UnitTrack -- Security & PII

UnitTrack — Security & Data Handling

How UnitTrack protects the personal and utility data you entrust to it.

Version 1.0 · 2026-07-21 · UnitTrack is a product of Polish Clover Solutions.

> Audience. This document is written for the security, privacy, compliance, and procurement teams evaluating UnitTrack, and for the analysts who rely on its outputs. It describes what is implemented today and, separately and clearly, what is designed and on the roadmap — we distinguish the two on purpose. Every control below is grounded in the running system and mapped to a recognized standard (primarily NIST SP 800-53 Rev. 5, SP 800-122, SP 800-188, SP 800-63B, and FIPS 140-3).
>
> What this document is not. It is a good-faith technical description, not a warranty, certification, or legal advice. UnitTrack has not (yet) undergone a third-party audit (e.g. SOC 2) or a FIPS 140-3 module validation; where a control depends on such external validation, we say so.

---

1. What data UnitTrack handles

UnitTrack ingests utility meter/usage extracts — the kind of CSV a utility exports from its billing or meter-data system — and turns them into disclosure-controlled analyses and maps that are safe to publish.

A customer extract may contain personal data (PII): customer names, service addresses, account identifiers, and precise geocoded coordinates, alongside non-identifying analytic fields (consumption, read dates, rate class). UnitTrack treats all uploaded customer data as potentially PII-bearing and protects it accordingly.

Data flow (as built):

1. Upload — a customer uploads a source extract through the application. The file is encrypted at rest immediately (see §2).
2. Ingest — the file is parsed into per-row records (`RawRecord`) whose contents are encrypted at rest field-by-field. The decrypted file exists only transiently, in memory / a short-lived temp file, during parsing.
3. Analyze — analyses run over the ingested records and apply statistical disclosure control (suppression, dominance, complementary suppression, optional differential privacy) so that only aggregates that meet the disclosure rules ever leave the system (see §6 and the companion Methods & Formulas document).
4. Publish — outputs (tables, choropleth maps, reports) carry the disclosure controls with them, and exports are gated so a below-threshold result cannot be downloaded.

Non-paying evaluation happens entirely on shared synthetic sample data via a public, no-login workbench — real customer data is never involved there.

---

2. Encryption at rest

All PII is encrypted at rest with AES-256-GCM (authenticated encryption). This maps to NIST SP 800-53 SC-28 (Protection of Information at Rest) and SC-28(1) (Cryptographic Protection), which explicitly permits encryption

at the level of files, records, or fields.

What is encrypted:

- Uploaded source files. The raw extract is stored as ciphertext on disk via an encrypting storage layer; the plaintext is never written to persistent storage. It is decrypted only in memory (or a short-lived, `0600`-permissioned temp file in ephemeral storage) when the server needs to parse it, and that temp is deleted on every code path.
- Ingested records (`RawRecord.payload`) — the verbatim source row, which may contain PII (names, addresses, account identifiers, and any coordinates), stored as an encrypted JSON envelope. This is where uploaded PII lives today.
- Enrichment overlays (`EnrichmentRun.derived`) — values produced during enrichment (e.g. geocoded coordinates, weather normalization) are stored encrypted.
- The separated identity/location store (roadmap — see §4). The data model defines dedicated encrypted fields for customer name, contact details, and normalized service address in a separate `CustomerIdentity` record — and for service-point geometry — but these are part of the designed tokenized separation and are not yet populated by the pipeline. These fields are already encrypted at rest — service-point coordinates are as identifying as the address they came from, so `ServiceLocation.geometry` is field-encrypted like the address beside it; only population awaits the standardize step. Today, PII is protected by the record- and file-level encryption above.

Cryptographic details (as implemented):

- AES-256-GCM, using the `cryptography` library's AEAD interface.
- A fresh 96-bit random nonce per encryption (no nonce reuse).
- Each ciphertext carries a scheme/key-version tag — `gcm1` for the shared master key, `gcm2:<key_id>` for a per-organization data key — so per-org envelope encryption and key rotation work without a schema change.
- Keys are 32 bytes (256-bit), validated at load.

