

Project: CryptoCore - Technical Requirements Document (Sprint 8)

Sprint Goal: Polish the library, ensure robustness, and create comprehensive documentation. Prepare for demonstration.

1. Project Structure & Repository Hygiene

The codebase must be refined to professional standards with comprehensive documentation and testing infrastructure.

| ID | Requirement Description | Priority |
|-------|--|----------|
| STR-1 | All requirements from previous Sprints (STR-1 to STR-4) must still be met. | Must |
| STR-2 | The repository must include a comprehensive documentation directory structure. - Required Files: API.md, USERGUIDE.md, DEVELOPMENT.md - Suggested Structure: docs/ — API.md — USERGUIDE.md — DEVELOPMENT.md — examples STR-3 A comprehensive test suite **must** be organized in a dedicated directory. Must - **Suggested Structure:** tests/ — unit/ # Unit tests for individual functions — integration/ # Integration tests for combined features — vectors/ # Known-answer test vectors — run_tests.py # Main test runner | Must |
| STR-4 | All source files must include consistent code documentation (docstrings/comments following language standards). Must | |

2. Comprehensive Test Suite

A rigorous, automated test suite must validate every component against authoritative test vectors.

| ID | Requirement Description | Priority |
|--------|---|----------|
| TEST-1 | Unit Test Coverage: Every public function in the library must have corresponding unit tests. - Coverage should exceed 90% of code paths. - Tests must be organized by module (e.g., test_aes.py, test_hmac.py, test_gcm.py). | Must |
| TEST-2 | Known-Answer Tests (KATs): The test suite must include NIST-provided test vectors for every algorithm: - AES: ECB, CBC, CFB, OFB, CTR modes (NIST SP 800-38A) - GCM: Test vectors from NIST SP 800-38D - SHA-256, SHA3-256: Test vectors from NIST FIPS 180-4 and FIPS 202 - HMAC: Test vectors from RFC 4231 | Must |

| ID | Requirement Description | Priority |
|--------|---|----------|
| | - PBKDF2: Test vectors from RFC 6070 | |
| TEST-3 | Integration Tests: The test suite must include end-to-end tests for the CLI tool: <ul style="list-style-type: none"> - Encryption/decryption round-trip for all modes - Hash verification - HMAC generation and verification - Key derivation | Must |
| TEST-4 | Negative Tests: The test suite must include tests for error conditions: <ul style="list-style-type: none"> - Invalid inputs (wrong key lengths, malformed IVs, etc.) - Authentication failures (tampered ciphertext, wrong AAD, wrong HMAC keys) - File system errors (missing files, permission issues) | Must |
| TEST-5 | Performance Tests: The test suite should include benchmarks for critical operations: <ul style="list-style-type: none"> - AES encryption/decryption throughput - Hash function performance - Key derivation with various iteration counts | Should |
| TEST-6 | Interoperability Tests: The test suite must verify compatibility with external tools: <ul style="list-style-type: none"> - OpenSSL for all symmetric modes - Standard system tools (sha256sum, openssl kdf, etc.) | Must |
| TEST-7 | Memory Safety Tests: The test suite should include tests for memory handling: <ul style="list-style-type: none"> - Large file processing (files > 1GB) - Proper cleanup of sensitive data (keys, passwords) | Should |
| TEST-8 | Test Automation: A single command must run the entire test suite. <ul style="list-style-type: none"> - Python: pytest or custom run_tests.py - C: make test or similar | Must |

Example Test Runner Structure:

```
# tests/run_tests.py
import unittest
import sys

def run_all_tests():
    # Discover and run all tests
    loader = unittest.TestLoader()
    suite = loader.discover('tests/unit', pattern='test_*.py')
    suite.addTests(loader.discover('tests/integration', pattern='test_*.py'))

    runner = unittest.TextTestRunner(verbosity=2)
    result = runner.run(suite)

    return result.wasSuccessful()
```

```
if __name__ == '__main__':
    success = run_all_tests()
    sys.exit(0 if success else 1)
```

3. API Documentation

Comprehensive API documentation must be created for library users and developers.

| ID | Requirement Description | Priority |
|-------|---|----------|
| DOC-1 | An API.md file must document every public function, class, and module in the library. | Must |
| DOC-2 | For each function/class, the documentation must include: <ul style="list-style-type: none">- Purpose and functionality description- Complete signature with parameter types and return type- Parameter descriptions (including units, formats, constraints)- Return value description- Error conditions and exceptions raised- Examples of usage- Security considerations (if applicable) | Must |
| DOC-3 | The API documentation must be organized by module: <ul style="list-style-type: none">- cryptocore.aes - Block cipher functions- cryptocore.modes - Encryption modes (ECB, CBC, CFB, OFB, CTR, GCM)- cryptocore.hash - Hash functions (SHA-256, SHA3-256)- cryptocore.mac - MAC functions (HMAC, CMAC)- cryptocore.kdf - Key derivation functions (PBKDF2, key hierarchy)- cryptocore.csprng - Cryptographically secure random number generation | Must |
| DOC-4 | The documentation must include a dependency graph or module relationship diagram. | Should |
| DOC-5 | The documentation must include version information and compatibility notes. | Must |

