GetChkd Identity Provider (IDP) Service – Phase 1

The **GetChkd Identity Provider** is a secure, standards-compliant identity and access management (IAM) system. This Phase 1 release focuses on **JWT-based token issuance for OAuth2 and OpenID Connect (OIDC)**, and **SAML 2.0 token issuance for federated identity** via a unified `/token` endpoint. It enables secure, auditable, and extensible access control across enterprise systems, with support for Salesforce and Splunk.

> This project is proprietary and not intended for public distribution. For licensing, contact *`legal@getchkd.com`*.

⚡ Quick Start

`bash git clone https://gitlab.com/getchkdgroup/idp-service.git cd idp-service chmod +x ./scripts/generate-idp-keys.sh ./scripts/generate-tls-cert.sh ./scripts/generate-idp-keys.sh ./scripts/generate-tls-cert.sh cp scripts/set_idp_env.example.sh scripts/set_idp_env.sh source scripts/set_idp_env.sh go run cmd/main.go`

Swagger UI: http://localhost:8080/swagger/index.html

Delivered Features (Phase 1)

- OAuth2 Access Tokens (JWT) wi `scope`, `aud`, `azp`, `iat`, `exp`, `auth_time`
- OpenID Connect (OIDC) ID Tokens with standard claims

- SAML 2.0 Assertions issued v `/token` with provider-specific mappings (Salesforce, Splunk)
- OIDC discovery and JWKS endpoint
- OAuth2-compliant Token Introspectio `/introspect`) for OIDC and OAuth2 tokens only
- RSA-based JWT & SAML assertion signing (SHA-256)
- Swagger UI for API documentation
- `/healthz` and `/readiness` endpoints for Kubernetes probes
- Fixed window in-memory request rate limiting
- Hexagonal Architecture for dependency inversion and separation of concerns

> Current Ve *`v0.7.5`* – Phase 1 complete.
> Future phases will focus on identity federation and delegated SSO workflows.

Project Structure

This project follows a **Hexagonal Architecture**, a pattern used to promote clear **dependency management** and **separation of concerns**. This allows clean isolation between application core logic and external systems like HTTP interfaces, token signers, or persistence layers.

```
idp-service/ ├── cmd/ # Main application entry point ├──
config/ # Static configuration (keys, certs, etc.) ├── docs/
# API documentation (Swagger, Postman) ├── internal/ # Core
Hexagonal architecture │ ├── adapter/ # HTTP handlers,
middleware, DTOs │ ├── app/ # CQRS commands and handlers │
├── domain/ # Business models and interfaces │ ├──
infrastructure/ # Signing, persistence, external services │
├── util/ # Shared utility functions │ └── test/ # Test
helpers for integration/unit tests ├── scripts/ # Environment
setup and helper scripts ├── test-results/ # Output folder
for test artifacts ├── Dockerfile # Production-ready
```

```
container specification ├── docker-compose.yml # Local
service orchestration ├── THIRD_PARTY.md # OSS license and
dependency attribution ├── README.md # Project documentation
```

API Endpoints

Token Issuance

Endpoint	Method	Content-Type	Description
`/token`	POST	`application/json`	Issues OAuth2, OIDC, or SAML tokens

> Currently supported SAML recipients **Salesforce** and **Splunk**. More providers can be added in future phases.

Discovery & Validation

Endpoint	Method	Content-Type	Description
`/.well-known/openid-configuration`	GET	`application/json`	OIDC metadata discovery
`/jwks.json`	GET	`application/json`	Public signing keys (JWK set)
`/introspect`	POST	`application/x-www-form-urlencoded`	Validates OIDC and OAuth2 tokens only

Health & Status

Endpoint	Method	Content-Type	Description
`/healthz`	GET	`application/json`	Liveness probe (`200 OK` if alive)
`/readiness`	GET	`application/json`	Returns `"Ready"` or `"Not Ready"`

Token Request Examples

OIDC Token

```http POST /token Content-Type: application/json

{ "token_type": "oidc", "oidc": { "sub": "user-123", "aud": "caldy-client-id", "scope": "openid" } } ```

## OAuth2 Token

```http POST /token Content-Type: application/json

{ "token_type": "oauth2", "oauth2": { "sub": "user-123", "aud": "https://api.example.com", "scope": "read" } } ```

## SAML Token (Salesforce)

```http POST /token Content-Type: application/json

{ "token_type": "saml", "saml": { "sub": "jrw@belltane.com", "audience": "https://saml.salesforce.com", "recipient": "https://poc-getchkd-dev-ed.develop.my.salesforce.com" } } ```

## SAML Token (Splunk)

