated into API responses as metadata, or in other cases used to trigger takedown actions.

Clients can indicate which labeling services they would like to have hydrated by using the Atproto-Accept-Labelers HTTP header on requests. The format of this header is a comma-separated array of DIDs. The AppView will include the Atproto-Content-Labelers HTTP header in responses, to indicate the set of labelers that were successfully resolved and hydrated into the response.

6. Authentication and Authorization

6.1. OAuth

AT includes an OAuth profile for client authentication and authorization to the user's host (PDS) as an auth server. The AT use-case is different from a traditional OAuth platform or identity provider (IdP) in that there are many independent hosting providers and many independent client applications in the network, and no single authority with whom to register (or revoke) clients.

The AT OAuth profile builds on the draft OAuth 2.1 Authorization Framework ([OAUTH21]), which means:

* only the "authorization code" OAuth 2.0 grant type is supported, not "implicit" or other grant types
* mandatory Proof Key for Code Exchange (PKCE, RFC 7636)
* security best practices (Best Current Practice for OAuth 2.0 Security and draft-ietf-oauth-browser-based-apps) are required

In addition:

* DPoP (with mandatory server issued nonces) is required to bind auth tokens to specific client software instances (eg, end devices or browser sessions)
* Pushed Authentication Requests (PAR) are used to streamline the authorization request flow

Clients use the draft OAuth Client ID Metadata Document ([OAUTHCLIENTID]) mechanism to use a public URL as a client_id, and publish a JSON metadata document at that URL. Users usually initiate the auth flow by providing a global account identifier (handle or DID), which the client resolves to an Auth Server (PDS).

6.2. Permissions

To describe permissions on PDS resources granted to OAuth client sessions, AT includes a custom authorization scope language. This includes a string syntax for including in OAuth client auth requests. AT-specific resources for which permissions can be requested include writing to the account's public repository (granular by record collection), uploading of media files (blobs), external API requests to third parties (granular by API endpoint and/or by service provider), network identity, and aspects of account hosting itself.

Clients may commonly request permissions to resources across many NSID namespaces. To address the separate issues of dynamic feature extension (new schemas) and overwhelmingly complex authentication requests, the system allows for the definition of "permission sets", which are bundles of permissions on related resources in the same namespace. These are published as Lexicon schemas and resolved at runtime by Auth Servers. They can include short summary descriptions for display to end users during the auth approval flow. They are constrained in scope to named resources under the same NSID domain authority, to prevent cross-domain privilege escalation.

6.3. Service Auth

Requests between AT network services can be authenticated using signed tokens (JWTs) as HTTP Authentication Bearer tokens. DIDs are used to identity individual services, and signatures can be verified by resolving the DID document and extracting the appropriate key. Multiple public keys can be published for distinct service types, and distinguished using identifier fragments. For example, did:web:api.example.com#example_service. If no fragment is included, #atproto is used.

The target service is indicated in the audience (aud) field, the Lexicon API endpoint being called is identified in the lxm field, and the requesting service (or account) is indicated in the issuer (iss) field.

7. Security Considerations

A detailed analysis of attack surfaces and security mitigations is out of scope for this overview document, but a few broad themes can be addressed.

AT public data repositories are for manifestly public data, which may be redistributed, archived, and enumerated individually or in bulk form by any party on the public internet.

Account identities may be pseudonymous, but hosting providers and client applications are likely to have visibility of identifying metadata, such as account emails and device IP addresses. Authenticated proxy requests to other services in the network may strip _some_ identifying metadata, but still include strong authentication.

Identity resolution requires care. The relationship between handles and DIDs must be verified bidirectionally. Cleartext DNS TXT resolution can be observed or manipulated on untrusted networks; clients on end-devices (including browser apps) should use DNS-over-HTTPS or similar techniques.

Hosting (and re-hosting) of user-uploaded media files raises many web platform security concerns. Account hosts should have restrictive CORS policies. Client apps (including OAuth client apps) should not be operated on the same origin as user-uploaded media files.

Hosting providers should not trust authenticated client requests, especially those resulting in proxied or other outbound network requests to arbitrary hosts (for example, identity resolution of arbitrary hostnames). Mitigations against SSRI and request amplification are essential.

Binary data formats must be treated as untrusted, including CBOR data objects and serialized CAR files. When parsing these formats, recursion limits and overall memory allocation quotas should be enforced.

8. IANA Considerations

The at:// URI scheme has a provisional registration. See https://www.iana.org/assignments/uri-schemes/uri-schemes.xhtml.

9. References

9.1. Normative References

[CBOR] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", STD 94, RFC 8949, DOI 10.17487/RFC8949, December 2020, <https://www.rfc-editor.org/rfc/rfc8949>.