Key custody (as implemented, with an honest boundary):

- The data-encryption key is supplied via a secret environment variable and is held outside the database, satisfying the intent of SC-28(3) (keys stored separately from the data they protect).
- The system fails closed: in production the application refuses to start if the encryption key (or the Django secret key) is missing, so a misconfiguration can never silently drop to an insecure mode.
- The key is never logged. Operational tooling only ever displays a truncated SHA-256 fingerprint of the key, never the key itself, and can verify a backup key decrypts existing data before rotation.
- Per-org envelope encryption (implemented): each organization's PII is encrypted under its own 256-bit data-encryption key (DEK); the DEK is stored only in wrapped form — AES-GCM encrypted under the master key-encryption key (KEK) — and never persisted in the clear. This gives per-tenant key separation and is the prerequisite for customer-managed keys (BYOK).
- Context binding (AAD, implemented): every field ciphertext is authenticated against its table + column as AES-GCM associated data (NIST SP 800-38D), so a stored blob cannot be relocated into a different field without failing its integrity check — an attacker with write access can't move an encrypted value from one column into another. (The context is recomputed at read time, not stored, so the ciphertext format is unchanged; ciphertext written before this binding still reads via a no-AAD fallback. Binding is column-scoped; per-row binding is a future step — it needs opaque primary keys, since a new row's key isn't assigned when the value is encrypted.)
- Boundary / roadmap (see §11): the KEK is still an environment secret managed by the hosting platform, and encryption uses a general-purpose cryptographic library — not yet a KMS/HSM-hosted KEK or a FIPS 140-3 CMVP-validated module. Moving the KEK into a cloud KMS (hardware-protected key store, automated rotation) is the planned hardening step and is the standard way to satisfy SC-28(3)'s hardware-key-store option for organizations that require it.

---

3. Encryption in transit

All traffic is served over TLS (HTTPS). In production the application:

- redirects HTTP → HTTPS (`SECURE_SSL_REDIRECT`);
- sends HTTP Strict Transport Security with a 1-year max-age, `includeSubDomains`, and `preload`;
- honors the platform's TLS-termination header (`SECURE_PROXY_SSL_HEADER`) so the app correctly recognizes secure requests behind the load balancer.

The session cookie is marked `HttpOnly` (not readable by JavaScript); both the session and CSRF cookies are `Secure` (HTTPS-only) with `SameSite=Lax`. (Django intentionally leaves the CSRF cookie readable so its value can accompany AJAX requests; the token is still validated server-side.) This maps to SP 800-53 SC-8 (Transmission Confidentiality and Integrity).

Sessions have both an absolute cap (12 hours) and an idle timeout (60 minutes of inactivity signs the user out), and expire at browser close (SP 800-53 AC-12, session termination).

---

4. Pseudonymization & separation of identity

Design (target architecture). UnitTrack's data model separates who from what: analytic entities (accounts, meters) are designed to carry only a pseudonymous token, while real identifiers live in a separate, access-controlled identity store (a 1:1 `CustomerIdentity` record) with all identifying fields encrypted. The token will be a keyed HMAC-SHA-256 of the source identifier — never a bare hash — with the secret held outside the database. This follows NIST SP 800-122 §4.2.3 (the re-identification mapping must live in a separate system with access controls) and the NISTIR 8053 / NYC-taxi lesson that unkeyed hashing of small identifier spaces is brute-forceable.

Honest current state. This separation is modeled and designed but not yet populated by the ingest pipeline — the tokenizing "standardize" step is a Phase-2 item. Today, ingested customer data (including any PII in the source rows) is protected by the field- and file-level encryption described in §2, not by tokenized separation. We call this out explicitly so evaluators can assess the system as it actually runs. When the standardize step ships, PII will additionally be split into the separated, tokenized identity store; the encryption described in §2 applies in both cases.

---

5. Access control

Application-mediated access only. PII is never reachable by direct database or file access; every path goes through the application, which enforces the controls below (SP 800-122 §4; SP 800-53 AC-3).

Multi-tenant isolation. All customer data is organization-scoped. A tenant user can only ever resolve datasets belonging to an organization they are a member of; cross-tenant reads and writes are blocked at the query layer. This isolation has been adversarially reviewed with no cross-tenant access path found. The one exception is platform operators (UnitTrack's own staff), who can reach tenant data for support and administration — a standard vendor-operator capability, exercised through the application; privileged configuration changes are recorded to the audit log.

Role-based access control with least privilege (AC-6) and separation of duties (AC-5). There are five roles, and access to raw PII is its own capability, held only by the PII Steward — not even an Organization Owner can read raw PII by virtue of being an admin:

Role	Manage org	Run pipeline	View results	Export	View audit	Access raw PII
Owner / Org Admin	✓	✓	✓	✓	✓	—
PII Steward	—	✓	✓	✓	—	✓
Analyst	—	✓	✓	✓	—	—
Viewer	—	—	✓	—	—	—
Auditor	—	—	—	—	✓	—

The PII Steward role cannot be self-granted — an Owner cannot make themselves a PII Steward; a second administrator must assign it (enforcing AC-5 separation of duties).

