FastPass - Project Specification

Document Purpose & Maintenance Protocol: This document serves as the authoritative, high-level specification for FastPass. It provides architectural context and design decisions for developers and maintainers:

- **Current project architecture** and design principles
- **Feature specifications** and capabilities
- **CLI interface** and usage patterns
- **Security requirements** and constraints
- **Extension points** and architectural boundaries

Maintenance Requirement: This document MUST be updated whenever significant changes are made to the architecture, feature set, or CLI interface. It focuses on design decisions and system architecture rather than implementation details.

Project Mission & Purpose

FastPass is a command-line tool that provides universal file encryption and decryption capabilities across multiple file formats. It serves as a unified front-end wrapper for specialized crypto tools (msoffcrypto-tool, PyPDF2) to add or remove password protection from Microsoft Office documents and PDF files.

Core Problem Solved: Eliminates the need to learn and manage multiple separate tools for file encryption/decryption across different formats. Provides a consistent, secure interface for password protection operations while maintaining file integrity and implementing enterprise-grade security practices.

Key Differentiator: Unified CLI interface with enterprise security patterns including file isolation, in-memory validation, multiple password sources (CLI, stdin), and secure password handling. Follows proven architecture patterns with “it just works” simplicity for reliability and security.

Product Requirements Document (PRD)

Project Overview

- **Project Name:** FastPass
- **Version:** v1.0
- **Target Platform:** Primary Windows support with Python package distribution. Cross-platform compatible libraries but Windows-optimized.
- **Technology Stack:** Python, msoffcrypto-tool, PyPDF2, filetype library, pathlib
- **Timeline:** Development complete - Production ready v1.0
- **Team Size:** Single developer maintained

Target Users

- **Primary Users:** IT administrators, security professionals, business users
- **Secondary Users:** Developers, system integrators, automation script writers
- **User Experience Level:** Intermediate (comfortable with command-line tools)
- **Use Cases:** Batch file encryption, automated security workflows, document protection

Feature Specifications

Core Functionality

- ☒ Universal file encryption/decryption interface
- ☒ Microsoft Office document password protection (modern and legacy formats)
- ☒ PDF password protection and removal

- ☒ Single file processing with simplified CLI interface
- ☒ Automatic file format detection using filetype library
- ☒ Streamlined command structure for ease of use
- ☒ **Library Interface**: `DocumentProcessor` class for programmatic access
- ☒ **Convenience Functions**: `encrypt_file()`, `decrypt_file()`, `is_password_protected()`

Security & File Safety

- ☒ File format validation using filetype library
- ☒ Path traversal attack prevention with security boundaries
- ☒ Secure temporary file creation with proper permissions
- ☒ Password memory clearing and secure handling
- ☒ Error message sanitization to prevent information disclosure
- ☒ Legacy Office format protection (decrypt-only limitation documented)

Password Management

- ☒ Multiple password attempts via CLI arguments
- ☒ JSON password array input via stdin for automation
- ☒ Mixed CLI and stdin password support
- ☒ Secure password handling and memory cleanup
- ☒ Password validation before file processing

File Operations

- ☒ In-place modification with validation-based safety
- ☒ Output directory specification for processed files
- ☒ File integrity verification after operations
- ☒ Secure cleanup of temporary files

Utility Features

- ☒ Password protection status checking
- ☒ Comprehensive help with format support table
- ☒ Debug logging with automatic log file creation

☒ Simplified user interface for reliability

Success Metrics

- **Performance Targets:** File processing < 10 seconds for typical business documents
- **User Experience:** Zero data loss through validation, "it just works" simplicity, clear error messages
- **Reliability:** 99.9% successful completion rate for valid inputs
- **Security:** No password exposure in logs, secure temporary file handling

Constraints & Assumptions

- **Technical Constraints:** Requires underlying crypto libraries (msoffcrypto-tool, PyPDF2) to be available
- **Platform Constraints:** Cross-platform compatible with pure Python dependencies
- **Security Constraints:** Must maintain file confidentiality and integrity throughout operations
- **User Constraints:** Must have appropriate file permissions for input and output directories
- **Assumptions:** Users understand file encryption concepts and password management practices

