Security Architecture

Architecture Diagrams

Document Version: 1.0 **Last Updated:** 2025-10-26 **Status:** WARNING: **NOT IMPLEMENTED** - Planning Only (Phase 6+) **Priority:** CRITICAL - Required for FDA submission and HIPAA compliance

---

> ## WARNING: IMPORTANT NOTICE - ENCRYPTION NOT IMPLEMENTED
>
> **Current Status (Phase 5):** This document describes the **PLANNED** security architecture.

NO ENCRYPTION IS CURRENTLY IMPLEMENTED in this version of TOSCA.

- [FAILED] Database is **NOT encrypted** (plaintext SQLite)
- [FAILED] Video files are **NOT encrypted** (plaintext MP4)
- [FAILED] Configuration files are **NOT encrypted**
- [FAILED] Audit trail signatures **NOT implemented** (no HMAC)
- [FAILED] User authentication **NOT implemented**

Implementation Target: Phase 6 (before clinical testing)

This document serves as: - Design specification for future implementation - FDA submission documentation (planned features) - Architecture review for security requirements

DO NOT use current version for: - Clinical trials (encryption required) - Production deployment (HIPAA violation) - PHI/PII storage (unencrypted data)

Table of Contents

Overview

Purpose

This document defines the security architecture for the TOSCA Medical Laser Control System, including encryption strategy, key management, access control, and audit trail integrity.

Classification: Medical Device - FDA Class II/III (TBD)

Compliance Requirements: - **FDA 21 CFR Part 11:** Electronic records and signatures - **HIPAA:** Protected Health Information (PHI) safeguards - **IEC 62304:** Medical device software lifecycle (Enhanced Documentation Level) - **IEC 60601-1:** Medical electrical equipment safety

Security Objectives

1. **Confidentiality:** Protect patient data from unauthorized access
2. **Integrity:** Prevent tampering with treatment records and audit trails
3. **Availability:** Ensure system availability for authorized users
4. **Accountability:** Maintain complete audit trail of all actions
5. **Non-repudiation:** Prevent denial of treatment actions

Regulatory Requirements

FDA 21 CFR Part 11

Electronic Records Requirements:

1. **Validation of Systems** (§11.10(a))
   - System validation documentation
   - Security controls validation
   - Encryption implementation testing
2. **Ability to Generate Accurate and Complete Copies** (§11.10(b))
   - Encrypted data must be exportable in human-readable form
   - Complete audit trail export capability
3. **Protection of Records** (§11.10(c))
   - Records protected from unauthorized access
   - Encryption of data at rest
   - Access control mechanisms
4. **Audit Trail** (§11.10(e))
   - Secure, computer-generated, time-stamped audit trail
   - Protection from modification or deletion
   - HMAC signatures for integrity verification
5. **Device Checks** (§11.10(h))
   - Validation of authority of individuals
   - Checks to ensure only authorized individuals can use/access the system

HIPAA Security Rule

Administrative Safeguards: - Access control (§164.308(a)(4)) - Security awareness and training (§164.308(a)(5)) - Security incident procedures (§164.308(a)(6))

Physical Safeguards: - Workstation use (§164.310(b)) - Workstation security (§164.310(c)) - Device and media controls (§164.310(d))

Technical Safeguards: - Access control (§164.312(a)) - **ENCRYPTION REQUIRED** - Audit controls (§164.312(b)) - Integrity controls (§164.312(c)) - Transmission security (§164.312(e))

Protected Health Information (PHI) in TOSCA: - Patient identifiers (name, ID, DOB, gender) - Treatment parameters and outcomes - Session recordings (video files) - Operator notes and assessments

---

Encryption Strategy

Data Classification

Data Type	Classification	Encryption Required	Integrity Required
Patient identifiers	PHI	[DONE] AES-256	[DONE] HMAC
Treatment parameters	PHI	[DONE] AES-256	[DONE] HMAC
Session video recordings	PHI	[DONE] AES-256	[FAILED] (file integrity)
Audit trail events	Critical	[DONE] AES-256	[DONE] HMAC
System configuration	Sensitive	[DONE] AES-256	[DONE] HMAC
Calibration data	Sensitive	[DONE] AES-256	[DONE] HMAC
User credentials	Critical	[DONE] Argon2id	[DONE] HMAC
Temporary logs	Low	[FAILED]	[FAILED]

Database Encryption

Scope: SQLite database file containing all patient data and treatment records

Method: SQLCipher (AES-256-CBC encryption for SQLite)

