Project: CryptoCore - Technical Requirements Document (Sprint 6)

Sprint Goal: Combine encryption and authentication into a single, secure operation.

1. Project Structure & Repository Hygiene

The codebase must be extended to support authenticated encryption modes while maintaining existing structure.

ID	Requirement Description	Priority
STR-1	All requirements from previous Sprints (STR-1 to STR-4) **must** still be met.	Must
STR-2	New source files for authenticated encryption **must** be created. - **Suggested Path:** `src/modes/gcm.py/gcm.c` for GCM implementation - **Suggested Path:** `src/aead/` for general AEAD interfaces and Encrypt-then-MAC	Must
STR-3	The `README.md` file **must** be updated to include: - Documentation for the new GCM mode and AAD functionality. - Explanation of AEAD concepts and security properties. - Examples of encryption and decryption with associated data. - Warnings about the catastrophic failure behavior with wrong AAD/tampered data.	Must
STR-4	The build system **must** be updated to include any new source files for AEAD implementations.	Must

2. Command-Line Interface (CLI) Parser

The encryption/decryption CLI must be extended to support GCM mode and associated data.

ID	Requirement Description	Priority
CLI-1	The `--mode` argument **must** be extended to accept `gcm`.	Must
CLI-2	A new `--aad DATA` argument **must** be added for authenticated encryption modes. - The AAD **must** be provided as a hexadecimal string. - The AAD **should** be optional; if not provided, it should be treated as empty.	Must
CLI-3	For GCM mode, the tool **must** generate a random 12-byte nonce (instead of a 16-byte IV) during encryption. - The nonce **must** be prepended to the ciphertext output, similar to IV handling in previous modes.	Must
CLI-4	For GCM decryption, the nonce **must** be read from the input file (first 12 bytes) or provided via `--iv` (which should be renamed to `--nonce` for consistency, but maintaining `--iv` for backward compatibility is acceptable).	Must
CLI-5	The output during decryption **must** be suppressed if authentication fails. The tool **must** exit with an error without writing any decrypted data to the output file.	Must

Example Invocations:

```
# GCM Encryption with AAD
$ cryptocore --algorithm aes --mode gcm --encrypt --key 00112233445566778899aabbccddeeff --input plaintext.txt --output ciphertext.bin --aad aabbccddeeff

# GCM Decryption with correct AAD
$ cryptocore --algorithm aes --mode gcm --decrypt --key 00112233445566778899aabbccddeeff --input ciphertext.bin --output decrypted.txt --aad aabbccddeeff
> [SUCCESS] Decryption completed successfully

# GCM Decryption with wrong AAD
$ cryptocore --algorithm aes --mode gcm --decrypt --key 00112233445566778899aabbccddeeff --input ciphertext.bin --output decrypted.txt --aad wrongaad123
> [ERROR] Authentication failed: AAD mismatch or ciphertext tampered
> Exit code: 1
```

3. Authenticated Encryption Implementation

This sprint involves implementing both a general Encrypt-then-MAC paradigm and the specific GCM mode.

ID	Requirement Description	Priority
AEAD-1	**Encrypt-then-MAC paradigm must** be implemented as a composite function. - This **should** combine any block cipher mode (e.g., CTR) with HMAC from Sprint 5. - The implementation **must** follow: `C = E(K_e, P)`, `T = MAC(K_m, C ‖ AAD)`, output `C ‖ T` - Different keys **should** be used for encryption and MAC (derive from master key using KDF or use key separation).	Must
AEAD-2	**GCM mode must** be implemented from scratch. - The implementation **must** follow NIST SP 800-38D specification. - It **must** use the AES primitive from earlier sprints in CTR mode for encryption.	Must
AEAD-3	The GCM implementation **must** include Galois Field multiplication in GF(2^{128}). - The field representation **must** use the irreducible polynomial $x^{128} + x^7 + x^2 + x + 1$. - The implementation **must** be efficient (using precomputed tables or the Karatsuba algorithm is recommended).	Must
AEAD-4	GCM **must** correctly handle: - Nonce (12 bytes recommended, but support for other lengths with GHASH) - Associated Data (AAD) of arbitrary length - Plaintext of arbitrary length	Must
AEAD-5	The authentication tag **must** be 16 bytes (128 bits) and appended to the ciphertext. - Output format: `Nonce (12 bytes) ‖ Ciphertext ‖ Tag (16 bytes)`	Must
AEAD-6	During decryption, the implementation **must** verify the tag before outputting any plaintext. - If verification fails, **no plaintext** should be output and an error should be raised immediately.	Must

Expected GCM Implementation Structure:

```
# Python: src/modes/gcm.py
class GCM:
    def __init__(self, key, nonce=None):
        self.aes = AES(key)  # AES implementation from earlier sprints
        self.nonce = nonce or os.urandom(12)
        # Precompute multiplication table for GHASH
        self._precompute_table()

    def _ghash(self, aad, ciphertext):
        """Compute GHASH in GF(2^128)"""
        # Implementation of Galois Field multiplication
        pass

    def _mult_gf(self, x, y):
        """Multiply two elements in GF(2^128) modulo x^128 + x^7 + x^2 + x + 1"""
        pass

    def encrypt(self, plaintext, aad=b""):
        # Generate J0 from nonce
        # Encrypt using CTR mode with J0+1, J0+2, ...
        # Compute authentication tag
```

```
        # Return nonce + ciphertext + tag
        pass

    def decrypt(self, data, aad=b""):
        # Extract nonce, ciphertext, tag
        # Verify authentication tag
        # If verification fails, raise exception immediately
        # If verification succeeds, decrypt using CTR mode
        pass
```

Galois Field Multiplication Example:

```
def _mult_gf(self, x, y):
    """GF(2^128) multiplication using the irreducible polynomial"""
    z = 0
    v = y
    for i in range(127, -1, -1):
        if (x >> i) & 1:
            z ^= v
        if v & 1:
            v = (v >> 1) ^ 0xE1000000000000000000000000000000
        else:
            v >>= 1
    return z
```

4. File I/O for AEAD

The file handling must be extended to support the new AEAD formats and security requirements.

