Brightli Security & Privacy Architecture

Security Model

I built Brightli's security around defense-in-depth principles in order to layer multiple controls so no single failure compromises everything. The architecture balances user protection with experience because security that creates friction gets disabled or bypassed. All design decisions align with mobile platform best practices while prioritizing data privacy through technical controls.

Data Protection at Rest

User data persists in iOS native secure storage with OS-managed encryption. The app leverages the platform's built-in protection rather than implementing custom encryption because iOS security gets more scrutiny and updates. This approach ensures data remains protected even if the device gets compromised physically, since the secure enclave handles encryption keys independently from application code.

Credential Management

Sensitive API credentials stay isolated from source code through injection during build processes. Separate configuration profiles are used for development and production environments to prevent accidental exposure. This practice addresses hardcoded credentials in source code and separating environments so that development API keys can't access production data.

Credential protection includes:

- Secrets injection preventing source code exposure
- Separate dev and production credentials limiting blast radius
- Build-time configuration preventing runtime credential access
- Credential rotation procedures for suspected compromise

Secure Data Transmission

Cloud synchronization uses iOS native encrypted transport protocols for data confidentiality during transit between device and cloud storage. Apple's CloudKit encryption is leveraged rather than implementing custom protocols because the platform handles certificate pinning, TLS version enforcement, and protocol updates automatically. This provides strong transit security while reducing the attack surface.

Subscription Integrity

Server-side transaction validation prevents unauthorized feature access through multi-layer verification. StoreKit receipts are validated on the server rather than client-side subscription status to prevent premium access through unauthorized app state modification.

Privacy-First Architecture

The architecture implements zero collection, tracking, or processing of user-generated content. Data minimization limits collection strictly to subscription management requirements with explicit user consent for optional cloud features. This is enforced through technical architecture:

- All user-generated content stays exclusively on-device
- Pattern recognition and analytics run locally
- Cloud sync uses end-to-end encryption preventing server-side access
- Zero third-party analytics or tracking SDKs
- Subscription data isolated from user content

This architecture creates competitive differentiation while building user trust through verifiable technical implementation.

Threat Mitigation Strategy

The security architecture addresses mobile-specific threat vectors including insecure data storage, insufficient cryptography, and improper platform usage. Data minimization principles reduce potential breach impact by limiting what's collected to only what's necessary for operation. The threat model covers:

- Insecure data storage mitigated through iOS secure storage and encryption
- Insufficient cryptography prevented by leveraging platform-native encryption
- Improper platform usage addressed through native API integration
- Device compromise detection through subscription validation anomalies
- Transaction replay attacks caught by server-side validation

Privacy Architecture

User privacy drives architectural decisions throughout the system rather than being added as an afterthought. The local-first design ensures sensitive user content never transmits to external servers for processing or storage. All analytics and pattern detection operations execute entirely on-device, ensuring that behavioral insights remain strictly private.