Báo Cáo Đồ Án MyFS - Hệ Thống Lưu Trữ Tập Tin An Toàn

GVHD: Thái Hùng Văn

Thành viên nhóm:

Họ và Tên	MSSV
Nguyễn Thị Quỳnh Anh	22520064
Lưu Trung Kiên	22520706

1. Tổng Quan

MyFS là một hệ thống lưu trữ tập tin an toàn được thiết kế để lưu trữ dữ liệu trong một file MyFS.DRI (tương tự như file .ISO/.ZIP/.RAR) trên cloud disk, với metadata được mã hóa và lưu trữ trong file **MyFS.METADATA** trên removable disk. Hệ thống đảm bảo rằng cả hai disk phải được kết nối để có thể truy cập dữ liệu.

2. Tiêu Chí Thiết Kế

2.1. Bảo Mật Dữ Liệu (Ưu Tiên Cao Nhất)

- Mã hóa **AES-256-CBC** cho toàn bộ volume
- Mã hóa riêng cho từng tập tin với mật khẩu độc lập
- Xác thực máy tính thông qua Machine ID (WMI hoặc MAC address)
- Mật khẩu động (dynamic challenge)
- Không lưu trữ mật khẩu dưới dạng plaintext

2.2. Toàn Vẹn Dữ Liệu

- Kiểm tra hash cho từng tập tin
- Kiểm tra toàn vẹn hệ thống
- Tự động sao lưu và khôi phục
- Xác thực metadata

2.3. Phục Hồi Dữ Liệu

- Khôi phục tập tin đã xóa
- Tự động sửa chữa khi phát hiện thay đổi
- Sao lưu tự động với mật khẩu riêng

2.4. Giới Hạn Kỹ Thuật

- Tối đa 100 tập tin trong volume
- Không hỗ trợ hệ thống thư mục
- Kích thước tập tin tối đa: 4GB
- Tập tin > 100MB không yêu cầu bảo mật cao

3. Kiến Trúc Hệ Thống

3.1. Các Module Chính

1. **myfs.py**: Module chính, xử lý CLI và điều phối
2. **myfs_formatter.py**: Tạo và định dạng volume
3. **myfs_security.py**: Xử lý bảo mật và xác thực
4. **myfs_self_repair.py**: Tự động sửa chữa và sao lưu
5. **myfs_file_handler.py**: Xử lý tập tin
6. **myfs_metadata.py**: Quản lý metadata
7. **myfs_connector.py**: Kết nối các module
8. **myfs_hardware.py**: Xác định thông tin máy tính

3.2. Cấu Trúc Dữ Liệu

- **MyFS.DRI**: File volume chính
  - Header với thông tin máy tính
  - Bảng quản lý tập tin
  - Vùng dữ liệu được mã hóa
- **MyFS.METADATA**: File metadata
  - Thông tin xác thực
  - Metadata của các tập tin
  - Thông tin sao lưu

4. Các Chức Năng Đã Triển Khai

4.1. Quản Lý Volume

- Tạo và định dạng volume mới
- Thiết lập, kiểm tra và thay đổi mật khẩu volume
- Kiểm tra tính toàn vẹn volume

4.2. Quản Lý Tập Tin

- Liệt kê tập tin (bao gồm tập tin đã xóa)
- Import tập tin với metadata
- Export tập tin với tùy chọn đường dẫn gốc
- Xóa tập tin (có thể khôi phục)
- Xóa vĩnh viễn
- Khôi phục tập tin đã xóa (nếu xóa ở chế độ bình thường)

4.3. Bảo Mật

- Thiết lập mật khẩu và mã hóa AES-256-CBC cho volume
- Thiết lập và mã hóa riêng cho từng tập tin
- Xác thực máy tính
- Mật khẩu động
- Kiểm tra toàn vẹn hệ thống