Project Directory Structure

```
fastpass/
├── fastpass/                    # Main package source code
│   ├── __init__.py              # Library interface exports
│   ├── __main__.py              # Makes package executable with 'python -m
fastpass'
│   ├── app.py                   # Main application class
│   ├── cli.py                   # CLI argument parsing and main entry point
│   ├── exceptions.py            # Custom exception classes
│   ├── core/                    # Core business logic
│   │   ├── __init__.py
│   │   ├── document_processor.py # Library interface (NEW)
│   │   ├── file_handler.py      # File processing pipeline
│   │   ├── security.py          # Security validation and path checking
│   │   ├── crypto_handlers/     # Crypto tool integrations
│   │   │   ├── __init__.py
│   │   │   ├── office_handler.py # msoffcrypto-tool integration
│   │   │   └── pdf_handler.py   # PyPDF2 integration
│   │   └── password/            # Password handling modules
│   │       ├── __init__.py
│   │       └── password_manager.py # Password validation and management
│   └── utils/                   # Utility modules
│       ├── __init__.py
│       ├── logger.py            # Logging configuration
│       └── config.py            # Configuration management
│
```

```
├── tests/                     # Comprehensive test suite
│   ├── __init__.py
│   ├── conftest.py            # Test configuration and fixtures
│   ├── test_cli_basic.py      # Basic CLI functionality tests
│   ├── test_integration_basic.py # Basic integration tests
│   ├── unit/                  # Unit tests (comprehensive)
│   │   ├── test_cli_parsing.py
│   │   ├── test_library_interface.py    # NEW: Library interface tests
│   │   ├── test_office_handler_*.py     # Multiple office handler tests
│   │   ├── test_pdf_handler_*.py        # Multiple PDF handler tests
│   │   └── test_security_validation.py
│   ├── integration/           # Integration tests
│   │   └── test_library_integration.py # NEW: Library integration tests
│   ├── security/              # Security tests
│   │   └── test_attack_simulation.py
│   ├── e2e/                   # End-to-end tests
│   │   └── test_complete_workflows.py
│   └── fixtures/              # Test fixtures and sample files
│       └── sample_files/
├── dev/                       # Development documentation
│   ├── fastpass_specification.md      # This document
│   ├── fastpass_business_logic_diagram.md
│   └── manual_test_files/     # Manual testing files
├── main.py                    # Application entry point wrapper
├── pyproject.toml             # Modern Python package configuration
├── setup.py                   # Package setup (legacy compatibility)
├── requirements.txt           # Python dependencies
├── uv.lock                    # UV package manager lock file
└── README.md                  # User documentation
```

Python Dependencies

Requirements (requirements.txt):

```
msoffcrypto-tool>=5.0.0    # Office document encryption/decryption
PyPDF2>=3.0.0              # PDF processing and encryption
filetype>=1.2.0            # File type detection (replaces python-magic)
```

Package Distribution: - **CLI Application**: Available via `fastpass` command after installation - **Library Interface**: Import as `from fastpass import DocumentProcessor` - **PyInstaller Compatible**: Pure Python dependencies for executable bundling - **Modern Package Structure**: Uses `pyproject.toml` for configuration

Password Handling Architecture

Current simplified password management:

- **Multiple CLI Passwords**: Support multiple `-p` arguments for password attempts
- **stdin JSON Arrays**: Accept JSON password arrays via stdin for automation

- **Mixed Input Support**: Combine CLI passwords with stdin passwords in single command
- **Priority Order**: CLI passwords tried first, then stdin passwords
- **Secure Memory Handling**: Clear passwords from memory after use
- **No Password Files**: Removed password list file support for simplicity

Command Line Reference

Current Simplified CLI Structure

```
Usage: fastpass {encrypt|decrypt|check} [options]

Commands:
  encrypt                   Add password protection to file
  decrypt                   Remove password protection from file
  check                    Check if file requires password

Required Options:
  -i, --input FILE          Single file to process (quotes for paths with
spaces)

Password Options:
  -p, --password PASS...    Passwords to try (multiple passwords supported)
  -p stdin                  Read password array from JSON via stdin
                           Mix with CLI passwords: -p "pwd1" stdin "pwd2"

Output Options:
  -o, --output-dir DIR      Output directory (default: in-place modification)

Utility Options:
  --debug                  Enable detailed logging (auto-saves to temp
directory)
  -h, --help               Show help with format support table
  --version                Show version information

Supported File Formats:
+--------+-----+  +--------+-----+  +--------+-----+
| Format | EDC |  | Format | EDC |  | Format | EDC |
+--------+-----+  +--------+-----+  +--------+-----+
| .pdf   | EDC |  | .docm  | EDC |  | .doc   | -DC |
| .docx  | EDC |  | .xlsm  | EDC |  | .xls   | -DC |
| .xlsx  | EDC |  | .pptm  | EDC |  | .ppt   | -DC |
| .pptx  | EDC |  | .dotx  | EDC |  |        |     |
| .potx  | EDC |  | .xltx  | EDC |  |        |     |
+--------+-----+  +--------+-----+  +--------+-----+

Legend: E=Encryption, D=Decryption, C=Check
```

```
CLI Examples:
  # Encrypt file with password
  fastpass encrypt -i contract.docx -p "mypassword"
  fastpass encrypt -i "file with spaces.pdf" -p "secret"

  # Decrypt file with multiple password attempts
  fastpass decrypt -i encrypted.pdf -p "password123" "backup_pwd" "old_pwd"

  # Check if file is password protected
  fastpass check -i document.pdf
  fastpass check -i "my document.docx"

  # Specify output directory
  fastpass encrypt -i document.docx -p "secret" -o ./encrypted/

  # Passwords from stdin JSON array (automation)
  echo '["pwd1", "pwd2", "pwd3"]' | fastpass decrypt -i file.pdf -p stdin

  # Mixed CLI and stdin passwords
  echo '["pwd3"]' | fastpass decrypt -i file.pdf -p "pwd1" "pwd2" stdin

Library Examples:
  # Object-oriented interface
  from fastpass import DocumentProcessor

  with DocumentProcessor() as processor:
      result = processor.encrypt_file("document.docx", "password123")
      if result.success:
          print(f"Encrypted: {result.input_file}")

  # Convenience functions
  from fastpass import encrypt_file, decrypt_file, is_password_protected

  result = encrypt_file("contract.pdf", "secret")
  protected = is_password_protected("document.docx")
  result = decrypt_file("encrypted.pdf", ["pwd1", "pwd2"])

Exit Codes:
  0  Success
  1  General error (file access, crypto tool failure)
  2  Invalid arguments or command syntax
  3  Security violation (path traversal, invalid format)
  4  Password error (wrong password, authentication failure)
```