```http POST /token Content-Type: application/json

{ "token_type": "saml", "saml": { "sub": "jrw@belltane.com", "audience": "https://splunkcloud.com", "recipient": "https://poc-getchkd-dev-ed.develop.splunkcloud.com/saml/acs" } } ```

---

# OIDC Token Response Example

`json { "id_token": "eyJraWQiOiJrZXktMSIsInR5cCI6IkpXVCIsImFsZyI6IlJTMjU2In0...", "token_type": "Bearer", "expires_in": 3600 }`

### OAuth2 Token Response Example

`json { "access_token": "eyJraWQiOiJrZXktMiIsInR5cCI6IkpXVCIsImFsZyI6IlJTMjU2In0...", "token_type": "Bearer", "expires_in": 3600, "scope": "read" }`

### SAML Token Response Example

`json { "assertion": "<saml:Assertion xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion" ... />", "format": "urn:oasis:names:tc:SAML:2.0:bindings:HTTP-POST" }`

### OpenID Discovery Response Example

`json { "issuer": "https://idp.example.com", "authorization_endpoint": "https://idp.example.com/auth", "token_endpoint": "https://idp.example.com/token", "jwks_uri": "https://idp.example.com/jwks.json", "response_types_supported": ["code", "id_token", "token id_token"], "subject_types_supported": ["public"], "id_token_signing_alg_values_supported": ["RS256"] }`

### JWKS JSON Response Example

`json { "keys": [ { "kty": "RSA", "kid": "key-1", "use": "sig", "alg": "RS256", "n": "....", "e": "AQAB" } ] }`

---

## Discovery & Introspection Requests

`http GET /.well-known/openid-configuration Accept: application/json`

`http GET /jwks.json Accept: application/json`

```` ```http POST /introspect Content-Type: application/x-www-form-urlencoded ````
````

token= ```

Response: `json { "active": true, "sub": "user-123", "scope": "openid", "exp": 1718458214, "iat": 1718454614, "aud": "caldy-client-id", "token_type": "id_token" }`

Security Notes

- All tokens (OIDC, OAuth2, and SAML) are signed using **RSA SHA-256**
- All issued OAuth2 and OIDC tokens are **JWT-based**. Opaque tokens are **not** supported.
- Only **OIDC and OAuth2 tokens** are introspectable; **SAML tokens are not**
- Only **JWT-formatted tokens** are accepted at `/introspect`; opaque tokens are rejected.
- Claims included: `iat`, `exp`, `auth_time`, `aud`, `azp` (optional)
- Expiry enforcement and clock skew validation are implemented
- Key material is stored under `./config/keys/` and should be secured with Vault/KMS in production

---

Logging & Middleware

- Logging via **Zap** (structured logger); controlled via `LOG_LEVEL`
- Middleware includes:
  - GZIP compression
  - Secure headers
  - Rate limiting (fixed window)
  - Request ID injection
  - Request payload size limit (1MB default)

`bash export LOG_LEVEL=debug`

Installation

Ensure the script is executable and run it:

`bash chmod +x ./scripts/generate-idp-keys.sh ./scripts/generate-idp-keys.sh`

This will create an RSA key pair and a self-signed certificate in `./config/keys/`.

---

2 Generate TLS certificate (for HTTPS)

`bash chmod +x ./scripts/generate-tls-cert.sh ./scripts/generate-tls-cert.sh`

This will create a self-signed TLS certificate in `./config/certs/`.

---

3 Configure environment variables

`bash cp scripts/set_idp_env.example.sh scripts/set_idp_env.sh`

Open `scripts/set_idp_env.sh` and fill in the required values. Then export:

`bash source scripts/set_idp_env.sh`

> Note: **Do not** use a *`.env`* file. This project intentionally avoids *`.env`* files in line with best practices and security policies.

▶ Running the Service

`bash docker compose up --build`

Swagger UI: http://localhost:8080/swagger/index.html

Development & Testing

To run **unit** tests locally:

`bash go test ./...`

To run **integration** tests locally:

`bash go test -tags=integration ./...`

Ensure your environment variables are sourced before running tests.

License

References

- OAuth 2.0 (RFC 6749)
- OpenID Connect Core 1.0
- SAML 2.0 Core

Postman Collection

For valid request/response formats for `/token`, `/introspect`, and discovery endpoints:

1. Open Postman → **Import**
2. Select `docs/postman/GetChkd.postman_collection.json`

3. Browse `/token`, `/introspect`, and discovery groups for usage
4. Intended for developer testing and validation