4.4. Tự Động Sửa Chữa

- Phát hiện thay đổi hệ thống
- Tự động sao lưu
- Tự động khôi phục từ sao lưu
- Cập nhật sao lưu với phiên bản hiện tại

5. Cách Sử Dụng

5.1. Các Lệnh Cơ Bản

- Hiển thị trợ giúp `python myfs.py -h`
- Tạo volume mới `python myfs.py create`
- Liệt kê tập tin `python [myfs.py](http://myfs.py) list` hoặc `python myfs.py list -a`
- Import tập tin `python myfs.py import <đường_dẫn> [-e]`
- Export tập tin `python myfs.py export <index> <đường_dẫn> [-o]`
- Xóa tập tin `python myfs.py delete <index> [-p]`
- Khôi phục tập tin `python myfs.py recover <index>`
- Đặt mật khẩu tập tin `python myfs.py setpass <index>`
- Kiểm tra toàn vẹn `python myfs.py verify <index>`
- Kiểm tra toàn vẹn hệ thống `python myfs.py integrity`
- Tạo/cập nhật sao lưu `python myfs.py backup`
- Chế độ tương tác `python myfs.py interactive`

5.2. Chế Độ Tương Tác

- `list`: Liệt kê tập tin
- `list -d`: Liệt kê cả tập tin đã xóa
- `import <đường_dẫn>`: Import tập tin
- `import -e <đường_dẫn>`: Import và mã hóa
- `export <index> <đường_dẫn>`: Export tập tin
- `export -o <index> <đường_dẫn>`: Export về đường dẫn gốc
- `delete <index>`: Xóa tập tin
- `delete -p <index>`: Xóa vĩnh viễn
- `recover <index>`: Khôi phục tập tin
- `setpass <index>`: Đặt mật khẩu tập tin
- `verify <index>`: Kiểm tra toàn vẹn
- `passwd`: Đổi mật khẩu volume
- `integrity`: Kiểm tra toàn vẹn hệ thống
- `backup`: Tạo/cập nhật sao lưu
- `exit`: Thoát

6. Yêu Cầu Hệ Thống

- Python
- **pycryptodome** library
- Hệ điều hành: Windows/Linux/MacOS

7. Thuật Toán và Triển Khai

7.1. Mã Hóa và Bảo Mật

7.1.1. Mã Hóa AES

Hàm `encrypt_aes_cbc` và `decrypt_aes_cbc` trong module `myfs_utils.py` thực hiện việc mã hóa và giải mã dữ liệu sử dụng thuật toán AES-256-CBC với padding PKCS#7. Các hàm này đảm bảo tính bảo mật cao cho dữ liệu được lưu trữ trong volume.

```
# myfs_utils.py
def encrypt_aes_cbc(plaintext_bytes, key_bytes, iv_bytes):
  """Encrypts plaintext using AES-256-CBC with PKCS#7 padding."""
  cipher = AES.new(key_bytes, AES.MODE_CBC, iv_bytes)
  padded_plaintext = pad(plaintext_bytes, AES.block_size, style="pkcs7")
  ciphertext = cipher.encrypt(padded_plaintext)
  return ciphertext

def decrypt_aes_cbc(ciphertext_bytes, key_bytes, iv_bytes):
  """Decrypts using AES-256-CBC with PKCS#7 padding."""
  cipher = AES.new(key_bytes, AES.MODE_CBC, iv_bytes)
  padded_plaintext = cipher.decrypt(ciphertext_bytes)
  plaintext = unpad(padded_plaintext, AES.block_size, style="pkcs7")
  return plaintext
```

