boosting digital adoption

TSI Privacy Vault

User Guide

5th July 2025
V1.0

TSI Tech Solutions Cooperative Foundation
A section 8 company

https://tsicoop.org

Table of Contents

Introduction

TSI Privacy Vault solution is to provide a comprehensive, secure, and compliant solution for organizations to manage sensitive personal data, with a primary focus on Digital IDs like Aadhaar ID, Voter ID and ABHA ID.

Note: This project is under Active Development. Today, we keep your Digital IDs safe. Tomorrow, we're becoming a complete 'digital safe' for all the private info you handle—from customers and employees to partners.

High Level System Goals

Ensuring Data Security and Privacy: The primary goal is to provide an exceptionally secure environment for sensitive Indian identification data, particularly Aadhaar numbers, preventing unauthorized access, misuse, and data breaches. This includes implementing strong encryption, isolated storage, and robust key management.
Achieving Regulatory Compliance: To fully adhere to and enable compliance with UIDAI (Unique Identification Authority of India) guidelines regarding Aadhaar data storage and usage, as well as other relevant data protection regulations for other ID types (Voter ID, ABHA ID). This includes maintaining comprehensive audit trails.
Facilitating Secure Integration and Reduced Risk: To offer a reliable and secure mechanism for organizations to integrate sensitive ID data into their business processes while significantly minimizing the "surface area" of risk. This is achieved by centralizing sensitive ID storage and promoting the use of non-identifiable reference keys across other systems.
Providing a Scalable and Extensible Solution: To create a secure digital safe that isolates and protects your organisations' private data for effortless privacy compliance.

Functional Design

Client Management: This module provides the capability to onboard and manage external applications or services that need to interact with the vault. It enables the registration of new API clients, securely issuing unique API keys and secrets for authentication. This ensures that only authorized entities can access the vault's sensitive operations.
Secure ID Vaulting: This is the central function, allowing the secure storage and retrieval of identification numbers.

- **Store ID:** Authorized clients can submit an identification number (specifying its type) to the vault. The system encrypts this ID and generates a unique,

non-identifiable reference key (token), which is returned to the client for use in their internal systems.
- **Retrieve ID by Reference:** Clients can present a reference key to the vault, which then retrieves the corresponding encrypted ID, decrypts it securely, and returns the original identification number.
- **Retrieve Reference by ID Value:** For specific scenarios, the system also supports looking up a reference key when provided with an ID type and the actual identification number.

Auditing: Every significant operation performed on the vault, including storing and fetching identification numbers, is meticulously logged with details such as the accessing client, operation type, ID type, and a timestamp. This comprehensive logging ensures accountability, supports compliance requirements, and provides a critical trail for security monitoring and incident response.

Admin App

The Admin App provides a centralized web-based interface for authorized administrators to manage and monitor the secure ID vault system. Its primary functionality revolves around two key areas: **Client Management** and **System Oversight**. Within Client Management, administrators can onboard new API clients by registering them and obtaining their unique API keys and secrets. It also enables granular control over existing clients, allowing for viewing detailed client information, activating or deactivating their access, and securely regenerating API secrets when necessary, ensuring robust access control to the vault.

Beyond client management, the portal offers crucial System Oversight capabilities. Administrators can view and configure supported ID types (e.g., Aadhaar, Voter ID, ABHA ID), including their validation rules. Most importantly, the portal provides comprehensive access to the system's **Audit Logs**, allowing administrators to monitor all interactions with the vault. This includes filtering and searching for specific operations by date, client, ID type, or reference key, which is vital for security monitoring, compliance auditing, and forensic analysis in case of any suspicious activity.

Note: Admin App is currently unavailable. It will be developed in the future. For now, use the Admin APIs via Postman / CURL.

Admin API

1. Admin Authentication

Authentication: All Admin Portal APIs (except Register Admin & Admin Login) need a secure **Bearer Token** authentication mechanism, where an admin user first logs in to get a token, and this token is then sent in the Authorization: Bearer <token> header for all subsequent requests.

1.1 Register Admin

Allows a new administrative user to be registered in the system. This endpoint is typically highly secured and might only be callable by existing super-administrators or during initial system setup, rather than being publicly exposed.

