I am as a shadow product manager building this with complete help from you as AI expert,i am on day 10, describe each step for day 10 activities and helpdecode how to do it, in easy and elaborative step description

Based on the document, **Day 10** focuses on **Audit Logging & Encryption Hygiene**. Let me break down each step in easy, elaborative detail to help you implement this as a shadow product manager with AI assistance.[1]

Day 10: Audit Logging & Encryption Hygiene

Objective

Log all user actions in a secure, tamper-evident manner and verify that all sensitive data (PHI) is properly encrypted at rest.[1]

Step 1: Create the Audit Logging Module (audit.py)

What you're building: A system that records every action users take—like uploading notes, generating summaries, viewing logs—in a way that cannot be secretly modified later.[1]

How it works:

- Every time someone does something in your app, you'll write a log entry to a file
- Each log entry contains: timestamp, username, what action they performed, what data they touched, and a **hash** (like a digital fingerprint) of the previous log entry
- By linking each entry to the previous one through hashes, you create a **chain**—if someone tries to delete or edit an old entry, the chain breaks and you'll know tampering occurred[1]

Create a new file: `app/audit.py`

Code template:

```
import json
import hashlib
from datetime import datetime
from pathlib import Path
```

```python
class AuditLogger:
    def __init__(self, log_file_path="logs/app_audit.jsonl"):
        self.log_file = Path(log_file_path)
        self.log_file.parent.mkdir(parents=True, exist_ok=True)
        # Create file if it doesn't exist
        if not self.log_file.exists():
            self.log_file.touch()

    def _get_last_hash(self):
        """Read the last log entry and return its hash"""
        try:
            with open(self.log_file, 'r') as f:
                lines = f.readlines()
                if lines:
                    last_entry = json.loads(lines[-1])
                    return last_entry.get('sha256_curr', '')
        except:
            pass
        return ''  # First entry has no previous hash

    def _compute_hash(self, log_entry):
        """Create a hash fingerprint of the log entry"""
        # Convert the log entry to a string and hash it
        entry_string = json.dumps(log_entry, sort_keys=True)
        return hashlib.sha256(entry_string.encode()).hexdigest()

    def log_action(self, user, action, resource, additional_info=None):
        """
        Main logging function - call this whenever a user does something

        Args:
            user: username (e.g., 'dr_smith')
            action: what they did (e.g., 'UPLOAD_NOTE', 'GENERATE_SUMMARY', 'VIEW_LOGS')
            resource: what they acted on (e.g., 'note_12345.txt', 'patient_record')
            additional_info: any extra details (dictionary)
        """
        # Get the hash of the previous log entry
        previous_hash = self._get_last_hash()

        # Create the new log entry
        log_entry = {
            "timestamp": datetime.utcnow().isoformat() + "Z",
            "user": user,
            "action": action,
            "resource": resource,
            "sha256_prev": previous_hash,
            "additional_info": additional_info or {}
        }

        # Compute hash of THIS entry
        current_hash = self._compute_hash(log_entry)
        log_entry["sha256_curr"] = current_hash

        # Append to log file (append-only = cannot change old entries)
        with open(self.log_file, 'a') as f:
            f.write(json.dumps(log_entry) + '\n')
        return log_entry
```

What each part does:

- `__init__`: Sets up where logs will be stored (`logs/app_audit.jsonl`)
- `_get_last_hash`: Reads the last log entry to get its hash (needed for chaining)
- `_compute_hash`: Creates a unique fingerprint (SHA-256 hash) for each log entry
- `log_action`: The main function you'll call from your Streamlit app whenever something happens

Step 2: Add OpenTelemetry-Style Attributes

What this means: OpenTelemetry is an industry-standard way of structuring logs so they can be exported to monitoring tools like OpenSearch later. You'll add special fields that make your logs compatible with this standard.[1]

Update the `log_action` function to include OpenTelemetry attributes:

```
import uuid

def log_action(self, user, action, resource, additional_info=None):
    """Enhanced with OpenTelemetry-style attributes"""
    previous_hash = self._get_last_hash()

    # Generate unique IDs for tracing
    trace_id = str(uuid.uuid4())
    span_id = str(uuid.uuid4())[:16]  # Shorter ID for span

    log_entry = {
        "timestamp": datetime.utcnow().isoformat() + "Z",
        "user": user,
        "action": action,
        "resource": resource,
        "sha256_prev": previous_hash,
        "additional_info": additional_info or {},

        # OpenTelemetry attributes
        "otel_trace_id": trace_id,
        "otel_span_id": span_id,
        "otel_service_name": "clinical-rag-app",
        "severity": "INFO"  # Can be DEBUG, INFO, WARN, ERROR
    }

    current_hash = self._compute_hash(log_entry)
    log_entry["sha256_curr"] = current_hash

    with open(self.log_file, 'a') as f:
        f.write(json.dumps(log_entry) + '\n')
    return log_entry
```