7.1.2. Xác Thực Mật Khẩu Động

Hàm `generate_dynamic_password` trong module `myfs_security.py` tạo ra một thử thách động dựa trên thời gian hiện tại và thời gian tạo volume. Thử thách này yêu cầu người dùng thực hiện một phép tính đơn giản với các số ngẫu nhiên, tăng cường bảo mật cho quá trình xác thực.

```
# myfs_security.py
def generate_dynamic_password(self):
    """Generates a dynamic password challenge based on timestamp and volume
data."""
    if not self.header_data:
      self._read_volume_header()

    # Seed with current time and volume creation time
    current_time = int(time.time())
    creation_time = self.header_data["creation_timestamp"]

    # Use XOR of times to seed random
    seed = current_time ^ creation_time
    random.seed(seed)

    # Generate a challenge (e.g., simple math operation with random numbers)
    a = random.randint(1, 100)
    b = random.randint(1, 100)
```

```
    op = random.choice(['+', '-', '*'])

    # Create challenge string
    challenge = f"What is {a} {op} {b}?"

    # Calculate expected answer
    if op == '+':
      answer = a + b
    elif op == '-':
      answer = a - b
    else:  # '*'
      answer = a * b

    return challenge, str(answer)
```

7.1.3. Xác thực máy tính

Hàm `verify_machine_id` trong module `myfs_security.py` thực hiện việc xác thực máy tính thông qua việc tạo và kiểm tra hash của thông tin phần cứng. Điều này đảm bảo volume chỉ có thể được truy cập từ máy tính đã tạo ra nó.

```
# myfs_security.py
def verify_machine_id(self):
    """Verifies if the current machine is the one that created the volume."""
    if not self.header_data:
      self._read_volume_header()

    # Get current machine ID hash
    current_machine_id_hash, _ = get_machine_id_hash()

    # Compare with stored machine ID hash
    return current_machine_id_hash == self.header_data["machine_id_hash"]
```

7.2. Quản Lý Volume

7.2.1. Thiết lập và format volume (với mật khẩu volume)

Hàm `format_new_volume` trong module `myfs_formatter.py` thực hiện việc tạo và định dạng một volume MyFS mới, bao gồm việc tạo file MyFS.DRI và MyFS.METADATA. Quá trình này bao gồm việc xác thực đầu vào, thu thập thông tin máy tính, tạo các thành phần mã hóa, và ghi dữ liệu vào các file tương ứng.

```
# myfs_formatter.py
def format_new_volume(myfs_dri_filepath_str, myfs_metadata_filepath_str,
volume_password_str):
    """Creates and formats new MyFS volume."""
    print("Starting MyFS volume formatting...")

    # Validate inputs
```

```
    if not volume_password_str:
        print("Error: Volume password cannot be empty.")
        return False

    if os.path.exists(myfs_dri_filepath_str) or
os.path.exists(myfs_metadata_filepath_str):
        print("Error: Target files already exist.")
        return False

    # Get computer info
    creation_timestamp = int(time.time())
    creating_machine_name = get_hostname()
    creating_machine_id_hash, raw_machine_id_str = get_machine_id_hash()

    if not raw_machine_id_str:
        print("Error: Failed to get computer fingerprint.")
        return False

    # Generate crypto material
    volume_password_salt_bytes = os.urandom(SALT_SIZE)
    kek_metadata_bytes = derive_key_pbkdf2(volume_password_str,
volume_password_salt_bytes)

    # Create volume files
    volume_id_bytes = uuid.uuid4().bytes

    dri_header_buffer = create_dri_header(
        volume_id_bytes, creation_timestamp, creating_machine_id_hash,
        creating_machine_name, volume_password_salt_bytes
    )

    try:
        # Write MyFS.DRI
        with open(myfs_dri_filepath_str, "wb") as f_dri:
            f_dri.write(dri_header_buffer)
            empty_ft_entry = bytearray(DRI_FT_ENTRY_SIZE)
            for i in range(DRI_FT_ENTRY_COUNT):
                f_dri.write(empty_ft_entry)
    except IOError as e:
        print(f"Error writing MyFS.DRI: {e}")
        return False

    # Write MyFS.METADATA
    try:
        plaintext_metadata = create_empty_supplemental_metadata()
        encrypted_iv = os.urandom(AES_IV_SIZE)
        encrypted_metadata = encrypt_aes_cbc(
            bytes(plaintext_metadata),
            kek_metadata_bytes,
            encrypted_iv
        )
        metadata_header = create_key_header(
            volume_id_bytes, volume_password_salt_bytes,
            encrypted_iv, len(encrypted_metadata)
```

```
        )
        with open(myfs_metadata_filepath_str, "wb") as f_metadata:
            f_metadata.write(metadata_header)
            f_metadata.write(encrypted_metadata)
            f_metadata.write(calculate_sha256(metadata_header +
encrypted_metadata))
    except IOError as e:
        print(f"Error writing MyFS.METADATA: {e}")
        return False

    print("Volume formatting completed successfully!")
    print(f"  MyFS.DRI created at: {myfs_dri_filepath_str}")
    print(f"  MyFS.METADATA created at: {myfs_metadata_filepath_str}")
    return True
```