- **URI:** /api/admin/register
- **HTTP Method:** POST

Request:

JSON

```
None
{
 "_func": "register_admin",
 "username": "admin",
 "password": "admin@123",
 "email": "admin@tsicoop.org",
 "role": "SYSTEM_ADMIN"
}
```

Notes on Request:

- username: Unique identifier for the admin user.
- password: Should be a strong password, which will be hashed and salted before storage in the database.
- email: Email address for the admin user
- role: The role assigned to the new admin (e.g., "SYSTEM_ADMIN", "AUDIT_VIEWER", "CLIENT_MANAGER"). This implies a Role-Based Access Control (RBAC) system.

Response (201 Created):

JSON

```
None
{"_created":true,"userid":1}
```

Errors:

- **400 Bad Request:** (e.g., username already exists, weak password, invalid role)
  JSON
- **500 Internal Server Error:** (e.g., unexpected server issue)
  JSON

1.2 Admin Login

Allows an administrator to authenticate and obtain an authorization token.

- **URI:** `/api/admin/login`
- **HTTP Method:** `POST`
- **Authorization Header:** NOT REQUIRED

Request:

JSON

```
None
{
 "_func": "admin_login",
 "username": "admin",
 "password": "admin@123"
}
```

Response (200 OK):

JSON

```
None
{
  "_success": true,
  "role": "SYSTEM_ADMIN",
  "message": "Login successful",
  "email": "admin@tsicoop.org",
  "username": "admin",
  "token":
"eyJhbGciOiJIUzI1NiJ9.eyJyb2xlIjoiU1lTVEVNX0FETUlOIiwibmFtZ
SI6ImFkbWluIiwic3ViIjoiYWRtaW5AdHNpY29vcC5vcmciLCJpYXQiOjE3
NTI1MDA4OTQsImV4cCI6MTc1MzM2NDg5NH0.H2E11JTIGShGRm2KYtY3AjM
pK7k_vVLWuoQtfkuhKBI"
}
```

Errors:

- **401 Unauthorized:** Invalid credentials.
- **500 Internal Server Error:** Unexpected error.

2. Admin Client Management

2.1 Get All Clients

Retrieves a list of all registered API clients.

- **URI:** `/api/admin/clients`
- **HTTP Method:** `POST`
- **Authorization Header:** `Authorization: Bearer <token>`

Request:

JSON

```
{
 "_func": "get_all_clients"
}
```

Response (200 OK):

JSON

None

```
[
  {
    "apiKey": "ext-0f2e0522-c5b9-4eb9-a6db-dadb39f6c944",
    "clientName": "MyWebAppClient5",
    "active": true,
    "createdDatetime": "2025-07-14T13:25:08.094597"
  }
]
```

Errors:

- **401 Unauthorized:** Invalid or missing token.
- **500 Internal Server Error:** Unexpected error.

2.2 Get Client Details

Retrieves detailed information for a specific API client.

- **URI:** `/api/admin/clients`
- **HTTP Method:** `POST`

- **Authorization Header:** `Authorization: Bearer <token>`

Request:

```
JSON
{
 "_func": "get_client_details",
 "api_key": "ext-0f2e0522-c5b9-4eb9-a6db-dadb39f6c944"
}
```

Response (200 OK):

JSON

```
None
{
  "apiKey": "ext-0f2e0522-c5b9-4eb9-a6db-dadb39f6c944",
  "clientName": "MyWebAppClient5",
  "active": true,
  "apiSecret": "ext-6aba6d69-6d51-49f7-8163-972c82f9bfaf",
  "createdDatetime": "2025-07-14T13:25:08.094597"
}
```

Errors:

- **401 Unauthorized:** Invalid or missing token.
- **404 Not Found:** Client with the given API key does not exist.
- **500 Internal Server Error:** Unexpected error.

2.3 Update Client Status

Activates or deactivates an existing API client.