Example API Documentation Format:

```
# CryptoCore API Documentation

## Module: cryptocore.aes

### `encrypt_block(key, plaintext)`
Encrypts a single 16-byte block using AES-128.

**Parameters:**
- `key` (bytes): 16-byte encryption key
- `plaintext` (bytes): 16-byte block to encrypt

**Returns:**
- `bytes`: 16-byte encrypted block
```

****Raises:****
- `ValueError`: If key or plaintext length is incorrect

****Example:****
```python  
from cryptcore.aes import encrypt\_block  
  
key = b'0' \* 16  
plaintext = b'hello world!!!!!!'  
ciphertext = encrypt\_block(key, plaintext)

**Security Considerations:** - Never use ECB mode for multiple blocks. Use authenticated encryption modes like GCM instead.

## 4. User Guide

A comprehensive user guide must be created for end users of the CLI tool.

| ID   | Requirement Description                                                                                                                                                                                                                                                                                                                   | Priority |
|------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------|
| UG-1 | A USERGUIDE.md file <b>must</b> provide complete instructions for using the CLI tool.                                                                                                                                                                                                                                                     | Must     |
| UG-2 | The user guide <b>must</b> include installation instructions for various platforms.                                                                                                                                                                                                                                                       | Must     |
| UG-3 | The user guide <b>must</b> include examples for every major use case: <ul style="list-style-type: none"><li>- File encryption/decryption with all modes</li><li>- File hashing and verification</li><li>- HMAC generation and verification</li><li>- Key derivation from passwords</li><li>- Working with associated data (GCM)</li></ul> | Must     |
| UG-4 | The user guide <b>must</b> include troubleshooting section with common errors and solutions.                                                                                                                                                                                                                                              | Must     |
| UG-5 | The user guide <b>must</b> include security best practices: <ul style="list-style-type: none"><li>- Key management recommendations</li><li>- Mode selection guidance</li><li>- Password security guidelines</li></ul>                                                                                                                     | Must     |
| UG-6 | The user guide <b>must</b> include a quick reference cheat sheet.                                                                                                                                                                                                                                                                         | Must     |
| UG-7 | The user guide <b>should</b> include comparison with other tools (OpenSSL, GPG) for users familiar with those tools.                                                                                                                                                                                                                      | Should   |

### Example User Guide Section:

## File Encryption and Decryption

### Basic AES Encryption

To encrypt a file with AES-256 in CBC mode:  
```bash  
cryptcore --algorithm aes --mode cbc --encrypt \
--key 00112233445566778899aabbccddeeff00112233445566778899aabbccddeeff \
--plaintext file.txt --output file.txt.enc

```
--input secret.txt \  
--output secret.enc
```

Working with GCM and Associated Data

GCM provides both confidentiality and integrity protection. You can also include additional authenticated data (AAD):

```
# Encrypt with AAD  
cryptocore --algorithm aes --mode gcm --encrypt \  
  --key 00112233445566778899aabbccddeeff \  
  --input database.sql \  
  --output database.enc \  
  --aad "database_version_3.2"  
  
# Decrypt with the same AAD (verification happens automatically)  
cryptocore --algorithm aes --mode gcm --decrypt \  
  --key 00112233445566778899aabbccddeeff \  
  --input database.enc \  
  --output database_decrypted.sql \  
  --aad "database_version_3.2"
```

Important: If the AAD doesn’t match or the ciphertext was tampered with, the tool will fail without outputting any data.

5. Quality Assurance

Final quality checks to ensure production-ready code.

| ID | Requirement Description | Priority |
|------|--|----------|
| QA-1 | All code must be reviewed for security vulnerabilities: <ul style="list-style-type: none">- No hardcoded keys or passwords- Proper memory management (for C)- Constant-time operations where required- Secure random number generation | Must |
| QA-2 | The code must be checked for common issues: <ul style="list-style-type: none">- Buffer overflows (C)- Integer overflows- Memory leaks | Must |
| QA-3 | All external dependencies must be documented and pinned to specific versions. | Must |
| QA-4 | The project must include a CHANGELOG.md documenting changes from previous sprints. | Must |
| QA-5 | The project must include a CONTRIBUTING.md with guidelines for future contributors. | Should |
| QA-6 | The project should include a SECURITY.md file with security disclosure guidelines. | Should |

Example Security Checklist:

Security Checklist

Critical Security Requirements

- [] No secret keys logged or printed (except when explicitly requested)
- [] All random values generated using cryptographically secure RNG
- [] Sensitive memory cleared after use
- [] Authentication performed before decryption (GCM, HMAC)
- [] No use of deprecated or insecure algorithms (ECB mode only for testing)
- [] Input validation on all user-provided data
- [] Proper error handling without leaking sensitive information