Why these fields matter:

- `trace_id`: Unique identifier that can link related actions together
- `span_id`: Sub-identifier for individual operations within a trace
- `service_name`: Identifies which application created the log
- `severity`: Helps filter logs by importance level

Step 3: Integrate Audit Logging into Your Streamlit App

Where to add logging: Anywhere users interact with sensitive data.[1]

Update your `app/main.py` (Streamlit app):

```
import streamlit as st
from audit import AuditLogger

# Initialize the audit logger at the top
audit_logger = AuditLogger()

# Example: Log when user uploads a note
if st.button("Upload Note"):
    # Your existing upload code...

    # Add audit log
    audit_logger.log_action(
        user=st.session_state.get('username', 'anonymous'),
        action="UPLOAD_NOTE",
        resource=uploaded_file.name,
        additional_info={"file_size": len(note_text)}
    )

# Example: Log when user generates a summary
if st.button("Generate Summary"):
    # Your existing summarization code...

    audit_logger.log_action(
        user=st.session_state.get('username', 'anonymous'),
        action="GENERATE_SUMMARY",
        resource=f"note_{note_id}",
        additional_info={"retrieval_count": len(retrieved_chunks)}
    )

# Example: Log when admin views logs
if user_role == "admin" and st.button("View Audit Logs"):
    audit_logger.log_action(
        user=st.session_state['username'],
        action="VIEW_LOGS",
        resource="app_audit.jsonl"
    )
    # Display logs
    with open("logs/app_audit.jsonl", "r") as f:
        st.text(f.read())
```

Key actions to log:

- UPLOAD_NOTE: When notes are uploaded
- GENERATE_SUMMARY: When AI generates summaries
- VIEW_LOGS: When someone accesses audit logs
- ACCESS_PHI: If anyone accesses encrypted PHI data
- DEID_PROCESS: When de-identification runs
- INDEX_UPDATE: When vector database is updated

Step 4: Verify Encryption of PHI (Fernet Encryption)

What you're checking: All PHI (Protected Health Information) must be encrypted before being saved to disk.[1]

Review your existing `deid_pipeline.py` (from Day 5) to confirm encryption is working:

```
from cryptography.fernet import Fernet
import json
from pathlib import Path

# Load or generate encryption key (DO NOT store in Git)
KEY_FILE = Path("secure_store/encryption.key")
KEY_FILE.parent.mkdir(parents=True, exist_ok=True)

if not KEY_FILE.exists():
    key = Fernet.generate_key()
    with open(KEY_FILE, 'wb') as f:
        f.write(key)
else:
    with open(KEY_FILE, 'rb') as f:
        key = f.read()

cipher = Fernet(key)

# Example: Encrypting PHI span map
def save_phi_map(note_id, phi_spans):
    """
    Encrypt and save the mapping of PHI locations

    Args:
        note_id: unique identifier for the note
        phi_spans: dictionary mapping PHI entities to their original values
    """
    phi_json = json.dumps(phi_spans)
    encrypted_data = cipher.encrypt(phi_json.encode())
    # Save encrypted data
    encrypted_file = Path(f"secure_store/{note_id}_phi.enc")
    with open(encrypted_file, 'wb') as f:
        f.write(encrypted_data)

    print(f"✓ PHI map encrypted and saved to {encrypted_file}")

# Example: Decrypting PHI (only for authorized clinicians)
def load_phi_map(note_id):
    """Decrypt and load PHI map"""
    encrypted_file = Path(f"secure_store/{note_id}_phi.enc")

    with open(encrypted_file, 'rb') as f:
        encrypted_data = f.read()

    decrypted_json = cipher.decrypt(encrypted_data).decode()
    return json.loads(decrypted_json)
```

Verification checklist:

- ✓ Encryption key stored in `secure_store/encryption.key`
- ✓ Key file added to `.gitignore` (never committed to GitHub)
- ✓ All PHI files have `.enc` extension
- ✓ Original PHI text is never saved unencrypted

Step 5: Implement Key Rotation with MultiFernet

What is key rotation? Periodically changing encryption keys to improve security. If one key is compromised, older data remains protected by previous keys.[1]