7.2.2. Thay đổi Mật khẩu volume

Hàm `change_volume_password` trong module `myfs_security.py` cho phép thay đổi mật khẩu của volume, bao gồm việc xác thực mật khẩu cũ, giải mã metadata, tạo salt và IV mới, và mã hóa lại metadata với mật khẩu mới.

```
# myfs_security.py
def change_volume_password(self, old_password, new_password):
    """Changes the volume password."""
    if not self.header_data:
        self._read_volume_header()

    # Verify old password
    if not self.check_volume_password(old_password):
        raise ValueError("Incorrect old password.")

    try:
        # Import the metadata class here to avoid circular imports
        from myfs_metadata import MyFSMetadata

        # Load metadata using the old password with original salt/IV
        metadata_handler = MyFSMetadata(self.metadata_path)
        try:
            metadata_dict = metadata_handler.load(old_password)
        except Exception as e:
            print(f"Error loading metadata: {str(e)}")
            raise

        # Generate new salt and IV for the volume and metadata
        new_volume_salt = get_random_bytes(SALT_SIZE)
        new_metadata_iv = get_random_bytes(AES_IV_SIZE)

        # Update volume header with new salt
        with open(self.volume_path, "rb+") as f:
            # Skip to salt position (8 + 16 + 8 + 32 + 64 = 128 bytes)
            f.seek(128)
```

```
            f.write(new_volume_salt)

        # Create new metadata with the loaded data and new crypto parameters
        new_metadata_handler = MyFSMetadata(self.metadata_path)

        # Manually set header data for the new metadata handler
        new_metadata_handler.header_data = {
            "volume_id": metadata_handler.header_data["volume_id"],
            "salt": new_volume_salt,  # Use new volume salt for metadata
encryption
            "iv": new_metadata_iv,    # Use new IV for metadata encryption
            "metadata_size": 0,       # Will be updated during save
            "header_checksum": b""    # Will be updated during save
        }

        # Copy the metadata dictionary to the new handler
        new_metadata_handler.metadata = dict(metadata_handler.metadata)

        # Save metadata with the new password and new crypto parameters
        try:
            new_metadata_handler.save(new_password)
        except Exception as e:
            print(f"Error saving metadata: {str(e)}")
            # If save fails, restore the old salt in volume header
            with open(self.volume_path, "rb+") as f:
                f.seek(128)
                f.write(self.volume_password_salt)
            raise

        # Verify the change by trying to load with new password
        try:
            verify_handler = MyFSMetadata(self.metadata_path)
            verify_dict = verify_handler.load(new_password)
        except Exception as e:
            print(f"Verification failed: {str(e)}")
            # If verification fails, restore the old salt in volume header
            with open(self.volume_path, "rb+") as f:
                f.seek(128)
                f.write(self.volume_password_salt)
            raise Exception("Password change verification failed, changes
reverted")

        # Update stored data only after successful verification
        self.volume_password_salt = new_volume_salt
        self.header_data["volume_password_salt"] = new_volume_salt

        return True
    except Exception as e:
        print(f"Error in change_volume_password: {str(e)}")
        raise
```

