**Pattern Name: Security and Privacy Protection**

**Intent.**

Specify mechanisms and components that protect user data, device integrity, and communications while preserving user privacy and control.

**Motivation.**

Mobile devices store sensitive personal and enterprise data and are constantly connected to networks and services. Threats include unauthorized access, data exfiltration, tampering, and privacy invasion by apps or third parties. A uniform pattern helps define the security and privacy-related requirements, organizes the major actors (e.g., AuthenticationManager, EncryptionModule, PermissionManager), and prescribes how they should interact so the device remains secure yet usable. The pattern supports multiple authentication methods, secure storage, policy enforcement, and audit trails while allowing user consent and transparency.

**Constraints.**

* Authentication mechanisms (biometrics, PIN/password, token) must be available and configurable; fallback options required when a method fails.
* Sensitive keys and credentials must be stored in a hardware-backed secure store (SecureEnclave / Trusted Execution Environment) when available.
* Communications must use transport security (TLS) and validate certificates; keys must be rotated and updates signed.
* App permissions require explicit user consent; permission grants must be revocable and scope-limited.
* Auditing must record security-relevant events (auth attempts, permission changes, data access) without leaking private data.
* Privacy-by-default: data collection must be minimized; personally identifiable data must be anonymized where feasible.
* Offline operation: core security functions (local authentication, secure storage) must work without network connectivity.
* Security updates and patches must be deliverable and verifiable; device must be able to rollback or disable faulty updates safely.