Create `app/key_rotation.py`:

```
from cryptography.fernet import Fernet, MultiFernet
import json
from pathlib import Path
from datetime import datetime

class KeyManager:
    def __init__(self, key_dir="secure_store/keys"):
        self.key_dir = Path(key_dir)
        self.key_dir.mkdir(parents=True, exist_ok=True)
        self.keys_file = self.key_dir / "key_history.json"

        # Initialize or load keys
        if not self.keys_file.exists():
            self._create_initial_key()

        self.load_keys()

    def _create_initial_key(self):
        """Generate the first encryption key"""
        key = Fernet.generate_key()
        key_record = {
            "keys": [
                {
                    "key": key.decode(),
                    "created_at": datetime.utcnow().isoformat() + "Z",
                    "active": True
                }
            ]
        }
        with open(self.keys_file, 'w') as f:
            json.dump(key_record, f, indent=2)

    def load_keys(self):
        """Load all keys (current + historical)"""
        with open(self.keys_file, 'r') as f:
            data = json.load(f)

        # Convert string keys to Fernet objects
        fernet_keys = [Fernet(k['key'].encode()) for k in data['keys']]

        # MultiFernet tries each key in order for decryption
        self.cipher = MultiFernet(fernet_keys)
        self.current_key = fernet_keys[^1_0]  # Most recent key

    def rotate_key(self):
        """Generate a new key and add it to the front"""
        new_key = Fernet.generate_key()

        with open(self.keys_file, 'r') as f:
            data = json.load(f)

        # Add new key at the beginning
        data['keys'].insert(0, {
            "key": new_key.decode(),
            "created_at": datetime.utcnow().isoformat() + "Z",
            "active": True
        })

        # Mark old keys as inactive
        for k in data['keys'][1:]:
            k['active'] = False

        with open(self.keys_file, 'w') as f:
            json.dump(data, f, indent=2)

        print(f"✓ New key generated. Total keys in rotation: {len(data['keys'])}")
        self.load_keys()  # Reload

    def encrypt(self, plaintext):
        """Encrypt using the current (newest) key"""
        return self.cipher.encrypt(plaintext.encode())

    def decrypt(self, ciphertext):
        """Decrypt using any available key (tries newest first)"""
        return self.cipher.decrypt(ciphertext).decode()

# Usage example
if __name__ == "__main__":
    km = KeyManager()

    # Encrypt some PHI
    encrypted = km.encrypt("Patient: John Doe, MRN: 12345")
    print(f"Encrypted: {encrypted[:50]}...")

    # Rotate key
    km.rotate_key()

    # Old encrypted data still decrypts with old key
    decrypted = km.decrypt(encrypted)
    print(f"Decrypted after rotation: {decrypted}")
```

How MultiFernet works:

- Stores multiple keys in a list (newest first)
- Encrypts with the **newest** key
- Decrypts by trying each key until one works (automatic backward compatibility)

Step 6: Document Key Rotation Policy

Create `docs/key_rotation_policy.md`:

```
# Encryption Key Rotation Policy

## Overview
All PHI is encrypted at rest using Fernet symmetric encryption. Keys are rotated quarterl

## Key Management
- **Storage:** Keys stored in `secure_store/keys/` (excluded from Git)
- **Algorithm:** Fernet (AES-128 in CBC mode with HMAC authentication)
- **Rotation:** MultiFernet allows multiple keys for backward compatibility

## Rotation Schedule
- **Frequency:** Every 90 days
- **Process:**
  1. Generate new key using `KeyManager.rotate_key()`
  2. New key becomes active for all new encryption
  3. Historical keys retained for decryption of old data
  4. After 2 years, consider re-encrypting old data with current key

## Manual Rotation Command
```

python app/key_rotation.py

```


## Security Controls
```

```
- Keys never committed to version control (`.gitignore`)
- Key file permissions set to 600 (owner read/write only)
- Audit log entry created on each rotation
```