[CONTROLLEDID] Longley, D., Sporny, M., Sabadello, M., Reed, D., Steele, O., and C. Allen, "Controlled Identifiers v1.0", May 2025, <https://www.w3.org/TR/2025/REC-cid-1.0-20250515/>.

[RFC7049] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", RFC 7049, DOI 10.17487/RFC7049, October 2013, <https://www.rfc-editor.org/rfc/rfc7049>.

[W3CDID] Sporny, M., Longley, D., Sabadello, M., Reed, D., Steele, O., and C. Allen, "Decentralized Identifiers (DIDs) v1.0", July 2022, <https://www.w3.org/TR/2022/REC-did-core-20220719/>.

9.2. Informative References

[ATPAPER] Kleppmann, M., Frazee, P., Gold, J., Graber, J., Holmgren, D., Ivy, D., Johnson, J., Newbold, B., and J. Volpert, "Bluesky and the AT Protocol: Usable Decentralized Social Media", December 2024, <https://doi.org/10.1145/3694809.3700740>.

[ATREPO] Holmgren, D. and B. Newbold, "Authenticated Transfer Repository and Synchronization", September 2025, <https://datatracker.ietf.org/doc/draft-holmgren-at-repository/>.

[DIDPLC] Holmgren, D., "did:plc Method Specification v0.1", May 2023, <https://web.plc.directory/spec/v0.1/did-plc>.

[DIDWEB] Gribneau, C., Prorock, M., Steele, O., Terbu, O., Xu, M., and D. Zagidulin, "did:web Method Specification (Draft)", July 2024, <https://w3c-ccg.github.io/did-method-web/>.

[JSONSCHEMA] Wright, A., Andrews, H., Hutton, B., and G. Dennis, "JSON Schema: A Media Type for Describing JSON Documents", June 2022, <https://datatracker.ietf.org/doc/draft-bhutton-json-schema/>.

[MST] Auvolat, A. and F. Taani, "Merkle Search Trees: Efficient State-Based CRDTs in Open Networks", October 2019, <https://inria.hal.science/hal-02303490/document>.

[OAUTH21] Hardt, D., Parecki, A., and T. Lodderstedt, "The OAuth 2.1 Authorization Framework", May 2025, <https://datatracker.ietf.org/doc/draft-ietf-oauth-v2-1/>.

[OAUTHCLIENTID] Parecki, A. and E. Smith, "OAuth Client ID Metadata Document", July 2025, <https://datatracker.ietf.org/doc/draft-parecki-oauth-client-id-metadata-document/>.

Appendix A. DID Documents

A DID document describes an account in the AT network if it includes the following:

* a handle declaration in the alsoKnownAs array (described in the "Handle" section below).

* an AT signing key under verificationMethod, with the id ending #atproto and the controller matching the DID of the document itself. The type should be "Multikey", as described in the Controlled Identifiers W3C standard ([CONTROLLEDID]). The supported key types are described in Appendix B.

* a service endpoint under service, with the id ending #atproto_pds and type matching "AtprotoPersonalDataServer". This must be a public HTTPS URL, with no path segment, indicating the current hosting provider of the account.

Appendix B. Cryptographic Keys

To maximize interoperability, AT supports only a small set of cryptographic systems and curve types. This set may expand slowly over time as an evolvability mechanism. The two elliptic curve types supported are:

* NIST P-256, aka "secp256r1"

* “secp256k1”, aka K-256

Unless otherwise specified, public keys are encoded in "Multikey" string encoding, as described in [CONTROLLEDID]. As specified in that document, P-256 keys are prefixed with the bytes 0x8024 before being encoded. For similar historical reasons, K-256 keys are prefixed with 0xE701. Compressed 32-byte binary encodings are used for both key types.

In ECDSA, both key types have malleable signatures. In the context of AT data signatures, only "low-S" signatures are considered valid. This does not apply when using the same keys in other contexts, such as JWT signatures.

Acknowledgements

A complete description of the Authenticated Transfer system was jointly published by the protocol design team in [ATPAPER]. Since that publication, Matthieu Sieben has contributed significantly. Aaron Goldman contributed to an earlier prototype.

The authors would like to sincerely thank the entire AT open source development community ("Atmosphere") for their feedback, commentary, creativity, and patience throughout the development of protocol.

The AT system was directly informed and influenced by existing P2P and federated social web technologies, in particular Secure Scuttlebutt, dat://, IPFS, and ActivityPub. The authors would like to thank the designers and developers of these protocols for their ongoing contributions to the field and openness to collaboration.

Authors' Addresses

Bryan Newbold
Bluesky Social
Email: bryan@blueskyweb.xyz

Daniel Holmgren
Bluesky Social
Email: daniel@blueskyweb.xyz