Key Simplifications from Previous Versions

- **Single file processing only** - No batch or recursive operations
- **Streamlined commands** - `check-password` renamed to `check`

- **Simplified password input** - Multiple `-p` arguments instead of password list files
- **Automatic features** - Debug logging auto-saves, no manual log file specification
- **Removed complexity** - No dry-run, verify, list-supported, allowed-dirs options
- **JSON array format** - Simplified stdin password format

High-Level Architecture Overview

FastPass follows a layered architecture with clear separation of concerns:

Core Processing Pipeline

CLI Interface:

```
[CLI Input] → [Security Validation] → [Crypto Handler] → [File Processing] →
[Cleanup]
```

Library Interface:

```
[DocumentProcessor] → [Security Validation] → [Crypto Handler] → [File
Processing] → [ProcessingResult]
```

1. **CLI Parsing & Validation**
    - Parse simplified command structure (encrypt/decrypt/check)
    - Validate single file input and password arguments
    - Handle stdin JSON password arrays with mixing support
2. **Library Interface (NEW)**
    - `DocumentProcessor` class for programmatic file operations
    - `ProcessingResult` dataclass with operation details
    - Convenience functions: `encrypt_file()`, `decrypt_file()`, `is_password_protected()`
    - Context manager support for automatic cleanup
3. **Security & File Validation**
    - File format detection using `filetype` library
    - Path traversal protection with security boundaries
    - File accessibility and permission validation
4. **Crypto Tool Integration**
    - **Office Handler**: Uses `msoffcrypto-tool` for .docx, .xlsx, .pptx and variants
    - **PDF Handler**: Uses `PyPDF2` for .pdf encryption/decryption
    - **Legacy Support**: Read-only decryption for .doc, .xls, .ppt formats
5. **File Processing**
    - Secure temporary file creation with proper permissions (0o600)
    - In-place modification or output directory support
    - Multiple password attempts with validation

6. **Cleanup & Error Handling**
    - Secure cleanup of temporary files
    - Comprehensive error handling with appropriate exit codes
    - Debug logging with automatic timestamp-based log files

Architecture Principles

- **Simplicity First**: Single file processing reduces complexity and attack surface
- **Security by Default**: Built-in security boundaries, no custom configuration needed
- **Wrapper Pattern**: Unified interface over specialized crypto libraries
- **Fail-Safe Design**: Comprehensive validation before any file operations
- **Clean Separation**: Each handler specializes in specific file format families

Configuration Management

- **Default Configuration**: Sensible defaults for all security and operational settings
- **Environment Variables**: `FASTPASS_*` variables for deployment customization
- **CLI Overrides**: Command-line arguments take highest precedence
- **Configuration Files**: Supports loading config.json from the user's home directory (~/.fastpass/) or the current project directory (./fastpass.json) for persistent settings, though not required for basic operation

Supported File Format Architecture

Format Family	Handler	Capabilities	Notes
Modern Office	msoffcrypto	Full EDC	.docx, .xlsx, .pptx + variants
Legacy Office	msoffcrypto	Decrypt only	.doc, .xls, .ppt (security limitation)
PDF Documents	PyPDF2	Full EDC	Complete encryption/decryption support

