

# Sistema de Backup Seguro con Dask

Proyecto Final - Sistemas Operativos (ST0257)

Universidad EAFIT

---

## Integrantes del Equipo

- **David Lopera Londoño**
  - **Verónica Zapata Vargas**
  - **Juan Diego Acuña Giraldo**
- 

## 1. Resumen Ejecutivo

### 1.1. Objetivo del Proyecto

El **Sistema de Backup Seguro** es una solución integral desarrollada para la asignatura de Sistemas Operativos que implementa un sistema completo de respaldo con características empresariales. El proyecto combina algoritmos de compresión clásicos, encriptación de grado militar y procesamiento paralelo para crear una herramienta robusta y eficiente.

### 1.2. Características Clave

- **Múltiples algoritmos de compresión:** ZIP, GZIP, BZIP2
- **Encriptación AES-256:** Seguridad de nivel militar
- **Procesamiento paralelo:** Optimización con Dask
- **Almacenamiento flexible:** Local, fragmentado y nube
- **Interface intuitiva:** CLI completa con ayuda detallada
- **Testing exhaustivo:** Suite completa de pruebas unitarias

### 1.3. Innovaciones Técnicas

1. **Paralelismo inteligente** con Dask para optimizar recursos del sistema
2. **Fragmentación automática** para distribución en múltiples dispositivos
3. **Validación de integridad** con checksums MD5
4. **Arquitectura modular** que facilita mantenimiento y extensión

### 1.4. Métricas del Proyecto

- **+2,500 líneas de código** Python bien documentado
- **25+ tests unitarios** con cobertura ~85%
- **4 módulos core** completamente funcionales

- **3 algoritmos de compresión** implementados
  - **Support para 2 servicios de nube** (modo simulado)
- 

## 2. Estructura del Proyecto

```
SO-ProyectoFinal/
├── src/ ..... # Código fuente principal
│   ├── core/ ..... # Módulos principales
│   │   ├── scanner.py ..... # Escaneo de directorios
│   │   ├── compressor.py ..... # Algoritmos de compresión
│   │   ├── encryptor.py ..... # Encriptación AES-256
│   │   ├── storage.py ..... # Gestión de almacenamiento
│   │   └── restore.py ..... # Restauración de backups
│   ├── utils/ ..... # Utilidades y helpers
│   │   ├── logger.py ..... # Sistema de logging
│   │   ├── error_handler.py ..... # Manejo de errores
│   │   ├── parallel.py ..... # Utilidades de paralelismo
│   │   └── rebuild_generator.py # Generación de scripts
│   └── main.py ..... # Punto de entrada principal
├── tests/ ..... # Suite de testing
│   ├── test_scanner.py ..... # Tests del scanner
│   ├── test_compressor.py ..... # Tests del compressor
│   └── test_requirements.py ..... # Tests de requisitos
├── backups/ ..... # Directorio de backups
├── restored/ ..... # Directorio de restauración
├── logs/ ..... # Archivos de log
├── run_tests.py ..... # Ejecutor de tests
├── test.mk ..... # Makefile de testing
├── requirements.txt ..... # Dependencias
└── README.md ..... # Documentación principal
```

---

## 3. Características Principales

### 3.1. Algoritmos de Compresión Clásicos

#### 3.1.1. ZIP - Rápido y Universal

- **Compatibilidad:** Soportado en todos los sistemas operativos
- **Velocidad:** Excelente balance velocidad/compresión
- **Uso ideal:** Backups generales y distribución

#### 3.1.2. GZIP - Estándar de la Industria

- **Eficiencia:** Balance óptimo velocidad/compresión

- **Formato:** TAR.GZ para múltiples archivos
- **Uso ideal:** Archivos de texto y código fuente

### 3.1.3. BZIP2 - Máxima Compresión

- **Compresión:** La mejor ratio de compresión
- **Formato:** TAR.BZ2 para múltiples archivos
- **Uso ideal:** Archivos grandes con tiempo disponible

## 3.2. Seguridad Robusta

### 3.2.1. Encriptación AES-256

- **Estándar:** Encriptación de grado militar
- **Clave:** 256 bits de longitud
- **Algoritmo:** Advanced Encryption Standard

### 3.2.2. Derivación de Claves PBKDF2

- **Iteraciones:** 100,000 iteraciones por defecto
- **Salt:** Sal aleatoria para cada archivo
- **Hash:** SHA-256 como función base

### 3.2.3. Validación de Integridad

- **Checksums:** MD5 para cada fragmento
- **Verificación:** Automática durante restauración
- **Detección:** Corrupción de datos inmediata