Multi-factor authentication for everyone (AAL2). MFA (TOTP second factor) is required for all authenticated access, including the Django admin control plane and any access to real (non-sample) datasets. This maps to NIST SP 800-63B AAL2. Failed logins are rate-limited per account.

Team invitations are bound to the invited email address, expire after ~7 days, are revocable by an owner, and use 192-bit unguessable tokens — a leaked invite link cannot move a different user into another tenant.

---

6. Data minimization & disclosure control

PII is masked by default. In the data-preview screens, identifying fields are masked unless a PII Steward explicitly reveals them, and every reveal is audited — who, when, and the scope (which fields, and the number of rows revealed). This is minimum-necessary disclosure (SP 800-53 AC-6, PT privacy family).

Only disclosure-controlled aggregates are published. Before any result is shown or exported, UnitTrack applies statistical disclosure control — minimum-count (k-anonymity) suppression, (n,k)-dominance suppression, complementary suppression, optional differential privacy, and a residual re-identification-risk score. Suppressed groups reveal nothing quantitative (not the value, not the contributor/row counts), whole-population models are withheld below a minimum population, and exports are hard-blocked when a published group falls below the safe floor. The methods, formulas, thresholds, and their primary-literature citations are documented in the companion Methods & Formulas document.

---

7. Audit & accountability

UnitTrack keeps an append-only audit log (`AuditEvent`) of security-relevant actions — authentication (login / logout / failed login), configuration and role changes, PII access, and exports — each recorded with the actor, timestamp, client IP address (resolved with a trusted-proxy-aware reader that reads the address the closest trusted proxy vouched for, so a forged `X-Forwarded-For` can't poison the recorded source — the same resolver guards rate-limiting), and user agent. The log is append-only, and this is enforced at three layers: application code only ever creates entries; add / change / delete are disabled in the administrative console; and a database trigger (`BEFORE UPDATE OR DELETE`) rejects any deletion and any change to a recorded event — so the log is tamper-evident even to a direct database session. (The trigger permits only the actor/organization foreign key being nulled when a user or org is deleted; the event itself and its content survive.) No audit rows are ever purged — data-retention purges target datasets and PII, not the log. This maps to SP 800-53 AU-2 (Event Logging), AU-3 (Content of Audit Records), and AU-9 (Protection of Audit Information), and supports AU-6 review. Auditors are a distinct role with read access to the audit trail and nothing else.

---

8. Application & platform security

- Fail-closed configuration. Production defaults to secure behavior: `DEBUG` is off by default, required secrets must be present or the app refuses to start, and the build runs Django's `check --deploy` to surface any misconfiguration before release. Allowed hosts are pinned (no wildcards).
- Content Security Policy & security headers. Every response carries a Content Security Policy: script sources are allow-listed to first-party and our named CDNs (blocking injected external scripts), with `frame-ancestors 'none'`, `object-src 'none'`, `base-uri 'self'`, `form-action 'self'`, plus `X-Content-Type-Options: nosniff`, `Referrer-Policy: same-origin`, and a restrictive `Permissions-Policy`. Third-party scripts (e.g. Plotly) are pinned with Subresource Integrity. The policy currently retains `'unsafe-inline'` for legacy inline handlers, so protection against inline script injection rests on Django's template auto-escaping (below) rather than CSP; removing `'unsafe-inline'` is a planned hardening step.
- Rate limiting. The public, no-login analysis endpoints are capped per client to bound automated abuse; the brute-force login throttle is scoped per-account. Limits share a durable, cross-process store so they hold across the whole service.
- Injection defenses. All database access is through the ORM (no raw SQL on user input); user-supplied values are validated against allow-lists; and CSV exports are neutralized against spreadsheet formula injection (CWE-1236), including the internal admin's view of the public waitlist.
- PII-minimizing error monitoring. Application error monitoring is configured to avoid transmitting PII — Django's `send_default_pii` is off, request bodies and stack-trace local variables are never sent, and sensitive headers (cookies, authorization, CSRF token) are stripped before an event leaves the server.
- Dependency posture. Runtime dependencies are scanned for known vulnerabilities on every change in CI (`pip-audit`) and kept current; the build upgrades `pip` to pick up its own security fixes.
- No live cross-site-scripting sink — output is auto-escaped and PII is never marked safe/raw in templates.