Implementation:

```python
# Example: SQLCipher integration
import sqlcipher3 as sqlite3

# Open encrypted database
conn = sqlite3.connect('treatment_data.db')
conn.execute(f"PRAGMA key = '{derived_key}'")
conn.execute("PRAGMA cipher_compatibility = 4")  # Latest version
```

Benefits: - [DONE] Transparent encryption (no schema changes) - [DONE] Industry-standard AES-256 - [DONE] FIPS 140-2 compliant (when using OpenSSL FIPS module) - [DONE] Page-level encryption (entire database encrypted)

Key Derivation:

```python
# PBKDF2-HMAC-SHA256 for key derivation
import hashlib
import secrets

def derive_database_key(master_password: str, salt: bytes, iterations: int = 100000) -> bytes:
    """Derive AES-256 key from master password"""
    return hashlib.pbkdf2_hmac(
        'sha256',
        master_password.encode('utf-8'),
        salt,
        iterations,
        dklen=32  # 256 bits
    )
```

Salt Storage: Stored in separate configuration file (not in database)

Video File Encryption

Scope: Session recordings (MP4 files containing treatment video)

Method: AES-256-GCM (Galois/Counter Mode for authenticated encryption)

Implementation:

```python
# Example: Video file encryption using cryptography library
from cryptography.hazmat.primitives.ciphers.aead import AESGCM
import os

def encrypt_video_file(input_path: str, output_path: str, key: bytes) -> None:
    """Encrypt video file with AES-256-GCM"""
    aesgcm = AESGCM(key)  # 32-byte key for AES-256
    nonce = os.urandom(12)  # 96-bit nonce

    with open(input_path, 'rb') as f:
        plaintext = f.read()

    ciphertext = aesgcm.encrypt(nonce, plaintext, None)

    # Write nonce + ciphertext to output file
    with open(output_path, 'wb') as f:
        f.write(nonce)
        f.write(ciphertext)
```

File Format:

```
[12-byte nonce][encrypted video data with 16-byte authentication tag]
```

Benefits: - [DONE] Authenticated encryption (prevents tampering) - [DONE] Built-in integrity verification - [DONE] No padding oracle vulnerabilities (GCM mode)

Configuration File Encryption

Scope: System configuration, calibration data, user credentials

Method: Fernet (symmetric encryption with timestamp)

Implementation:

```python
# Example: Configuration encryption using Fernet
from cryptography.fernet import Fernet

# Generate key (one-time, store securely)
key = Fernet.generate_key()
f = Fernet(key)

# Encrypt configuration
config_data = json.dumps(config_dict).encode('utf-8')
encrypted_config = f.encrypt(config_data)

# Decrypt configuration
decrypted_config = f.decrypt(encrypted_config)
config_dict = json.loads(decrypted_config.decode('utf-8'))
```

Benefits: - [DONE] Built-in timestamp (prevents replay attacks) - [DONE] HMAC signature (integrity verification) - [DONE] AES-128-CBC under the hood - [DONE] Simple API (less prone to misuse)

Key Management

Master Key Architecture

Two-Tier Key System:

1. **Master Password:** User-provided passphrase (min 16 characters)
2. **Derived Keys:** Cryptographically derived from master password

```
1. **Master Password (user input)**
2. **PBKDF2-HMAC-SHA256    Database Encryption Key (32 bytes)**
3. **PBKDF2-HMAC-SHA256    Video Encryption Key (32 bytes)**
4. **PBKDF2-HMAC-SHA256    Config Encryption Key (32 bytes)**
```

Key Storage

Security Requirements:

1. **NEVER hardcode keys in source code**
2. **NEVER store master password in plaintext**
3. **NEVER log encryption keys**

Recommended Storage Locations:

Key Type	Storage Location	Protection Method
Master password	User memory only	N/A (never stored)
Salt values	`config/security.json`	Read-only file permissions
Derived keys	Memory only (runtime)	Zeroed on exit
Key derivation params	`config/security.json`	Read-only file permissions

Example Security Configuration File:

```json
{
  "version": "1.0",
  "database": {
    "salt": "base64_encoded_salt",
    "iterations": 100000,
    "algorithm": "PBKDF2-HMAC-SHA256"
```

```
  },
  "video": {
    "salt": "base64_encoded_salt",
    "iterations": 100000,
    "algorithm": "PBKDF2-HMAC-SHA256"
  },
  "config": {
    "salt": "base64_encoded_salt",
    "iterations": 100000,
    "algorithm": "PBKDF2-HMAC-SHA256"
  }
}
```