7.2.3. Xác Thực Mật Khẩu

Hàm `check_volume_password` trong module `myfs_security.py` thực hiện việc xác thực mật khẩu volume bằng cách giải mã một phần nhỏ của metadata. Hàm này sử dụng salt được lưu trữ trong volume header và thử giải mã dữ liệu test để xác minh tính chính xác của mật khẩu.

```
# myfs_security.py
def check_volume_password(self, password):
    """Checks if the volume password is correct by decrypting a test value."""
    if not self.header_data:
        self._read_volume_header()

    try:
        # Read metadata file header
        with open(self.metadata_path, "rb") as f:
            # Skip magic number and volume ID (8 + 16 bytes)
            f.seek(24)

            # Read stored salt
            metadata_salt = f.read(16)

            # Verify salt matches volume
            if metadata_salt != self.volume_password_salt:
                return False

            # Read metadata IV
            metadata_iv = f.read(16)

            # Read metadata size
            metadata_size_bytes = f.read(8)
            metadata_size = struct.unpack("<q", metadata_size_bytes)[0]

            # Skip checksum (32 bytes)
            f.seek(24 + 16 + 16 + 8 + 32)

            # Read a small portion of encrypted metadata to verify password
            test_data = f.read(64)  # Just need a small portion

        # Derive key from password
        key = derive_key_pbkdf2(password, self.volume_password_salt)

        # Try to decrypt test data
        cipher = AES.new(key, AES.MODE_CBC, metadata_iv)
        try:
            cipher.decrypt(test_data)
            return True
        except:
            return False
    except:
        return False
```

7.3. Quản Lý Tập Tin

7.3.1. Liệt Kê Tập Tin

Hàm `list_files` trong module `myfs_connector.py` hiển thị danh sách các tập tin trong volume, bao gồm thông tin về tên, kích thước, trạng thái và tình trạng mã hóa. Hàm này hỗ trợ tùy chọn hiển thị cả các tập tin đã bị xóa.

```
# myfs_connector.py
def list_files(self, include_deleted=False):
    """List all files in the volume."""
    if not self.volume_password:
        raise ValueError("System not initialized with password.")

    # Get file list from file handler
    files = self.file_handler.list_files(self.volume_password, include_deleted)

    # Enhance with additional metadata
    for file_info in files:
        index = file_info['index']
        metadata = self.metadata.get_file_metadata(index)

        # Add metadata info to file_info
        file_info['original_path'] = metadata.get('original_path', '')
        file_info['is_encrypted'] = metadata.get('is_encrypted', False)

    return files
```

7.3.2. Import và Export file

Hàm `import_file` và `export_file` trong module `myfs_connector.py` thực hiện việc nhập và xuất tập tin vào/ra khỏi volume. Các hàm này hỗ trợ mã hóa tùy chọn cho từng tập tin, kiểm tra tính toàn vẹn, và xác thực mật khẩu khi cần thiết.

```
# myfs_connector.py
def import_file(self, filepath, file_password=None):
    """Import a file into the volume."""
    if not self.volume_password:
        raise ValueError("System not initialized with password.")

    # Import file using file handler
    file_index = self.file_handler.import_file(filepath, self.volume_password,
file_password)

    # Save changes to metadata
    self.save_changes()

    return file_index

def export_file(self, file_index, export_path, file_password=None,
use_original_path=False):
    """Export a file from the volume."""
```

```
    if not self.volume_password:
        raise ValueError("System not initialized with password.")

    # Export file using file handler
    exported_path = self.file_handler.export_file(
        file_index, export_path, self.volume_password, file_password,
use_original_path
    )

    return exported_path
```