## 3.3. Almacenamiento Flexible

### 3.3.1. Local/Disco Externo

- **Detección:** Automática de dispositivos externos
- **Verificación:** Integridad post-copia
- **Permisos:** Manejo robusto de permisos

### 3.3.2. Fragmentación USB

- **División:** Archivos grandes en fragmentos configurables
- **Metadatos:** JSON con información completa
- **Scripts:** Generación automática de rebuild.py
- **Portabilidad:** Cada fragmento independiente

### 3.3.3. Nube Simulada

- **Google Drive:** Integración simulada
- **Dropbox:** Soporte básico
- **Escalabilidad:** Preparado para APIs reales

## 3.4. Paralelismo con Dask

### 3.4.1. Compresión Paralela

- **Workers:** Configurables (1-16)
- **Chunks:** Procesamiento por fragmentos
- **Memoria:** Gestión eficiente de recursos

### 3.4.2. Escalabilidad Automática

- **CPU:** Detección automática de cores
- **Memoria:** Límites configurables
- **I/O:** Operaciones paralelas de lectura/escritura

### 3.4.3. Optimización Inteligente

- **SSD:** Hasta 8 workers recomendados
- **HDD:** 2-4 workers óptimos
- **USB:** 1-2 workers para evitar saturación

---

## 4. Tecnologías Utilizadas

Componente	Tecnología	Versión	Propósito	Beneficios
Paralelismo	Dask	2023.5.0+	Procesamiento distribuido	Escalabilidad y rendimiento
Encriptación	Cryptography	41.0.0+	Seguridad AES-256	Estándar de la industria
Interfaz	Argparse	Nativo	CLI intuitiva	Facilidad de uso
Progreso	tqdm	4.64.0+	Barras de progreso	Feedback visual
Logs	Logging	Nativo	Trazabilidad completa	Debugging eficiente
Requests	Requests	2.31.0+	APIs REST	Integración cloud
Testing	unittest	Nativo	Pruebas automatizadas	Calidad de código

---

## 5. Instalación y Configuración

### 5.1. Prerrequisitos del Sistema

- **Python:** 3.13 o superior
- **RAM:** 4GB mínimo (8GB recomendado)
- **Espacio:** 500MB para instalación
- **OS:** Windows, Linux, macOS

## 5.2. Instalación Rápida

bash

```
# 1. Clonar repositorio
git clone https://github.com/equipo/SO-ProyectoFinal.git
cd SO-ProyectoFinal

# 2. Crear entorno virtual (recomendado)
python -m venv backup_env
source backup_env/bin/activate # Linux/Mac
# backup_env\Scripts\activate # Windows

# 3. Instalar dependencias
pip install -r requirements.txt

# 4. Verificar instalación
python -m src.main --help
```

## 5.3. Verificación de Instalación

bash

```
# Test rápido del sistema
python -m src.main backup -d ./test_data -o test_backup.zip

# Ejecutar suite de tests
python run_tests.py

# Verificar todos Los módulos
python -c "from src.core import scanner, compressor, encryptor, storage; print('Todos los módul
```

---

## 6. Guía de Uso Completa

### 6.1. Backup Básico - Múltiples Carpetas

```
bash

# Ejemplo básico con 3 carpetas
python -m src.main backup \
... -d ./documentos ./proyectos ./fotos \
... -o backup_completo.zip \
... -a zip
```

```
# Con información detallada
python -m src.main backup \
... -d ./documentos ./proyectos ./fotos \
... -o backup_completo.zip \
... -a zip --workers 6 -v
```

### Resultado esperado:

```
Iniciando proceso de backup...
Encontrados 1,247 archivos en 3 carpeta(s)
Comprimiendo con zip...
Backup completado: backup_completo.zip (234.5 MB)
```

## 6.2. Backup con Encriptación AES-256

```
bash

# Encriptación interactiva (recomendado)
python -m src.main backup \
... -d ./datos_importantes \
... -o backup_seguro.zip.enc \
... -e
```

```
# Encriptación con contraseña en comando
python -m src.main backup \
... -d ./datos_importantes \
... -o backup_seguro.zip.enc \
... -e --password "MiClaveSegura123"
```

### Proceso de encriptación:

```
Ingrese contraseña para encriptación (mín. 8 caracteres): *****
Confirme la contraseña: *****
Aplicando encriptación AES-256...
Archivo encriptado: backup_seguro.zip.enc
```