---

9. Data lifecycle, authority, and retention

- Authority to process (PT-2). Each dataset can record why processing is authorized and a retention note, reflecting NIST's requirement that retaining PII be an affirmatively documented activity (SP 800-53 PT-2, where "processing" includes storage, maintenance, and disposal).
- Uploaded files are retained encrypted after ingest so a customer can re-map columns and re-ingest; they are not left in plaintext at any point.
- Deletion. Deleting a dataset removes its records and purges its encrypted source file from storage; organization data is organization-scoped and deleted with the organization.
- Retention & disposal (PT-2; NIST SP 800-122 minimization). A dataset can carry a retention window (`retention_days`); once past it, a scheduled, audit-logged purge (`purge_expired`, `AuditEvent.DATA_PURGE`) either de-identifies the raw records in place — stripping direct identifiers, keeping only disclosure-controlled usage — or deletes them, with a legal-hold flag to pause disposal for a dataset under dispute. Nothing is auto-purged until a window is explicitly set (safe default). Mechanism + draft default policy: `docs/RETENTION_POLICY.md`.
- Backups & recovery (CP-9 / CP-10). The database is backed up on an automated daily schedule by the managed platform, plus an independent weekly encrypted (GPG) off-site copy whose key is stored separately from the data. The restore procedure has been tested end to end, including a canary check (`verify_encryption_key`) confirming that encrypted PII decrypts correctly after a restore — so a recovery can't silently yield unreadable data.
- Open item (see §11) — counsel sign-off. The retention mechanism above is implemented; the concrete, statute-driven values (window length; de-identify vs. delete) are drafted from a sourced research brief (`docs/RESEARCH_UTILITY_PRIVACY_RETENTION.md` — CCPA/CPRA applicability, TX/NY PUC retention rules, the §1798.140(m) de-identification standard) but await outside-counsel review before a production engagement. We are transparent that finalizing these values is an open, customer-specific legal step.

---

10. Standards & controls mapping

Standard / control	How UnitTrack addresses it
SP 800-53 SC-28 / SC-28(1) — data at rest	AES-256-GCM field- and file-level encryption of all PII (§2)
SP 800-53 SC-28(3) — key custody	Key held outside the DB, fails closed, never logged; KMS/HSM envelope = roadmap (§2, §11)
SP 800-53 SC-8 — data in transit	TLS everywhere, HSTS (1-yr, preload), secure cookies (§3)
SP 800-53 AC-3 / AC-6 / AC-5 — access, least privilege, separation of duties	Application-mediated access; PII its own capability (PII Steward only); Steward not self-grantable (§5)
SP 800-63B AAL2 — authentication	MFA required for all authenticated access (§5)
SP 800-53 AU-2 / AU-3 — audit	Append-only audit log with actor/IP/user-agent (§7)
SP 800-53 PT-2 — authority to process PII	Per-dataset processing authority + retention note (§9)
SP 800-122 §4.2.3 / §4 — de-identification & app-mediated PII	Separated tokenized identity store (designed; Phase 2); masking + gated reveal today (§4, §6)
SP 800-188 / NISTIR 8053 — de-identification & keyed pseudonymization	Keyed HMAC pseudonymization design; disclosure control on outputs (§4, §6)
FIPS 140-3 — cryptographic module	AES-256-GCM via a general-purpose library today; CMVP-validated module = roadmap (§2, §11)

---

11. Current limitations & roadmap (stated plainly)

We would rather you learn these from us than find them in a review:

1. Tokenized identity separation is Phase 2. The separated, HMAC-tokenized `CustomerIdentity` store is designed and modeled but not yet populated by the pipeline; PII is protected today by field/file encryption (§2, §4).
2. Per-org envelope encryption is implemented; KEK custody is environment-based, not KMS/HSM. Each org's PII is encrypted under its own DEK, wrapped by the master KEK and never stored in the clear. What remains is hosting that KEK in a cloud KMS or HSM (hardware-protected key store, automated rotation) — the planned hardening (§2). Key rotation hooks exist in the ciphertext format (`gcm2:<key_id>`).
3. No FIPS 140-3 module validation yet. Encryption uses a widely-used cryptographic library, not a CMVP-validated module. Deployments that require FIPS validation would use a validated module (typically via the KMS above).
4. No third-party attestation yet (e.g. SOC 2 Type II, penetration test by an external firm). Internal adversarial security reviews have been performed and their findings remediated.
5. Retention/legal specifics are customer-dependent and warrant legal review (§9).

---

12. Shared responsibility

Security is shared. UnitTrack is responsible for the controls above. The customer is responsible for: the accuracy and lawful basis of the data they upload; who in their organization they invite and which single person they designate as PII Steward; keeping their own accounts and second-factor devices secure; and determining their own regulatory retention obligations.

---

13. Questions or a deeper review

We welcome security questionnaires, architecture deep-dives, and a walkthrough of any control above against the source. Contact security@polishcloversolutions.com.

This document describes UnitTrack as of the version date above and will be updated as the roadmap items in §11 land.