

# JACS: JSON Agent Communication Standard

hai.ai

[hai.ai](https://hai.ai)

AI Alliance, June 12, 2025

Human Assisted Intelligence, Inc © 2025

[jonathan@hai.io](mailto:jonathan@hai.io)

<https://www.linkedin.com/in/jonathanhendler/>

# About me

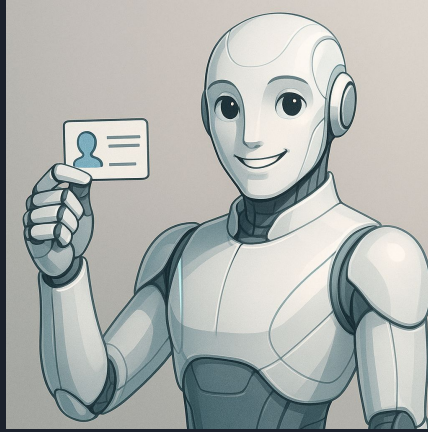
Small startup backend generalist.

Cofounded EdTech company, worked mental health care  
Reading papers on neural nets in 2001 while backpacking in  
Asia, before I went to college for CS. Lots of hacking Semantic  
Web, NLP, and deep learning trying create structured data  
from the web before LLMs/Transformers.



<https://www.linkedin.com/in/jonathanhendler/>

# Your Agent needs an ID <sup>TM</sup>



# Use cases

1. A file is sitting on a server. Where did it come from? Who has access to it? (e.g. email, shared docs)
2. An MCP server gets a request from an unknown agent, the oauth flow doesn't guarantee the identity of the client after the initial handshake.
3. A document is modified by multiple human and AI collaborators. Which one is latest, correct version? Did the collaborators agree?

# Features

1. **Embeddable library** in multiple programming languages to sign JSON or any binary files by attaching a JACS header
2. **JSON Schemas** for common agent use cases
3. **Authentication middleware** for http, mcp
4. **Observability middleware** for https, mcp

# JACS Origin Story

Had an email project using AI for my startup two years ago and found shortcomings validating content across systems. DNS/SPF, PGP, left gaps.

Also, I thought about agents.txt, and how web content and http requests also have very little ability to identify and manage clients and I'd need to also find a new solution here as well. ARC is email only. DKIM is domain only.

With chat taking over, email seemed less important overall. I needed something new.

JACs makes identity, authz, provenance easier.

- OAuth 2/OpenID
- JWT
- PGP
- DKIM, SPF
- DPKI and blockchain
- W3C DID
- SAML
- mTLS, TLS, x509
- Kerberos
- FIDO2/WebAuthn/Passkeys
- ARC
- DPoP
- Checksums (RFC 6249., RFC 2068, RFC 3230, new HTTP Digest-Headers)

# JACS for Auth

Making it easy to set up trusted identity

- Works in both the MCP server and MCP client
- Every request is verified in source identity and content
- Business logic can be built around decentralized identity



# Easy MCP and Web Auth

1. Rust CLI `$cargo install jacs`
2. CLI `$jacs init`
3. Python `$pip install jacs`
4. Node `$npm install jacs`

Python MCP Server `mcp = JACSMCPServer(FastMCP("Authenticated Echo Server"))`

Python MCP Client `client = JACSMCPClient(server_url)`

# Tech and Features

- Rust lib used in Python and Typescript
- JSON Schema <https://json-schema.org/>
- hashing and signing libraries: supports RSA, Ring ED25519 and (experimental) post quantum via dilithium
- Observability with Open Telemetry
- RBAC
- Data Lake integrations

# Roadmap

1. PKI solution
2. Integration with A2A, physical devices
3. RBAC middleware for http
4. RBAC for Data Lakes, Filesystems, and Databases
5. full data lake solution