## 6.3. Backup Fragmentado para USB

```
bash

# Fragmentos de 1GB para USBs múltiples
python -m src.main backup \
... -d ./archivo_grande \
... -o backup_usb \
... -s fragments \
... --fragment-size 1024
```

```
# Fragmentos pequeños con encriptación
python -m src.main backup \
... -d ./multimedia \
... -o backup_multimedia \
... -s fragments \
... --fragment-size 500 \
... -e
```

## Resultado de fragmentación:

```
Carpeta de backup creada: backup_fragments_1
Fragmentación completada:
... Directorio: backup_fragments_1/fragments
... 4 fragmentos de máximo 1024MB
... Metadatos: backup.metadata.json
... Scripts de reconstitución incluidos
```

## 6.4. Rebuild de Backup Fragmentado

```
bash

# Navegar al directorio de fragmentos
cd backups/backup_fragments_1/fragments

# Ejecutar script de reconstrucción
python rebuild.py

# Opciones adicionales
python rebuild.py info ... # Ver información de fragmentos
python rebuild.py help ... # Mostrar ayuda
```

## Scripts multiplataforma incluidos:

- `rebuild.py` - Script principal de Python
- `rebuild.bat` - Script para Windows
- `rebuild.sh` - Script para Linux/macOS

- `README.md` - Instrucciones completas

## 6.5. Backup a la Nube (Simulado)

bash

```
# Google Drive simulado
python -m src.main backup \
... -d ./mi_proyecto \
... -o backup_nube.zip \
... -s cloud \
... --cloud-service gdrive

# Dropbox con carpeta específica
python -m src.main backup \
... -d ./documentos \
... -o backup_docs.zip \
... -s cloud \
... --cloud-service dropbox \
... --cloud-folder "Backups/2025"
```

## 6.6. Restauración Completa

bash

```
# Restaurar backup normal
python -m src.main restore \
... -i backup_completo.zip \
... -o ./restaurado

# Restaurar backup encriptado
python -m src.main restore \
... -i backup_seguro.zip.enc \
... -o ./restaurado \
... --password "MiClaveSegura123"

# Restauración con solicitud interactiva de contraseña
python -m src.main restore \
... -i backup_seguro.zip.enc \
... -o ./restaurado
```

**Proceso de restauración:**

```
Iniciando proceso de restauración...
Descriptando archivo...
Extrayendo contenido...
Restauración completada
Archivos restaurados en: ./restaurado
Algunos archivos restaurados:
    ./restaurado/documentos/reporte.pdf
    ./restaurado/proyectos/main.py
```

---

## 7. Sistema de Testing

### 7.1. Arquitectura de Testing

```
tests/
├── test_scanner.py ..... # 15+ tests del módulo scanner
├── test_compressor.py ..... # 20+ tests del módulo compressor
└── test_requirements.py ..... # 3 tests de requisitos principales
```

### 7.2. Tests del Scanner (test\_scanner.py)

#### 7.2.1. Funcionalidades Probadas

- **Escaneo de directorios únicos**
- **Escaneo de múltiples directorios**
- **Procesamiento paralelo vs secuencial**
- **Manejo de directorios inexistentes**
- **Directorios vacíos**
- **Archivos con nombres Unicode**
- **Estructuras grandes de directorios**

```
python
```

```
def test_scan_single_directory(self):
    """Prueba el escaneo de un solo directorio"""
    files = scanner.scan_directory(self.test_dir)
    self.assertGreater(len(files), 0)
    # Verifica escaneo recursivo
    self.assertIn('archivo_sub.md', [os.path.basename(f) for f in files])
```

#### 7.2.2. Casos Edge Probados

- Directorios con permisos restringidos

- Archivos con caracteres especiales (café.txt, niño.py, corazón.md)
- Estructuras profundas de directorios (10+ niveles)
- Directorios con miles de archivos

## 7.3. Tests del Compressor (test\_compressor.py)

### 7.3.1. Algoritmos Probados

- **Compresión ZIP:** Archivos únicos y múltiples
- **Compresión GZIP:** TAR.GZ para múltiples archivos
- **Compresión BZIP2:** TAR.BZ2 con máxima compresión

### 7.3.2. Tests de Rendimiento

python

```
def test_parallel_vs_sequential_performance(self):
    """Compara rendimiento paralelo vs secuencial"""
    # Mide tiempos y calcula speedup
    speedup = sequential_time / parallel_time
    self.assertGreater(speedup, 0.8) # AL menos 80% de eficiencia
```

### 7.3.3. Casos Avanzados

- **Archivos grandes** (10MB+)
- **Operaciones concurrentes**
- **Diferentes números de workers** (1, 2, 4, 8)
- **Manejo de memoria** con archivos múltiples
- **Archivos binarios** vs texto