Key Rotation Strategy

Database Key Rotation: Not supported by SQLCipher (requires re-encryption)

Video Key Rotation: New key for each session (unique per recording)

Config Key Rotation: Manual rotation on password change

Rotation Triggers: - Suspected compromise - Employee departure - Annual security review - Password change

---

Access Control

User Authentication

Planned Implementation: Username + Password (Phase 6)

Future Enhancement: Hardware token or biometric (Phase 7+)

Password Requirements: - Minimum 16 characters - Must include uppercase, lowercase, number, special character - Cannot reuse last 5 passwords - Expires after 90 days (configurable)

Password Hashing:

```python
# Use Argon2id (winner of Password Hashing Competition)
import argon2

ph = argon2.PasswordHasher(
    time_cost=3,         # Number of iterations
    memory_cost=65536,   # 64 MB memory
    parallelism=4,       # 4 parallel threads
    hash_len=32,         # 256-bit hash
    salt_len=16          # 128-bit salt
)

# Hash password
hash = ph.hash(password)

# Verify password
try:
    ph.verify(hash, password)
    # Authentication successful
except argon2.exceptions.VerifyMismatchError:
    # Authentication failed
```

Role-Based Access Control (Future)

Planned Roles:

Role	Permissions	Encryption Key Access
Operator	Perform treatments	[DONE] (full access)

Administrator	Configure system	[DONE] (full access)
Auditor	View audit logs	[DONE] (read-only)
Technician	Calibration only	[FAILED] (no PHI access)

Audit Trail Integrity

HMAC Signatures

Purpose: Prevent tampering with audit trail records

Method: HMAC-SHA256 signatures for each audit event

Implementation:

```python
import hmac
import hashlib

def sign_audit_event(event_data: dict, secret_key: bytes) -> str:
    """Generate HMAC signature for audit event"""
    # Serialize event to canonical form
    canonical = json.dumps(event_data, sort_keys=True)

    # Generate HMAC signature
    signature = hmac.new(
        secret_key,
        canonical.encode('utf-8'),
        hashlib.sha256
    ).hexdigest()

    return signature

def verify_audit_event(event_data: dict, signature: str, secret_key: bytes) -> bool:
    """Verify HMAC signature for audit event"""
    expected_signature = sign_audit_event(event_data, secret_key)
    return hmac.compare_digest(expected_signature, signature)
```

Database Schema Enhancement

Add signature column to `treatment_events` table:

```sql
ALTER TABLE treatment_events ADD COLUMN signature TEXT;
```

Signing Process: 1. Generate canonical JSON representation of event 2. Compute HMAC-SHA256 signature using audit key 3. Store signature in `signature` column 4. On verification, recompute signature and compare

Benefits: - [DONE] Tamper-evident audit trail (required by FDA 21 CFR Part 11) - [DONE] Cryptographic proof of integrity - [DONE] Detect unauthorized modifications

---

Implementation Roadmap

Phase 1: Foundation (Before Phase 6)

Priority: CRITICAL

Tasks: 1. [DONE] Document security architecture (this file) 2. [PENDING] Select cryptography library (`cryptography` + `argon2-cffi` + `sqlcipher3`) 3. [PENDING] Implement key derivation functions 4. [PENDING] Create secure configuration file format 5. [PENDING] Add dependencies to `requirements.txt`

Dependencies:

```
# requirements.txt additions
```

```
cryptography>=41.0.0       # AES-GCM, Fernet, key derivation
argon2-cffi>=23.1.0        # Password hashing
sqlcipher3>=0.5.0          # SQLite encryption
```

Phase 2: Database Encryption (Before Clinical Testing)

Priority: CRITICAL

Tasks: 1. [PENDING] Integrate SQLCipher for database encryption 2. [PENDING] Implement key derivation from master password 3. [PENDING] Update `DatabaseManager` to use encrypted database 4. [PENDING] Add database encryption tests 5. [PENDING] Document key management procedures

Testing: - Verify encrypted database cannot be opened without key - Test key derivation with various passwords - Validate FIPS 140-2 compliance (if required)

Phase 3: Video Encryption (Before Clinical Testing)