- **URI:** `/api/admin/clients`
- **HTTP Method:** `POST`
- **Authorization Header:** `Authorization: Bearer <token>`

Request:

JSON

```
None
{
 "_func": "update_client_status",
 "api_key": "ext-0f2e0522-c5b9-4eb9-a6db-dadb39f6c944",
 "active": true
}
```

Response (200 OK):

JSON

```
None
{"apiKey":"ext-0f2e0522-c5b9-4eb9-a6db-dadb39f6c944","active":true}
```

Errors:

- **400 Bad Request:** Invalid request body.
- **401 Unauthorized:** Invalid or missing token.
- **404 Not Found:** Client with the given API key does not exist.
- **500 Internal Server Error:** Unexpected error.

2.4 Generate New Client Secret

Generates a new API secret for an existing client, invalidating the old one.

- **URI:** `/api/admin/clients`
- **HTTP Method:** `POST`
- **Authorization Header:** `Authorization: Bearer <token>`

Request:

```JSON
{
 "_func": "generate_new_client_secret",
 "api_key": "ext-0f2e0522-c5b9-4eb9-a6db-dadb39f6c944"
}
```

Response (200 OK):

JSON

```None
{

"apiKey":"ext-0f2e0522-c5b9-4eb9-a6db-dadb39f6c944",

"apiSecret":"3e9304e6-a9c7-4457-9dec-5354aefb6cef"

}
```

Errors:

- **401 Unauthorized:** Invalid or missing token.
- **404 Not Found:** Client with the given API key does not exist.
- **500 Internal Server Error:** Unexpected error.

3. Admin ID Type Management

3.1 Get All ID Types

Retrieves a list of all configured identification types.

- **URI:** `/api/admin/id-types`
- **HTTP Method:** `POST`
- **Authorization Header:** `Authorization: Bearer <token>`

Request:

```JSON
{
 "_func": "get_all_id_types"
}
```

Response (200 OK):

JSON

```None
{

  "idTypeName": "Aadhaar Number",

  "description": "Unique Identification Number issued by UIDAI.",

  "active": true,

  "validationRegex": "^[2-9]{1}[0-9]{3}\\\\s[0-9]{4}\\\\s[0-9]{4}$",

  "idTypeCode": "AADHAAR"

}
```

Errors:

- **401 Unauthorized:** Invalid or missing token.
- **500 Internal Server Error:** Unexpected error.

3.2 Update ID Type

Updates the details of an existing identification type.

- **URI:** `/api/admin/id-types`
- **HTTP Method:** `POST`
- **Authorization Header:** `Authorization: Bearer <token>`

Request:

JSON

```
None
{
 "_func": "update_id_type",
 "idTypeCode": "ABHAID",
 "idTypeName": "Abha ID",
 "description": "Health ID issued by ABDM.",
 "validationRegex":
"^[2-9]{1}[0-9]{3}\\\\s[0-9]{4}\\\\s[0-9]{4}$",
 "active": true
}
```

Response (200 OK):

JSON

```
None
{
  "idTypeName": "Abha ID",
  "description": "Health ID issued by ABDM.",
  "active": true,
  "validationRegex":
"^[2-9]{1}[0-9]{3}\\\\s[0-9]{4}\\\\s[0-9]{4}$",
  "idTypeCode": "ABHAID"
}
```

Errors:

- **400 Bad Request:** Invalid request body or data.
- **401 Unauthorized:** Invalid or missing token.
- **404 Not Found:** ID type with the given code does not exist.
- **500 Internal Server Error:** Unexpected error.

4. Admin Audit Log Management

4.1 Get Audit Logs

Retrieves system audit logs based on specified filters.

- **URI:** `/api/admin/audit-logs`
- **HTTP Method:** `POST`
- **Authorization Header:** `Authorization: Bearer <token>`

Request:

```
JSON
{
```

```
 "_func": "get_audit_logs",
 "startDate": "",
 "endDate": "",
 "apiKey": "",
 "operationType": "",
 "idType": "",
 "referenceKey": "",
 "page": 1,
 "size": 10
}
```

Response (200 OK):