Error Handling Strategy

- **Exit Code 0**: Successful operation
- **Exit Code 1**: General errors (file access, crypto tools)
- **Exit Code 2**: Invalid arguments or command syntax
- **Exit Code 3**: Security violations (path traversal, format issues)
- **Exit Code 4**: Password errors (authentication failures)

Security Requirements

Core Security Principles

- **Path Traversal Protection**: Restrict file operations to safe directories (home, current working directory, system temp)
- **Input Validation**: Validate all user inputs including file paths, passwords, and command arguments

- **Secure Temporary Files**: Create temporary files with restricted permissions (0o600) in secure locations
- **Password Security**: Clear passwords from memory after use, validate password lengths, sanitize error messages
- **File Format Validation**: Use `filetype` library for magic number validation before processing
- **Error Message Sanitization**: Prevent information disclosure through error messages

Security Boundaries

- **Allowed Directories**: User home directory, current working directory, system temporary directory
- **File Size Limits**: Maximum 500MB file size to prevent resource exhaustion
- **Password Limits**: Maximum 1024 character password length
- **Path Length Limits**: Maximum 260 character path length to ensure compatibility with Windows MAX_PATH limitations

Attack Vector Mitigation

- **Path Traversal**: Whitelist approach with absolute path resolution and validation
- **Symlink Attacks**: Detect and reject symbolic links in file paths
- **Resource Exhaustion**: File size limits and memory usage monitoring
- **Information Disclosure**: Sanitized error messages and secure logging practices

Extension Points & Development Guidelines

Adding New File Format Support

1. **Handler Creation**: Implement new handler class following the existing pattern
2. **Format Registration**: Add format mapping to configuration
3. **Security Validation**: Ensure proper file format validation using `filetype` library
4. **Testing**: Comprehensive unit and integration tests for new format

CLI Extension Guidelines

- **Backward Compatibility**: Maintain existing command structure and options

- **Security First**: All new options must maintain security boundaries
- **Simplicity**: Follow the “single file, clear purpose” design principle
- **Help Integration**: Update help text and format support table

Configuration Extension

- **Environment Variables**: Use `FASTPASS_*` prefix for new configuration options
- **Default Values**: Provide sensible defaults for all new settings
- **Validation**: Validate all configuration values with appropriate error handling

Testing Strategy

Test Categories

- **Unit Tests**: Individual component testing with mocking
- **Integration Tests**: End-to-end workflow testing

- **Security Tests**: Validation of security boundaries and attack prevention
- **Format Tests**: Comprehensive testing across all supported file formats

Quality Gates

- **Code Coverage**: Minimum 85% test coverage
- **Security Validation**: All security features must have dedicated test cases

- **Format Compatibility**: Test with real-world document samples
- **Performance Testing**: Verify processing time targets for typical files

Current Implementation Status

Completed Features

- ✅ Simplified CLI interface (encrypt/decrypt/check commands)
- ✅ **Library Interface**: `DocumentProcessor` class with context manager support
- ✅ **Convenience Functions**: `encrypt_file()`, `decrypt_file()`, `is_password_protected()`
- ✅ Single file processing with security validation
- ✅ Multiple password support via CLI arguments
- ✅ stdin JSON password array support with mixing
- ✅ Automatic debug logging with timestamps
- ✅ Comprehensive help system with format support table
- ✅ Modern Office format support (full EDC)
- ✅ PDF format support (full EDC)

- ✅ Legacy Office format support (decrypt only)
- ✅ **Comprehensive Test Suite**: 360+ tests with unit, integration, security, and E2E coverage
- ✅ **Package Distribution**: Modern `pyproject.toml` configuration with CLI entry point

Architecture Decisions

- **Single File Focus**: Removed batch and recursive processing to reduce complexity
- **Dual Interface Design**: Both CLI and library interfaces sharing the same core processing pipeline
- **Simplified Password Input**: Multiple `-p` arguments instead of password list files
- **Automatic Features**: Debug logging and format detection without manual configuration
- **Security by Default**: Built-in security boundaries without custom configuration options
- **Modern Package Structure**: `pyproject.toml` with proper entry points and library exports

Development Guidelines

- **Code Organization**: Modular architecture with clear separation of concerns
- **Error Handling**: Comprehensive exception handling with specific exit codes
- **Documentation**: Code should be self-documenting with clear function and variable names
- **Testing**: Test-driven development with comprehensive coverage

Conclusion

FastPass represents a successful evolution from a complex, feature-rich tool to a simplified, secure, and reliable file encryption utility. The architectural decisions prioritize security, usability, and maintainability over feature breadth.

The current implementation demonstrates that significant simplification can improve both security posture and user experience. By focusing on single-file processing with a clean CLI interface, FastPass provides a robust foundation for secure document encryption workflows.

Future development should maintain this philosophy of simplicity while ensuring comprehensive security validation and format support coverage.