7.3.3. Quản Lý Mật Khẩu Tập Tin

Hàm `set_file_password` trong module `myfs_connector.py` cho phép thiết lập hoặc thay đổi mật khẩu cho một tập tin cụ thể. Hàm này xử lý việc giải mã dữ liệu cũ và mã hóa lại với mật khẩu mới, đồng thời cập nhật metadata tương ứng.

```
# myfs_connector.py
def set_file_password(self, file_index, new_file_password,
old_file_password=None):
    """Set or change a file's password."""
    if not self.volume_password:
        raise ValueError("System not initialized with password.")

    # Change file password using file handler
    self.file_handler.set_file_password(file_index, self.volume_password,
new_file_password, old_file_password)

    # Save changes to metadata
    self.save_changes()

    return True
```

7.3.4. Xóa và Khôi Phục Tập Tin

Hàm `delete_file` và `recover_file` trong module `myfs_connector.py` thực hiện việc xóa và khôi phục tập tin trong volume. Hàm xóa hỗ trợ cả xóa thông thường (có thể khôi phục) và xóa vĩnh viễn, trong khi hàm khôi phục cho phép lấy lại các tập tin đã bị xóa thông thường.

```
# myfs_connector.py
def delete_file(self, file_index, permanent=False):
    """Delete a file from the volume."""
    if not self.volume_password:
        raise ValueError("System not initialized with password.")

    # Delete file using file handler
    self.file_handler.delete_file(file_index, self.volume_password, permanent)
```

```
        # Update metadata
        if permanent:
            self.metadata.delete_file_metadata(file_index, permanent=True)
        else:
            self.metadata.update_file_metadata(file_index, deleted=True)

        # Save changes to metadata
        self.save_changes()

        return True

    def recover_file(self, file_index):
        """Recover a deleted file."""
        if not self.volume_password:
            raise ValueError("System not initialized with password.")

        # Recover file using file handler
        self.file_handler.recover_file(file_index, self.volume_password)

        # Update metadata
        self.metadata.update_file_metadata(file_index, deleted=False)

        # Save changes to metadata
        self.save_changes()

        return True
```

7.4. Tự Động Kiểm Tra và Khôi Phục

7.4.1. Kiểm Tra Tính Toàn Vẹn

Hàm `verify_self_integrity` trong module `myfs.py` thực hiện việc kiểm tra tính toàn vẹn của ứng dụng và tự động sửa chữa nếu cần thiết. Hàm này kiểm tra các thành phần chính của hệ thống và thực hiện các biện pháp khôi phục khi phát hiện lỗi.

```
# myfs.py
def verify_self_integrity(self):
    """Verify system integrity and attempt auto-repair."""
    try:
        # Check if volume exists
        if not os.path.exists(self.volume_path):
            raise Exception("Volume file not found")

        # Check volume header
        with open(self.volume_path, "rb") as f:
            header = f.read(DRI_HEADER_SIZE)
            if len(header) != DRI_HEADER_SIZE:
                raise Exception("Invalid volume header size")

        # Check file table
        self._read_volume_header()
```

```
            if not self.header_data:
                raise Exception("Failed to read volume header")

            # Check metadata
            try:
                self._read_metadata(self.volume_password)
            except Exception as e:
                print(f"Warning: Metadata check failed: {str(e)}")
                # Attempt to repair metadata
                self._repair_metadata()

            # Check file data
            for i in range(self.header_data["ft_entry_count"]):
                entry = self._get_file_table_entry(i)
                if entry[0] == FileStatus.ACTIVE:
                    # Verify file data exists and is readable
                    data_offset = struct.unpack("<q",
entry[1+DRI_FT_FILENAME_HASH_LEN+32:1+DRI_FT_FILENAME_HASH_LEN+40])[0]
                    with open(self.volume_path, "rb") as f:
                        f.seek(data_offset)
                        # Just verify we can read the offset
                        f.read(1)

            return True

        except Exception as e:
            print(f"Integrity check failed: {str(e)}")
            return False
```