JSON

```
None
{
  "number": 1,
  "last": true,
  "numberOfElements": 3,
  "size": 10,
  "totalPages": 1,
  "pageable": {
    "pageNumber": 1,
    "pageSize": 10
```

```
  },
  "content": [
    {
      "idType": "AADHAAR",
      "apiKey": "ext-73fe5d43-80c0-4eaf-a331-c453d22e3e85",
      "clientName": "MyWebAppClient6",
      "logId": 3,
      "operationType": "FETCH",
      "referenceKey":
"5392a2a9-b904-454e-8d4d-c0379fb31277",
      "logDatetime": "2025-07-14T14:49:59.999137"
    },
    {
      "idType": "AADHAAR",
      "apiKey": "ext-73fe5d43-80c0-4eaf-a331-c453d22e3e85",
      "clientName": "MyWebAppClient6",
      "logId": 2,
      "operationType": "FETCH",
      "referenceKey":
"5392a2a9-b904-454e-8d4d-c0379fb31277",
      "logDatetime": "2025-07-14T14:46:29.460484"
```

```
    },
    {
      "idType": "AADHAAR",
      "apiKey": "ext-73fe5d43-80c0-4eaf-a331-c453d22e3e85",
      "clientName": "MyWebAppClient6",
      "logId": 1,
      "operationType": "STORE",
      "referenceKey":
"5392a2a9-b904-454e-8d4d-c0379fb31277",
      "logDatetime": "2025-07-14T14:41:49.435654"
    }
  ],
  "first": false,
  "totalElements": 3,
  "empty": false
}
```


Errors:

- **400 Bad Request:** Invalid filter parameters.
- **401 Unauthorized:** Invalid or missing token.
- **500 Internal Server Error:** Unexpected error.

Client - API

Authentication: All endpoints requiring authorization (Vault Operations) use X-API-Key and X-API-Secret headers for authentication.

1. Client Management

1.1 Register Client

Allows new API clients to register with the service and obtain their unique API key and secret.

- **URI:** /api/client/register
- **HTTP Method:** POST
- **Authorization Header:** NOT REQUIRED

Request:

JSON

```
None
{
 "_func": "register_client",
 "clientName": "MyWebAppClient5"
}
```

Response :

JSON

```
None
{"apiKey":"ext-0f2e0522-c5b9-4eb9-a6db-dadb39f6c944","apiSecret":"3e9304e6-a9c7-4457-9dec-5354aefb6cef"}
```

Errors:

- **400 Bad Request:** (e.g., if a client with the same name already exists, or invalid request body)
  JSON
- **500 Internal Server Error:** (e.g., an unexpected server issue)
  JSON

2. Vault Operations

2.1 Store ID

Stores an encrypted identification number (Aadhaar, Voter ID, ABHA ID) and returns a unique reference key.

- **URI:** /api/client/vault
- **HTTP Method:** POST
- **Authorization Headers:**
  - X-API-Key: Your API Key
  - X-API-Secret: Your API Secret

Request:

JSON

```
None
{
 "_func": "store_id",
 "idType": "AADHAAR",
 "idNumber": "123456789012"
}
```

Supported idType values: "AADHAAR", "VOTER_ID", "ABHA_ID"

Response (201 Created):

JSON

```
None
{"idType":"AADHAAR","referenceKey":"5392a2a9-b904-454e-8d4d
-c0379fb31277"}
```

Errors:

- **400 Bad Request:** (e.g., invalid idType, idNumber format validation failure)
  JSON
- **401 Unauthorized:** (e.g., missing or invalid X-API-Key or X-API-Secret)
  JSON
- **500 Internal Server Error:** (e.g., encryption failure, database error)
  JSON

2.2 Fetch ID by Reference

Retrieves a decrypted identification number using its unique reference key.

- **URI:** /api/client/vault
- **HTTP Method:** POST
- **Authorization Headers:**
  - X-API-Key: Your API Key
  - X-API-Secret: Your API Secret

Request:

```
JSON
{
 "_func": "fetch_id_by_reference",
 "reference-key": "5392a2a9-b904-454e-8d4d-c0379fb31277"
}
```

Response (200 OK):

JSON

```
None
{"idType":"AADHAAR","idNumber":"123456789012"}
```

Errors:

- **400 Bad Request:** (e.g., referenceKey is not a valid UUID format)
  JSON
- **401 Unauthorized:** (e.g., missing or invalid X-API-Key or X-API-Secret)
  JSON
- **404 Not Found:** (e.g., referenceKey does not exist)
  JSON
- **500 Internal Server Error:** (e.g., decryption failure)
  JSON

2.3 Fetch Reference by ID Value

Retrieves the unique reference key for a given identification number and its type.