Step 7: Test Everything Together

Create a test script `app/test_day10.py`:

```
from audit import AuditLogger
from key_rotation import KeyManager
import json

def test_audit_logging():
    """Test that audit logs are created correctly"""
    print("=== Testing Audit Logger ===")
    logger = AuditLogger()

    # Simulate user actions
    logger.log_action("dr_smith", "UPLOAD_NOTE", "note_001.txt", {"size": 1024})
    logger.log_action("dr_smith", "GENERATE_SUMMARY", "note_001.txt")
    logger.log_action("admin", "VIEW_LOGS", "app_audit.jsonl")

    # Verify hash chain
    with open("logs/app_audit.jsonl", "r") as f:
        lines = f.readlines()
        for i, line in enumerate(lines[-3:]):
            entry = json.loads(line)
            print(f"Entry {i+1}: {entry['action']} by {entry['user']}")
            print(f"  Previous hash: {entry['sha256_prev'][:16]}...")
            print(f"  Current hash: {entry['sha256_curr'][:16]}...")

    print("✓ Audit logging test complete\n")

def test_encryption():
    """Test encryption and key rotation"""
    print("=== Testing Encryption & Key Rotation ===")
    km = KeyManager()

    # Encrypt PHI
    original = "Patient: Jane Doe, DOB: 1980-05-15, SSN: 123-45-6789"
    encrypted = km.encrypt(original)
    print(f"Original: {original}")
    print(f"Encrypted: {encrypted[:50]}...")

    # Decrypt
    decrypted = km.decrypt(encrypted)
    assert decrypted == original, "Decryption failed!"
    print(f"Decrypted: {decrypted}")

    # Rotate key
    print("\n--- Rotating key ---")
    km.rotate_key()

    # Old data still decrypts
    decrypted_after_rotation = km.decrypt(encrypted)
    assert decrypted_after_rotation == original, "Old data can't be decrypted!"
    print(f"✓ Old encrypted data still decrypts: {decrypted_after_rotation}")

    # New encryption uses new key
    new_encrypted = km.encrypt("New patient data")
    print(f"✓ New encryption successful: {new_encrypted[:50]}...")

    print("✓ Encryption & rotation test complete\n")

if __name__ == "__main__":
    test_audit_logging()
    test_encryption()
    print("=== All Day 10 Tests Passed! ===")
```

Run the test:

```
cd app
python test_day10.py
```

Expected output:

```
=== Testing Audit Logger ===
Entry 1: UPLOAD_NOTE by dr_smith
  Previous hash: ...
  Current hash: ...
Entry 2: GENERATE_SUMMARY by dr_smith
  Previous hash: [matches previous current hash]
  Current hash: ...
Entry 3: VIEW_LOGS by admin
✓ Audit logging test complete

=== Testing Encryption & Key Rotation ===
Original: Patient: Jane Doe...
Encrypted: gAAAAABm...
Decrypted: Patient: Jane Doe...

--- Rotating key ---
✓ New key generated. Total keys in rotation: 2
✓ Old encrypted data still decrypts: Patient: Jane Doe...
✓ New encryption successful: gAAAAABm...
✓ Encryption & rotation test complete

=== All Day 10 Tests Passed! ===
```

Step 8: Update Your .gitignore

Add these lines to `.gitignore` to ensure keys never get committed:

```
# Encryption keys - NEVER commit these!
secure_store/
*.key
*_phi.enc

# Audit logs (optional - some teams commit logs, others don't)
logs/
*.jsonl
```

Deliverables for Day 10

By the end of today, you should have:

1. ✓ `app/audit.py`: Audit logger with hash-chained, append-only logs [1]
2. ✓ **OpenTelemetry attributes**: trace_id, span_id, service_name in logs [1]
3. ✓ `logs/app_audit.jsonl`: Sample audit log file with real entries [1]
4. ✓ `app/key_rotation.py`: Key manager with MultiFernet rotation [1]
5. ✓ `docs/key_rotation_policy.md`: Documentation of rotation schedule and procedures [1]
6. ✓ **Updated** `app/main.py`: Streamlit app with audit logging on all critical actions [1]
7. ✓ `.gitignore`: Ensures keys and encrypted PHI never get pushed to GitHub [1]
8. ✓ **Test script**: Validates audit logging and encryption work correctly [1]

Why This Matters for Interviews

When discussing this portfolio project with hiring managers, Day 10 demonstrates:

- **Security-first mindset**: You understand that logging and encryption aren't optional—they're core requirements for healthcare applications [1]
- **Compliance awareness**: Audit logs and encryption directly map to HIPAA Technical Safeguards (audit controls, integrity, person/entity authentication) [1]
- **Tamper evidence**: Hash-chained logs show sophisticated understanding beyond basic logging [1]
- **Operational readiness**: Key rotation policy shows you think about long-term security maintenance [1]
- **OpenTelemetry knowledge**: Using industry-standard log formats shows you're thinking about production observability [1]

This is exactly the kind of detail that separates a "built a demo" portfolio from a "thought through real-world deployment" portfolio. [1]

⁂

1. COMPLETE-WEEK-BY-WEEK-TASKS.docx