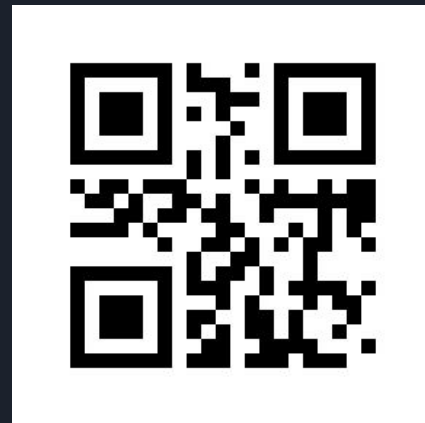
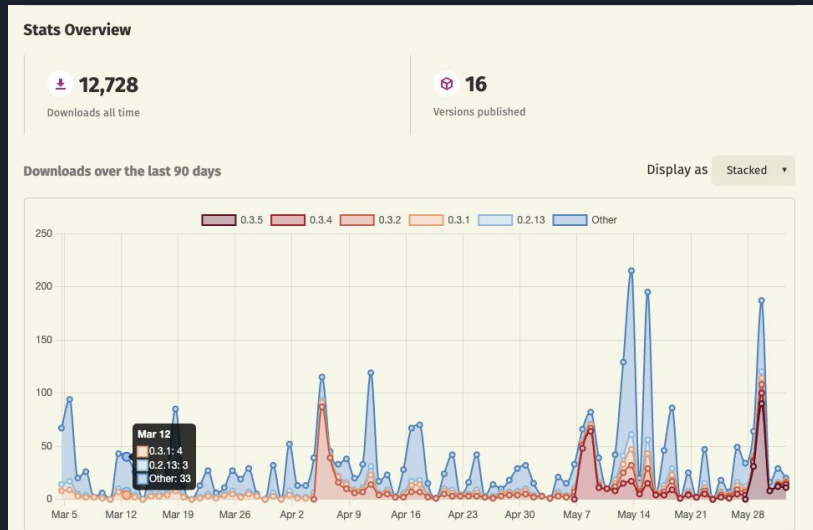
# What JACS needs today

1. Users - integrations, use cases
2. Contributors - extensions, modularity
3. Strategic Partners - paths to adoption

# Open Source

<https://github.com/HumanAssisted/JACS>

<https://crates.io/crates/jacs>



# Verify this document

View the JACS header file: <https://raw.githubusercontent.com/HumanAssisted/JACS/refs/heads/main/jacs/presentation/aialliance.presentation.jacs.json>

Download document: <https://raw.githubusercontent.com/HumanAssisted/JACS/refs/heads/main/jacs/presentation/aialliance.presentation.pdf>

Download public key: [https://raw.githubusercontent.com/HumanAssisted/JACS/refs/heads/main/jacs/presentation/jacs\\_keys/aialliance.presentation.key.pub](https://raw.githubusercontent.com/HumanAssisted/JACS/refs/heads/main/jacs/presentation/jacs_keys/aialliance.presentation.key.pub)

JACS header was created with:

```
$ JACS_PRIVATE_KEY_PASSWORD=hello jacs document create -v --attach aialliance.presentation.pdf -e false -o  
aialliance.presentation.jacs.json -a  
jacs/agent/08a79c8b-464c-41fb-b071-937e6543871d\:ba3cf18d-60f1-4fc7-bf0a-d1ec517ccbbe.json
```

From the jacs cli, verify with:

```
$JACS_PRIVATE_KEY_PASSWORD=hello jacs document verify -v -f jacs/aialliance.presentation.jacs.json -a  
jacs/agent/08a79c8b-464c-41fb-b071-937e6543871d\:ba3cf18d-60f1-4fc7-bf0a-d1ec517ccbbe.json
```

# License

Apache 2.0 - with Common Clause. (*Considering pure Apache 2.0*)

<https://commonsclause.com/>

# Easy MCP and Web Auth

1. Rust CLI `$ cargo install jacs`
2. CLI `$ jacs init`
3. Python `$ pip install jacs`
4. Node `$ npm install jacs`

## Python MCP Server

```
jacs_config_path = current_dir / "jacs.server.config.json"
os.environ["JACS_PRIVATE_KEY_PASSWORD"] = "hello"
jacs.load(str(jacs_config_path))
mcp = JACSMCPServer(FastMCP("Authenticated Echo Server"))
```

## Python MCP Client

```
client = JACSMCPClient(server_url)
```