- **URI:** /api/client/vault
- **HTTP Method:** POST
- **Authorization Headers:**
  - X-API-Key: Your API Key
  - X-API-Secret: Your API Secret

Request:

JSON

```
None
{
 "_func": "fetch_reference_by_id_value",
 "idType": "AADHAAR",
 "idNumber": "123456789012"
}
```

Supported idType values: "AADHAAR", "VOTER_ID", "ABHA_ID"

Response (200 OK):

JSON

```
None
{"reference-key":"5392a2a9-b904-454e-8d4d-c0379fb31277"}
```

Errors:

- **400 Bad Request:** (e.g., invalid idType, idNumber format validation failure)
  JSON
- **401 Unauthorized:** (e.g., missing or invalid X-API-Key or X-API-Secret)
  JSON
- **404 Not Found:** (e.g., no matching ID found for the given type and value)
  JSON
- **500 Internal Server Error:** (e.g., hashing failure)
  JSON

Database Design

1. api_user Table

- **Purpose:** Stores information about the API clients authorized to access the Aadhaar Vault Plus service. This table is crucial for authentication and attributing actions to specific clients in the event logs.
- **Columns:**
  - api_key (VARCHAR(255) | **Primary Key**): A unique identifier for the API client, used for authentication.
  - api_secret (VARCHAR(255) | **NOT NULL**): The secret key associated with the API key, also used for authentication.
  - client_name (VARCHAR(255) | **NOT NULL**, **UNIQUE**): A human-readable name for the client, which must be unique.
  - active (BOOLEAN | DEFAULT TRUE): Indicates whether the API client account is active (true) or deactivated (false).
  - created_datetime (TIMESTAMP WITHOUT TIME ZONE | DEFAULT CURRENT_TIMESTAMP): The timestamp when the API client was registered.

2. id_type_master Table

- **Purpose:** A master lookup table that defines and categorizes the different types of identification numbers supported by the vault (e.g., Aadhaar, Voter ID, ABHA ID). This ensures consistency and extensibility.

- **Columns:**
    - id_type_code (VARCHAR(50) | **Primary Key**): A short, unique code for the ID type (e.g., 'AADHAAR', 'VOTER_ID', 'ABHA_ID').
    - id_type_name (VARCHAR(255) | **NOT NULL**, **UNIQUE**): A human-readable name for the ID type (e.g., 'Aadhaar Number').
    - description (TEXT | NULLABLE): A longer description of the ID type.
    - validation_regex (TEXT | NULLABLE): An optional regular expression string for validating the format of IDs of this type.
    - active (BOOLEAN | DEFAULT TRUE): Indicates if this ID type is currently active/supported.

3. id_vault Table

- **Purpose:** The central and most critical table for securely storing the encrypted identification numbers and their corresponding reference keys.
- **Columns:**
    - reference_key (UUID | **Primary Key**): The unique, non-identifiable token (UUID) generated for each stored ID. This key is exposed to external systems.
    - id_type_code (VARCHAR(50) | **NOT NULL** | **Foreign Key** to id_type_master.id_type_code): Links the stored ID to its type defined in id_type_master.
    - encrypted_id_number (TEXT | **NOT NULL**): The encrypted base64representation of the actual identification number.
    - encrypted_data_key (TEXT | **NOT NULL**): Stores the Base64 encoded encrypted data key from KMS
    - hashed_id_number (VARCHAR(255) | NULLABLE | **Index**): A one-way, salted hash of the original identification number. This column facilitates efficient reverse lookups (finding reference_key from id_number) without needing to decrypt the entire table.
    - created_at (TIMESTAMP WITHOUT TIME ZONE | DEFAULT CURRENT_TIMESTAMP): The timestamp when the ID was initially stored in the vault.
- **Indexes:**
    - idx_hashed_id_number on (id_type_code, hashed_id_number): For efficient lookup when retrieving a reference_key based on the ID's type and its hashed value.