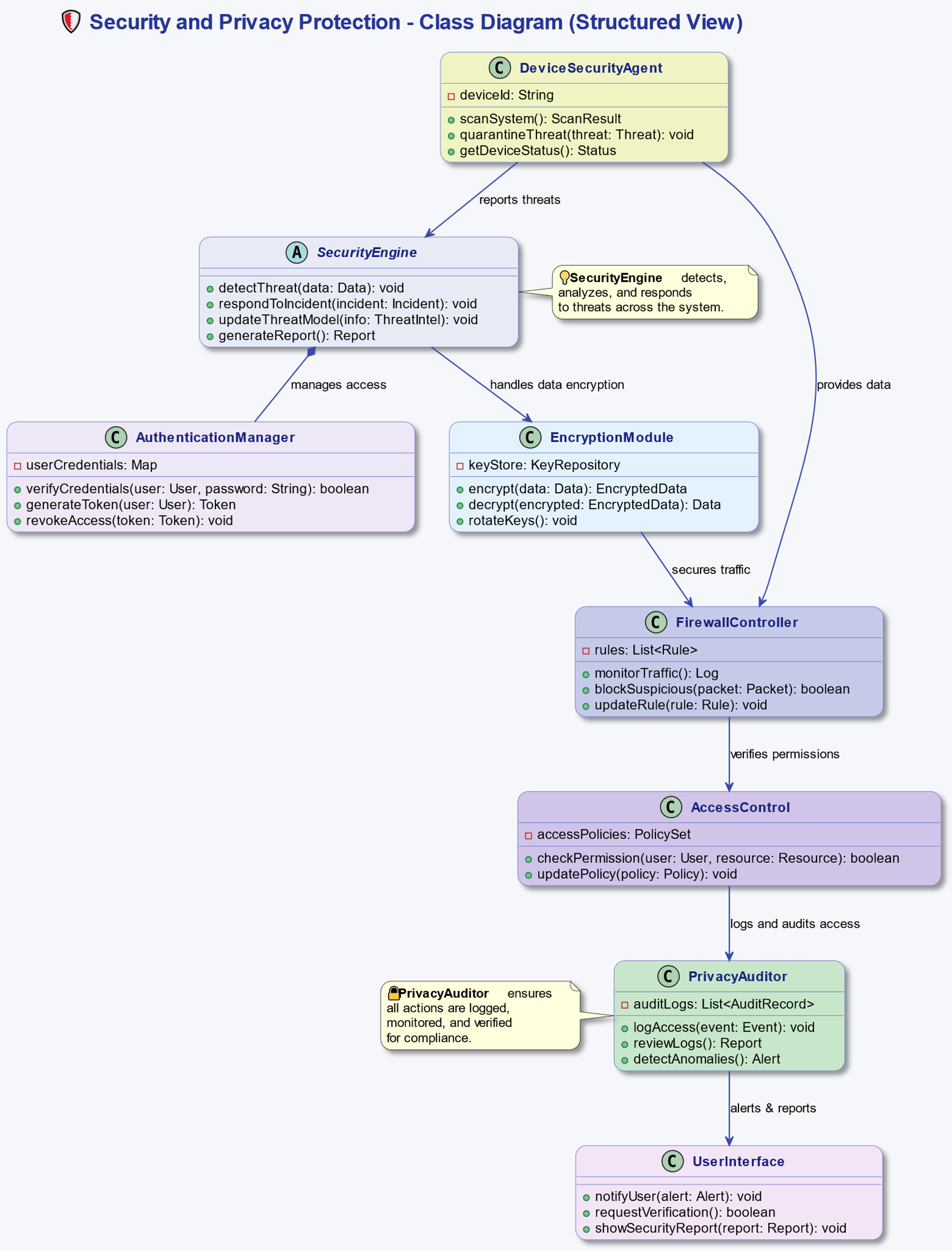
**Applicability.**

Useful in any system that processes sensitive data or interacts with external networks: mobile phones, tablets, wearables, smart home hubs, and enterprise IoT gateways.

**Structure.**

A UML class diagram represents the major components and their relationships. Key classes/components:

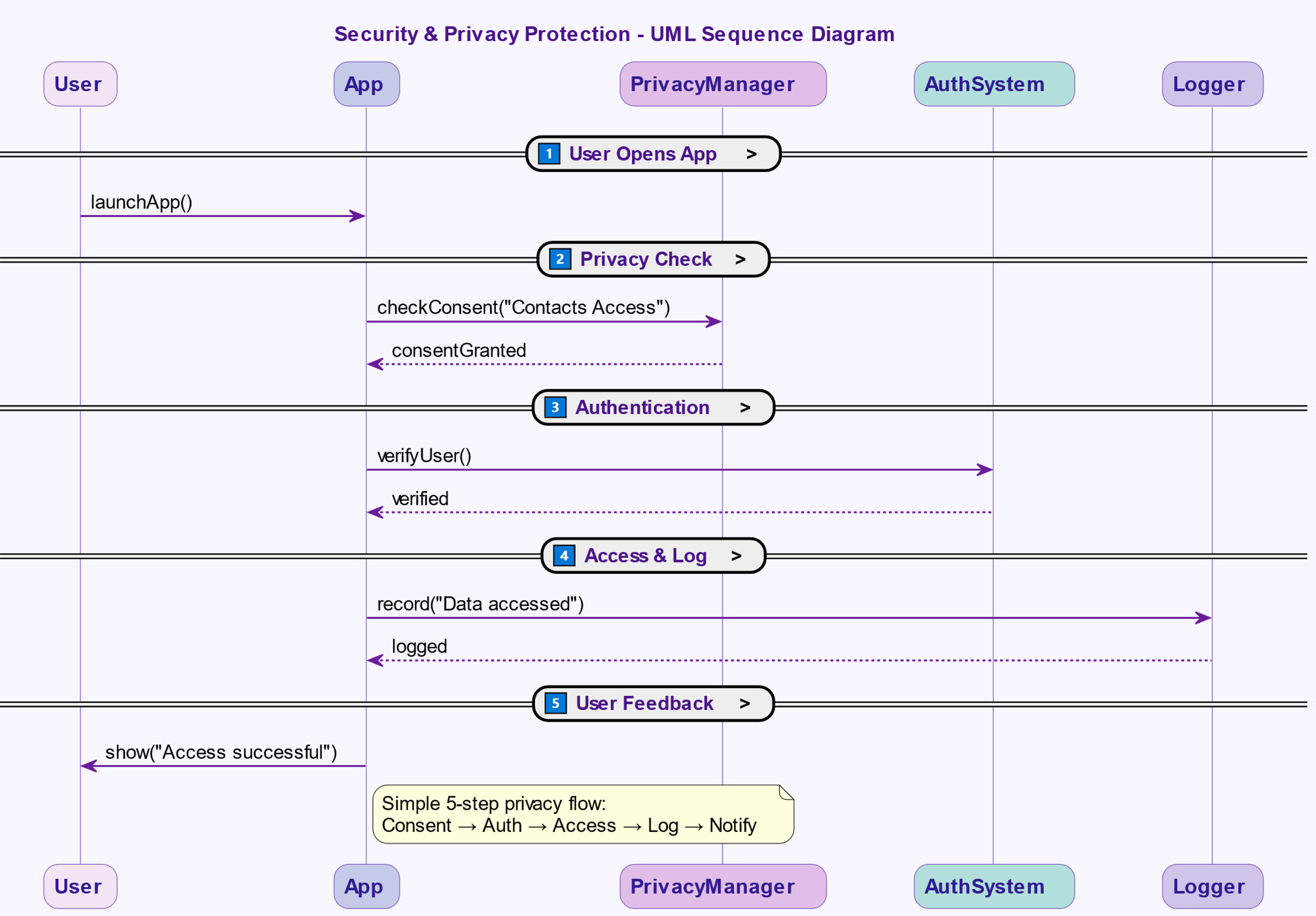
* **AuthManager (abstract):** Centralizes authentication workflows (password, biometric, token).
* **BiometricSensor:** Interface to biometric hardware (fingerprint, face).
* **CredentialStore (SecureEnclave):** Hardware-backed secure storage for keys and secrets.
* **PermissionManager:** Handles app permissions, consent, and scopes.
* **EncryptionModule:** Provides data-at-rest and data-in-transit encryption APIs.
* **PolicyEngine:** Evaluates security/privacy policies (corporate, OS, user preferences).
* **NetworkSecurity:** Manages TLS sessions, certificate validation, VPNs.
* **AuditLogger:** Records security events and writes tamper-evident logs.
* **UpdateManager:** Fetches, verifies (signature), and applies security updates.
* **PrivacyManager:** Handles anonymization, data minimization, and consent records.
* **UserInterface:** Prompts for consent, displays security notices, and handles overrides.
* **AppSandbox:** Isolates apps and mediates resource requests via PermissionManager.

