Task Manager Technical Documentation

Overview

The Task Manager is a secure command-line interface (CLI) application implementing a multi-user task management system with strong encryption and modern security practices. It features quantum-resistant password hashing, AES-256 encryption for task storage, and memory-hard key derivation.

Core Technologies

Security Components

- **Argon2 Password Hashing**: Memory-hard password hashing algorithm
- **AES-256 Encryption**: For task data protection
- **PBKDF2-HMAC-SHA256**: Key derivation function
- **Fernet**: Symmetric encryption implementation
- **Secrets Module**: Cryptographically strong random number generation

Python Libraries

- `cryptography` : Core cryptographic operations
- `argon2-cffi` : Password hashing
- `pyfiglet` : ASCII art banner generation
- `colorama` : Cross-platform colored terminal output
- `dataclasses` : Structured data management

Architecture Breakdown

1. Data Models

Task Model

```
@dataclass
class Task:
    id: int
    description: str
    status: str
    created_at: str
    user_id: str
```

Storage Format

- `users.json` : User credentials and encryption metadata
- `tasks.json` : Encrypted task data per user

2. Security Implementation

Password Hashing (Argon2)

```
ph = PasswordHasher(
    time_cost=16,        # 16 iterations
    memory_cost=2**16,   # 64MB memory usage
    parallelism=2,       # 2 parallel threads
    hash_len=32,         # 32-byte hash length
    salt_len=32          # 32-byte salt length
)
```

Key Derivation (PBKDF2)

```
kdf = PBKDF2HMAC(
    algorithm=hashes.SHA256(),
    length=32,
    salt=salt,
    iterations=100000,
)
```

Task Encryption Process

1. Generate unique salt per user
2. Derive encryption key from password using PBKDF2
3. Serialize task to JSON
4. Encrypt using Fernet (AES-256-CBC)
5. Base64 encode for storage

3. Core Components

TaskManager Class

Primary class managing all operations:

- User authentication
- Task CRUD operations
- Data persistence
- Encryption/decryption

Key Methods:

- `_encrypt_task()` : Task encryption
- `_decrypt_task()` : Task decryption
- `_get_next_task_id()` : Unique ID generation
- `_hash_password()` : Password hashing
- `_verify_password()` : Password verification

Task ID Management

```
def _get_next_task_id(self) -> int:
    if not self.tasks[self.current_user]:
        return 1

    max_id = 0
    for encrypted_task in self.tasks[self.current_user]:
        try:
            task = self._decrypt_task(encrypted_task, self.current_password)
            max_id = max(max_id, task.id)
        except Exception:
            continue
    return max_id + 1
```

4. User Interface

Session Management

- Maintains current user context
- Securely stores encryption password in memory
- Clears sensitive data on logout

Color Coding

```
def get_status_color(status: str) -> str:
    status = status.lower()
    if status == "completed":
        return Fore.GREEN
    elif status == "pending":
        return Fore.RED
    else:
        return Fore.YELLOW
```

Menu System

Two-tier menu structure:

1. Authentication Menu (Register/Login/Exit)
2. Task Management Menu (Add/View/Mark/Delete/Logout)

5. Data Flow

Task Creation

1. User inputs task description
2. System generates unique task ID
3. Creates Task object with metadata
4. Encrypts task data
5. Stores in user's task list
6. Persists to disk

Task Retrieval

1. Load encrypted task data
2. Derive encryption key from user's password
3. Decrypt task data
4. Deserialize to Task object
5. Display with appropriate formatting

Security Considerations

Encryption at Rest

- All task data encrypted before storage
- Each user's tasks encrypted with unique key
- No plaintext data stored on disk

Password Security

- Argon2 password hashing with strong parameters
- Unique salt per user
- Memory-hard algorithm resistant to hardware attacks

Key Management

- No encryption keys stored on disk
- Keys derived from user password using PBKDF2
- Separate salts for password hashing and encryption

Attack Resistance

- Rainbow Table Attacks: Prevented by unique salts
- Brute Force: Mitigated by Argon2's memory-hardness
- Quantum Attacks: Resistant through AES-256
- Memory Dumps: Minimal sensitive data in memory

Performance Considerations

Memory Usage

- Argon2 requires 64MB per hash operation
- Task decryption performed on-demand
- Efficient task ID management

Computational Overhead

- Password hashing intentionally slow (security feature)
- Task encryption relatively fast
- Bulk operations may require optimization

Limitations and Future Improvements

Current Limitations

1. Single-device usage
2. No backup/restore functionality

3. No password recovery mechanism
4. Limited task metadata

Potential Improvements

1. Implement task categories/tags
2. Add due dates and reminders
3. Multi-device synchronization
4. Backup encryption keys
5. Task sharing capabilities
6. Enhanced search functionality

Best Practices Implemented

1. Secure password storage
2. Strong encryption for data at rest
3. Memory-hard algorithms
4. Clean session management
5. Error handling for crypto operations
6. No sensitive data logging
7. Proper key derivation
8. Unique ID generation
9. Cross-platform compatibility
10. User-friendly interface