4. event_log Table

- **Purpose:** To maintain a comprehensive and immutable audit trail of all significant operations performed on the vault (e.g., storing an ID, fetching an ID).
- **Columns:**
    - log_id (BIGSERIAL | **Primary Key**): A unique, auto-incrementing identifier for each log entry.
    - api_key (VARCHAR(255) | NULLABLE | **Foreign Key** to api_user.api_key): The API key of the client that performed the operation. Nullable if an operation is not associated with a specific client (e.g., internal system event).
    - operation_type (VARCHAR(50) | **NOT NULL**): Describes the type of operation performed (e.g., 'STORE', 'FETCH').
    - id_type_code (VARCHAR(50) | NULLABLE | **Foreign Key** to id_type_master.id_type_code): The type of ID involved in the operation.
    - reference_key (UUID | NULLABLE | **Foreign Key** to id_vault.reference_key): The reference key of the ID affected by the operation. Nullable for 'STORE' operations where the reference key might be generated *after* the log entry is created, or for events not directly tied to a specific ID.
    - log_datetime (TIMESTAMP WITHOUT TIME ZONE | DEFAULT CURRENT_TIMESTAMP): The timestamp when the event occurred.

Relationships:

- api_user 1 -- M event_log: An API user can perform many logged events.
- id_type_master 1 -- M id_vault: An ID type can be associated with many vault entries.
- id_type_master 1 -- M event_log: An ID type can be involved in many logged events.
- id_vault 1 -- M event_log: A vault entry can be involved in many logged events (e.g., multiple fetches).

Security Design

Encryption:

- **Data at Rest:** All identification numbers are stored in the PostgreSQL database in an encrypted format using **AES 256 symmetric encryption**.
- **Data in Transit:** All API communication must be secured using **TLS/SSL (HTTPS)**. This is enforced by deploying the service behind a reverse proxy.

Key Management:

- All encryption/decryption keys are stored and managed in a managed Cloud KMS (e.g., AWS KMS) is utilized. Support for Azure Key Vault, Google Cloud KMS and on-premise physical HSM devices via PKCS#11 will be enabled in the future. This prevents keys from being exposed in application code, configuration files, or database.

Authentication & Authorization:

- **API Client Authentication:** Access to the vault's core operations requires robust authentication using unique X-API-Key and X-API-Secret pairs for each registered client.
- **Least Privilege:** Applications and database users are configured with the minimum necessary permissions required to perform their functions.
- **No Direct Access:** The database itself is not directly exposed to external networks; only the application service can connect to it.

Network Isolation:

- The entire vault system (application and database) is hosted on a **dedicated, isolated VLAN or private subnet** within the network infrastructure (on-premise or cloud VPC). This logically and physically separates it from other, potentially less secure, internal systems and the public internet.

Logging & Auditing:

- **Comprehensive Event Logging:** All critical operations (ID store, ID fetch, client registration) are meticulously logged in the event_log table, capturing details like api_key, operation_type, id_type, reference_key, and timestamp.
- **Centralized Logging:** Logs should be shipped to a centralized logging system (e.g., Splunk, ELK stack, cloud logging services) for aggregation, monitoring, and forensic analysis.
- **Immutable Audit Trail:** Log data is treated as immutable for compliance and post-incident investigation.

Vulnerability Management:

- Regular **security audits, penetration testing, and vulnerability scanning** of the application code, Docker images, and infrastructure components are performed.
- Using up-to-date dependencies and patching known vulnerabilities promptly.

Input Validation:

- Strict validation of all incoming API requests to prevent injection attacks, malformed data, and ensure adherence to expected formats for ID types and numbers.

Technology Choices

- Java middleware
- PostgreSQL database

Deployment Design

- The Java Jetty application is packaged into a lightweight, self-contained **Docker image** using multi-stage builds for optimal size and security.
- The PostgreSQL database can also be deployed as a Docker container, though for production, a managed database service is often preferred.