ID	Requirement Description	Priority
IO-1	For GCM encryption, the output file **must** contain: `12-byte nonce ‖ ciphertext ‖ 16-byte tag`.	Must
IO-2	For GCM decryption, the input file **must** be parsed as: `12-byte nonce ‖ ciphertext ‖ 16-byte tag`.	Must
IO-3	If authentication fails during decryption, the tool **must not** create or write to the output file. - The tool **must** delete any partially created output file if authentication fails.	Must
IO-4	The AAD **must** be passed to the encryption/decryption functions as a binary string.	Must

5. Testing & Verification

Comprehensive testing must verify both correctness and the critical security property of authentication failure.

ID	Requirement Description	Priority
TEST-1	**Known-Answer Tests:** The GCM implementation **must** pass test vectors from NIST SP 800-38D. - Specifically, test vectors with different AAD lengths and message lengths.	Must
TEST-2	**Round-trip Test:** Encrypting and then decrypting with the same AAD **must** return the original plaintext.	Must
TEST-3	**AAD Tamper Test:** Decryption with incorrect AAD **must** fail catastrophically. - The tool **must not** output any decrypted data. - The tool **must** return a non-zero exit code. - The tool **must** print a clear authentication error message.	Must
TEST-4	**Ciphertext Tamper Test:** Changing any bit in the ciphertext or tag **must** cause authentication failure. - The failure **must** be catastrophic (no plaintext output).	Must
TEST-5	**Nonce Reuse Test:** The implementation **must** generate a random nonce for each encryption. - Generating 1000 nonces should produce 1000 unique values.	Must
TEST-6	**Empty AAD Test:** The implementation **must** work correctly with empty AAD.	Must
TEST-7	**Large AAD Test:** The implementation **must** handle AAD larger than memory by processing in chunks.	Must
TEST-8	**Interoperability Test:** The GCM implementation **should** be tested against OpenSSL: `openssl enc -aes-128-gcm -K <key> -iv <nonce> -aad <aad> -in <file>`	Should
TEST-9	**Encrypt-then-MAC Test:** The composite Encrypt-then-MAC implementation **must** pass similar authentication tests.	Must

Example Test Commands:

```
# Test with NIST GCM test vector
$ echo -n "Hello GCM world" > test_input.txt
$ cryptocore --algorithm aes --mode gcm --encrypt --key 00000000000000000000000000000000 --iv 000000000000000000000000 --aad "" --input test_input.txt --output test_output.bin

# Tamper detection: modify ciphertext
$ dd if=test_output.bin of=tampered.bin bs=1 count=100 conv=notrunc
$ echo "XX" | dd of=tampered.bin bs=1 seek=50 count=2 conv=notrunc
$ cryptocore --algorithm aes --mode gcm --decrypt --key 00000000000000000000000000000000 --input tampered.bin --output should_fail.txt --aad ""
> [ERROR] Authentication failed: ciphertext tampered
> Exit code: 1

# Verify no output file was created
$ ls should_fail.txt
> ls: cannot access 'should_fail.txt': No such file or directory
```

Example Test Script for GCM Security Properties:

```
# tests/test_gcm_security.py
import os
from src.modes.gcm import GCM

def test_gcm_aad_tamper():
    """Test that wrong AAD causes catastrophic failure"""
    key = os.urandom(16)
    plaintext = b"Secret message"
    aad_correct = b"correct_aad"
    aad_wrong = b"wrong_aad"

    # Encrypt with correct AAD
    gcm = GCM(key)
    ciphertext = gcm.encrypt(plaintext, aad_correct)

    # Try to decrypt with wrong AAD
    gcm2 = GCM(key, gcm.nonce)
    try:
        result = gcm2.decrypt(ciphertext, aad_wrong)
        assert False, "Decryption should have failed with wrong AAD"
    except AuthenticationError:
        print("✓ Correctly failed with wrong AAD")
        # Verify no plaintext was returned
        assert 'result' not in locals() or result is None

def test_gcm_ciphertext_tamper():
```

```
"""Test that ciphertext tampering causes catastrophic failure"""
key = os.urandom(16)
plaintext = b"Another secret message"
aad = b"associated_data"

# Encrypt
gcm = GCM(key)
ciphertext = gcm.encrypt(plaintext, aad)

# Tamper with ciphertext (flip one bit)
tampered = bytearray(ciphertext)
tampered[20] ^= 0x01  # Flip one bit in the ciphertext

# Try to decrypt tampered ciphertext
gcm2 = GCM(key, gcm.nonce)
try:
    result = gcm2.decrypt(bytes(tampered), aad)
    assert False, "Decryption should have failed with tampered ciphertext"
except AuthenticationError:
    print("✓ Correctly failed with tampered ciphertext")
```