`COMPLIANCE_OVERVIEW.md`

Introduction

This document provides an overview of the compliance posture of the PII Guardian API software. It details the implementation of cryptographic components, analyzes its standing with respect to export control and foreign investment regulations, and outlines its role within the framework of GDPR. This analysis is based on the software's architecture and source code.

---

1. Export Control Classification

The software's classification is determined by its purpose and its use of cryptography.

- **Primary Function:** The application's purpose is Data Loss Prevention (DLP) and compliance. Its function is to allow tenants to scan and redact Personally Identifiable Information (PII) from their own data streams and files. It contains no features for surveillance or the interception of third-party communications.
  - *Evidence:* `README.md`, `app/services/pii_service.py`
- **Cryptography:** The software does not use encryption for data confidentiality. Cryptographic functions are employed exclusively for authentication, integrity, and secure credential storage.

Cryptographic Component Analysis

Component	Algorithm	Implementation Details
JWT Signing (Authentication)	`HS256` (HMAC using SHA-256)	A symmetric signing algorithm used to secure the management API. The key is a user-provided `SECRET_KEY` managed via environment variables. The cryptographic algorithm is hardcoded and cannot be modified by the end-user. *Evidence:* `app/core/security.py`, `app/core/config.py`
Password & API Key Storage	`bcrypt`	A one-way, salted, adaptive hashing function used for storing user passwords and API keys. This is a standard, non-reversible method for credential protection. *Evidence:* `app/core/security.py`
Webhook Secret Storage	`Fernet` (AES-128-CBC)	A symmetric authenticated encryption algorithm used to protect webhook secrets at rest in the database. The `WEBHOOK_ENCRYPTION_KEY` is managed via environment variables. *Evidence:* `app/core/security.py`

The use of cryptography is ancillary to the primary function of the application (PII scanning). The algorithms are standard, non-modifiable by the end-user, and serve authentication and integrity purposes, which supports a "mass-market" classification.

Licensing Implications (EU & Sweden)

- Based on the ancillary nature of the cryptography, the software is a candidate for self-classification under ECCN **5D992**. This classification typically does not require an export license for most destinations.
- As a contingency, if the software were classified as controlled (ECCN **5D002**), it would be eligible for authorization under the **EU General Export Authorisation No. EU008**, which is designed for mass-market dual-use items. This involves a one-time notification to the relevant national authority (e.g., Sweden's ISP).

---

2. Foreign Direct Investment (FDI) Screening (Sweden)

The software falls within the scope of Sweden's FDI screening legislation due to its capabilities in the following regulated areas:

- **Processing of Sensitive Personal Data:** The application's core function is to process and redact PII, which is an explicitly regulated activity.
- **Dual-Use Technology:** The software embeds cryptographic components for authentication and integrity, which are classified as dual-use technologies.

---

3. GDPR & Data Processing

The PII Guardian API is designed to function as a "Data Processor" on behalf of its operator (the "Data Controller") under the GDPR framework.

- **Data Processed:** The application processes any personal data contained within the text and files uploaded by its tenants for the purpose of scanning and redaction.
- **Data Storage:**
  - Tenant metadata, users, rules, and scan job results are stored in a PostgreSQL database.
  - Files uploaded for asynchronous scanning are stored in a configurable object storage backend (e.g., local disk, Amazon S3).
  - *Evidence:* `app/services/object_storage.py`, `README.md`
- **Cross-Border Data Flows:** As a self-hosted product, the physical location of data is determined by the operator's deployment choices. If an operator deploys the software on infrastructure outside the EU, they, as the Data Controller, are responsible for implementing the appropriate legal transfer mechanisms (e.g., Standard Contractual Clauses) to ensure compliance with GDPR.
- **Technical Readiness:** The software's robust, tenant-isolated architecture (`app/db/session.py`) and structured, request-aware logging (`app/middleware/logging.py`) provide the technical foundation required to support a Data Protection Impact Assessment (DPIA) or a Transfer Impact Assessment (TIA).

---

4. Third-Party & Operational Compliance

Area	Analysis & Operator Responsibility
US Re-export Regulations (EAR)	The software's dependencies include US-origin open-source libraries such as `cryptography`. As a publicly available library used for standard authentication functions, it is a candidate for **License Exception ENC** under the US Export Administration Regulations (EAR), which generally permits re-export without a specific license. The operator is responsible for confirming compliance based on the end-user and end-destination. *Evidence:* `requirements.txt`

Area	Analysis & Operator Responsibility
Open Source License Compliance	A review of key dependencies shows the use of permissive licenses (e.g., MIT, BSD, Apache 2.0). The project does not incorporate any copyleft licenses (e.g., GPL) that would restrict commercial use, modification, or the transfer of the intellectual property. *Evidence:* `requirements.txt`
End-User Screening	The software does not include a built-in mechanism for screening tenants or end-users against international sanctions lists. This is an operational process that remains the responsibility of the software's owner/operator.
Internal Compliance Programme (ICP)	The application provides the technical features necessary to support an operator's ICP. The structured logging middleware creates an auditable trail of API requests, and the authentication and authorization systems control access to data and management functions. *Evidence:* `app/middleware/logging.py`, `app/middleware/tenant_context.py`