7.4.2. Tự Động Sửa Chữa

Hàm `perform_self_repair` trong module `myfs_self_repair.py` thực hiện việc tự động sửa chữa khi phát hiện lỗi trong hệ thống. Hàm này bao gồm các bước kiểm tra và sửa chữa cho từng thành phần của hệ thống.

```
# myfs_self_repair.py
def perform_self_repair(self):
    """Perform self-repair of the system."""
    try:
        # Create backup before repair
        backup_path = self.create_backup()
        if not backup_path:
            raise Exception("Failed to create backup before repair")

        # Repair volume header
        self._repair_volume_header()

        # Repair file table
        self._repair_file_table()

        # Repair metadata
        self._repair_metadata()
```

```
        # Verify repair was successful
        if not self.verify_integrity():
            # Restore from backup if repair failed
            self.restore_from_backup(backup_path)
            raise Exception("Self-repair failed, restored from backup")

        return True

    except Exception as e:
        print(f"Self-repair failed: {str(e)}")
        return False
```

7.4.3. Sao Lưu và Khôi Phục

Hàm `create_backup` và `restore_from_backup` trong module `myfs_self_repair.py` thực hiện việc tạo sao lưu và khôi phục từ sao lưu khi cần thiết. Các hàm này đảm bảo tính toàn vẹn của dữ liệu trong quá trình sửa chữa.

```
# myfs_self_repair.py
def create_backup(self):
    """Create a backup of the current volume."""
    try:
        # Generate backup filename with timestamp
        timestamp = time.strftime("%Y%m%d_%H%M%S")
        backup_filename = f"myfs_backup_{timestamp}.bak"
        backup_path = os.path.join(os.path.dirname(self.volume_path),
backup_filename)

        # Copy volume file to backup
        shutil.copy2(self.volume_path, backup_path)

        return backup_path

    except Exception as e:
        print(f"Backup creation failed: {str(e)}")
        return None

def restore_from_backup(self, backup_path):
    """Restore volume from backup."""
    try:
        if not os.path.exists(backup_path):
            raise Exception("Backup file not found")

        # Verify backup integrity
        if not self._verify_backup_integrity(backup_path):
            raise Exception("Backup integrity check failed")

        # Restore from backup
        shutil.copy2(backup_path, self.volume_path)
```

```
            return True

        except Exception as e:
            print(f"Restore from backup failed: {str(e)}")
            return False
```

7.4.4. Kiểm Tra và Sửa Chữa Metadata

Hàm `_repair_metadata` trong module `myfs_self_repair.py` thực hiện việc kiểm tra và sửa chữa metadata của hệ thống. Hàm này đảm bảo tính nhất quán của metadata với dữ liệu thực tế trong volume.

```
# myfs_self_repair.py
def _repair_metadata(self):
    """Repair metadata inconsistencies."""
    try:
        # Read current metadata
        current_metadata = self._read_metadata(self.volume_password)
        if not current_metadata:
            raise Exception("Failed to read current metadata")

        # Verify each file's metadata
        for i in range(self.header_data["ft_entry_count"]):
            entry = self._get_file_table_entry(i)
            if entry[0] == FileStatus.ACTIVE:
                # Check if metadata exists for this file
                if f"file_{i}" not in current_metadata:
                    # Recreate missing metadata
                    self._recreate_file_metadata(i, entry)

                # Verify metadata consistency
                self._verify_file_metadata(i, entry,
current_metadata[f"file_{i}"])

        # Save repaired metadata
        self._save_metadata(self.volume_password)

        return True

    except Exception as e:
        print(f"Metadata repair failed: {str(e)}")
        return False
```

8. Kết Luận

MyFS đã đáp ứng đầy đủ các yêu cầu về bảo mật, toàn vẹn dữ liệu và khả năng phục hồi. Hệ thống cung cấp một giải pháp lưu trữ an toàn với các tính năng:

- Mã hóa mạnh mẽ
- Xác thực nhiều lớp

- Tự động sửa chữa
- Khôi phục dữ liệu
- Giao diện CLI dễ sử dụng