## 7.4. Tests de Requisitos (test\_requirements.py)

### 7.4.1. Requisito 1: Múltiples Carpetas

python

```
def test_multiple_folders_backup():
    """Valida selección de múltiples carpetas"""
    # 1. Crea: documentos/ proyectos/ imágenes/
    # 2. Ejecuta: backup -d "carpeta1" "carpeta2" "carpeta3"
    # 3. Verifica: todas incluidas en backup
    return success # ÉXITO o FALLO
```

### 7.4.2. Requisito 2: Encriptación AES-256

```

python

def test_encryption_backup():
    """Valida encriptación opcional"""
    # 1. Crea backup con -e --password
    # 2. Verifica archivo .enc generado
    # 3. Restaura con contraseña correcta
    # 4. Verifica contenido restaurado
    return success # ÉXITO o FALLO

def test_encryption_wrong_password():
    """Valida rechazo de contraseña incorrecta"""
    # 1. Intenta restaurar con contraseña incorrecta
    # 2. Verifica que falla apropiadamente
    return success # ÉXITO o FALLO

```

## 7.5. Ejecución de Tests

### 7.5.1. Métodos de Ejecución

#### 1. Script Principal (Recomendado)

```

bash

# Ejecutar todas las pruebas
python run_tests.py

# Ejecutar módulo específico
python run_tests.py --module scanner
python run_tests.py --module compressor

# Solo pruebas de rendimiento
python run_tests.py --performance

# Modo verbose para debugging
python run_tests.py --verbose

```

#### 2. Tests de Requisitos Individuales

```

bash

# Validación completa de requisitos principales
python tests/test_requirements.py

```

#### 3. Pipeline de Calidad con Makefile

```
bash

# Tests básicos
make -f test.mk test

# Tests + formato + Linting
make -f test.mk quality

# Solo pruebas de rendimiento
make -f test.mk test-performance

# Limpieza de archivos temporales
make -f test.mk clean-test
```

## 7.5.2. Salida de Tests Exitosos

## INICIANDO SUITE COMPLETA DE PRUEBAS

```
=====
```

EJECUTANDO PRUEBAS: scanner

```
=====
```

```
test_scan_single_directory ... OK
test_scan_multiple_directories_parallel ... OK
test_scan_nonexistent_directory ... OK
test_scan_empty_directory ... OK
test_unicode_filenames ... OK
```

Ran 15 tests in 2.345s - OK

EJECUTANDO PRUEBAS: compressor

```
=====
```

```
test_compress_single_file_zip ... OK
test_compress_multiple_files_zip ... OK
test_compression_ratio_comparison ... OK
test_parallel_vs_sequential_performance ... OK
```

Ran 20 tests in 8.234s - OK

## RESUMEN DE RESULTADOS

```
=====
```

Scanner..... PASÓ

Compressor..... PASÓ

Total: 2/2 módulos pasaron todas las pruebas

TODAS LAS PRUEBAS PASARON EXITOSAMENTE!

### 7.5.3. Validación de Requisitos

bash

```
python tests/test_requirements.py
```

## VALIDACIÓN DE REQUISITOS DEL SISTEMA DE BACKUP

---

### PRUEBA 1: Selección de múltiples carpetas

---

Carpetas de prueba creadas en: /tmp/backup\_test\_xyz

... documentos: 4 archivos

... proyectos: 6 archivos

... imágenes: 2 archivos

ÉXITO: Backup de múltiples carpetas completado

Archivo creado: backup\_multiple.zip (2.34 MB)

### PRUEBA 2: Encriptación opcional con AES-256

---

Ejecutando backup con encriptación...

ÉXITO: Backup con encriptación completado

Archivo encriptado: backup\_encrypted.enc (2.45 MB)

Probando restauración con contraseña...

ÉXITO: Restauración con desencriptación completada

Archivos restaurados: 12

### PRUEBA 3: Verificación de contraseña incorrecta

---

ÉXITO: El sistema rechazó correctamente la contraseña incorrecta

## RESUMEN DE RESULTADOS

---

Selección de múltiples carpetas ... PASÓ

Encriptación AES-256 ..... PASÓ

Validación de contraseña ..... PASÓ

Total: 3/3 pruebas pasaron

TODOS LOS REQUISITOS VALIDADOS EXITOSAMENTE!