Priority: HIGH

Tasks: 1. [PENDING] Implement AES-256-GCM video encryption 2. [PENDING] Update `RecordingManager` to encrypt videos on save 3. [PENDING] Add video decryption for playback 4. [PENDING] Test large file encryption performance 5. [PENDING] Validate authenticated encryption

Testing: - Encrypt/decrypt 1 GB video files (performance) - Verify tampering detection (authentication tag) - Test edge cases (corrupted files, wrong key)

Phase 4: Audit Trail Integrity (Before FDA Submission)

Priority: HIGH

Tasks: 1. [PENDING] Implement HMAC-SHA256 signing for audit events 2. [PENDING] Update database schema (add `signature` column) 3. [PENDING] Add signature verification on audit export 4. [PENDING] Create audit integrity validation tool 5. [PENDING] Document signature verification procedures

Testing: - Verify all audit events are signed - Test tamper detection (modify event, verify fails) - Validate signature chain integrity

Phase 5: Access Control (Phase 6+)

Priority: MEDIUM

Tasks: 1. [PENDING] Implement user authentication system 2. [PENDING] Add Argon2id password hashing 3. [PENDING] Create user management interface 4. [PENDING] Implement password policy enforcement 5. [PENDING] Add session management

Testing: - Brute-force resistance testing - Password policy validation - Session timeout testing

---

Testing & Validation

Security Testing Requirements

Unit Tests: - [DONE] Key derivation (various passwords, salts) - [DONE] Encryption/decryption round-trip (database, video, config) - [DONE] HMAC signature generation/verification - [DONE] Password hashing (Argon2id)

Integration Tests: - [DONE] Encrypted database operations (CRUD) - [DONE] Video recording with encryption - [DONE] Audit event signing during treatment - [DONE] Key management workflows

Security Tests: - [DONE] Tamper detection (modify encrypted data) - [DONE] Wrong key handling (graceful failure) - [DONE] Brute-force resistance (password hashing) - [DONE] Memory safety (key zeroing on exit)

Compliance Validation: - [DONE] FDA 21 CFR Part 11 compliance checklist - [DONE] HIPAA Security Rule compliance checklist - [DONE] IEC 62304 security requirements

Penetration Testing

Recommended Before Clinical Deployment:

1. **Static Analysis:**
    - Bandit (Python security linter)
    - Safety (dependency vulnerability scanner)
2. **Dynamic Analysis:**
    - Attempt to extract data without key
    - Modify audit trail (should fail verification)
    - SQL injection testing (prepared statements)
3. **Third-Party Audit:**
    - HIPAA security assessment
    - FDA pre-submission security review
    - Independent penetration testing

Security Best Practices

Development Guidelines

1. **Never log sensitive data** (passwords, keys, PHI)
2. **Use parameterized queries** (prevent SQL injection)
3. **Zero keys in memory** after use (prevent memory dumps)
4. **Validate all inputs** (prevent injection attacks)
5. **Use secure random** (`secrets` module, not `random`)
6. **Keep dependencies updated** (security patches)

Operational Guidelines

1. **Restrict file permissions** (database, config files)
2. **Regular security audits** (quarterly recommended)
3. **Incident response plan** (documented procedures)
4. **Security training** (all operators and administrators)
5. **Backup encryption keys** (secure off-site storage)

References

Standards & Regulations

- **FDA 21 CFR Part 11:** Electronic Records; Electronic Signatures
- **HIPAA Security Rule:** 45 CFR Part 164, Subpart C
- **IEC 62304:** Medical device software - Software life cycle processes
- **IEC 60601-1:** Medical electrical equipment - General requirements for basic safety
- **NIST SP 800-57:** Recommendation for Key Management
- **NIST SP 800-132:** Recommendation for Password-Based Key Derivation

Cryptography Resources

- **OWASP Cryptographic Storage Cheat Sheet**
- **NIST FIPS 197:** Advanced Encryption Standard (AES)
- **NIST FIPS 180-4:** Secure Hash Standard (SHA-256)
- **RFC 5869:** HMAC-based Extract-and-Expand Key Derivation Function (HKDF)
- **RFC 9106:** Argon2 Memory-Hard Function

Libraries

- **cryptography:** https://cryptography.io/
- **argon2-cffi:** https://argon2-cffi.readthedocs.io/
- **SQLCipher:** https://www.zetetic.net/sqlcipher/

Document Owner: Security Architect **Last Updated:** 2025-10-26 **Next Review:** Before Phase 6 implementation **Status:** Planning complete - Ready for implementation