****

**Behavior.**

Typical behavior flows illustrated via a sequence diagram include: (a) **User authentication and access to a sensitive resource**, and (b) **Permission request and consent flow**. Example sequence for access:

1. User requests access to a protected resource via **UserInterface**.
2. **AuthManager** initiates authentication: queries **BiometricSensor** or prompts password via UI.
3. **CredentialStore** verifies secret or signature in a secure enclave.
4. On success, **AuthManager** checks **PolicyEngine** and **PermissionManager** for allowed actions.
5. If access involves remote resources, **NetworkSecurity** establishes a TLS channel; **EncryptionModule** may encrypt data-at-rest/in-transit.
6. Result (allow/deny) is returned; **AuditLogger** records the event.
7. If the action required consent, **PrivacyManager** ensures consent is present or UI requests it; consent decisions are stored.



**Participants.**

Brief responsibilities:

* **AuthManager:** Coordinates authentication methods, manages session tokens, enforces lockout policies.
* **BiometricSensor:** Provides capture and verification for biometric checks; emits life ticks/health checks.
* **CredentialStore (SecureEnclave):** Stores private keys, encryption keys, biometric templates’ hashes; performs cryptographic operations.
* **PermissionManager:** Grants, revokes, and audits app permissions; enforces scope boundaries.
* **EncryptionModule:** Provides symmetric/asymmetric crypto operations and key management interfaces.
* **PolicyEngine:** Encodes and evaluates policies (corporate MDM policies, OS defaults, user preferences).
* **NetworkSecurity:** Handles TCP/TLS stack, certificate pinning, and secure channels.
* **AuditLogger:** Creates and stores tamper-evident logs (locally and/or to secure cloud audit store).
* **UpdateManager:** Verifies and applies signed updates; ensures integrity of security patches.
* **PrivacyManager:** Controls collection/minimization of telemetry, maintains consent records, anonymizes data.
* **UserInterface:** Solicits user credentials and consent; communicates status and warnings.
* **AppSandbox:** Runs third-party apps in isolation, mediates all sensitive resource accesses.

**Collaborations.**

* **AuthManager** uses **BiometricSensor** and **CredentialStore** to authenticate a user.
* **AuthManager** consults **PolicyEngine** for rules (e.g., require 2FA for financial apps).
* **PermissionManager** mediates app requests to sensitive resources (camera, contacts); if consent is missing, it requests UI prompt via **UserInterface**.
* **EncryptionModule** uses keys from **CredentialStore** to encrypt/decrypt data; **NetworkSecurity** uses the EncryptionModule for TLS session keys.
* **AuditLogger** receives events from AuthManager, PermissionManager, NetworkSecurity, and UpdateManager, adding timestamps and non-sensitive context.
* **PrivacyManager** inspects data flows and instructs components to anonymize or limit telemetry; it records user consents in a consent registry.
* **UpdateManager** verifies update signatures before allowing **UpdateManager** to write to system partitions; **AuditLogger** records update events.
* **AppSandbox** enforces that all inter-app resource requests go through **PermissionManager** and that data cannot be read without explicit scope.

**Consequences.**

1. Centralized and consistent security behavior — easier reasoning about system safety.
2. Increased trustworthiness: hardware-backed credential storage and signed updates reduce attack surface.
3. Clear audit trails help incident response and compliance.
4. User control and privacy preserved by consent and data-minimization mechanisms.
5. Introduces runtime and development complexity — more components to design, test, and maintain.
6. Performance overhead: encryption, policy checks, and logging may add latency; must be optimized.
7. Usability challenges: too many prompts reduce UX quality; policy design must balance security vs. convenience.