El sistema cumple con:

... Selección de múltiples carpetas con escaneo recursivo

... Encriptación opcional con AES-256

... Validación correcta de contraseñas

---

## 8. Interfaz de Usuario

### 8.1. Sistema de Ayuda Completo

#### 8.1.1. Ayuda Principal

bash

```
python -m src.main --help
```

## SISTEMA DE BACKUP SEGURO v1.0

---

Un sistema completo de respaldo con encriptación AES-256, compresión avanzada y múltiples opciones de almacenamiento.

### CARACTERÍSTICAS PRINCIPALES:

- .. Respaldo de múltiples carpetas simultáneamente
- .. Encriptación AES-256 de grado militar
- .. Compresión con algoritmos ZIP, GZIP y BZIP2
- .. Procesamiento paralelo con Dask para máximo rendimiento
- .. Almacenamiento local para discos externos
- .. Fragmentación automática para distribución en USBs
- .. Integración con Google Drive y Dropbox

### CASOS DE USO TÍPICOS:

- Respaldo completo del sistema antes de actualizaciones críticas
- Migración segura de datos entre computadoras
- Archivo a largo plazo de proyectos y documentos importantes

### OPCIONES GLOBALES:

- .. -h, --help ..... Mostrar esta ayuda completa y salir
- .. -v, --verbose ..... Activar modo detallado con información de depuración
- .. --workers N ..... Número de procesos paralelos (1-16, default: 4)

### COMANDOS DISPONIBLES:

- .. backup ..... Crear un respaldo completo y seguro
- .. restore ..... Restaurar un backup existente

Para ayuda específica: python -m src.main COMANDO --help

### 8.1.2. Ayuda del Comando Backup

bash

```
python -m src.main backup --help
```

## COMANDO BACKUP - Creación de Respaldos Seguros

---

### DIRECTORIOS Y SALIDA (Requeridos):

-d, --directories DIR [DIR ...] Directorios para respaldar  
-o, --output ARCHIVO Archivo/directorio de salida

### COMPRESIÓN:

-a, --algorithm {zip,gzip,bzip2} Algoritmo de compresión:  
• zip - Rápido, compatible (default)  
• gzip - Buena compresión, estándar  
• bzip2 - Máxima compresión, más lento

### SEGURIDAD Y ENCRIPCIÓN:

-e, --encrypt Activar encriptación AES-256  
--password PASS Contraseña (mejor usar modo interactivo)

### MODOS DE ALMACENAMIENTO:

-s, --storage {local,cloud,fragments} Modo de almacenamiento:  
• local - Archivo Único (default)  
• cloud - Google Drive/Dropbox  
• fragments - Dividir para USB

### CONFIGURACIÓN DE NUBE:

--cloud-service {gdrive,dropbox} Servicio de nube  
--cloud-folder CARPETA Carpeta destino en la nube

### CONFIGURACIÓN DE FRAGMENTOS:

--fragment-size MB Tamaño por fragmento (default: 1024)  
Recomendados:  
• USB 1GB : 900 MB  
• USB 4GB : 3800 MB

## 8.2. Ejemplos Contextuales Integrados

## EJEMPLOS DE USO COMPLETOS:

### RESPALDO BÁSICO LOCAL:

```
... python -m src.main backup -d ./documentos -o backup_docs.zip
```

### RESPALDO CON ENCRIPCIÓN:

```
python -m src.main backup -d ./privado -o backup_seguro.zip.enc -e
```

### FRAGMENTACIÓN PARA USB:

```
python -m src.main backup -d ./sistema -o backup -s fragments --fragment-size 500
```

### ALMACENAMIENTO EN LA NUBE:

```
... python -m src.main backup -d ./proyectos -o backup.zip -s cloud --cloud-service gdrive
```

### OPTIMIZACIÓN DE RENDIMIENTO:

- Para SSD rápidos: --workers 6-8
- Para HDD tradicionales: --workers 2-4
- Para fragmentación: ajustar fragment-size según USB

## 8.3. Feedback Visual Rico

### 8.3.1. Información Detallada en Tiempo Real

```
Iniciando proceso de backup...
Directorios a respaldar: ./documentos, ./proyectos, ./OTOS
Algoritmo: zip
Encriptación: SI (AES-256)
Workers: 6
Modo: Almacenamiento Local/Disco Externo
```

```
Escaneando directorios...
Encontrados 1,247 archivos en 3 carpeta(s)
```

```
Comprimiendo con zip...
La encriptación AES-256 se aplicará automáticamente...
```

```
Compresión completada: /tmp/backup_temp_xyz/backup_temp.zip
```

```
Iniciando almacenamiento...
Archivo almacenado localmente: backup_completo.zip.enc
```

```
BACKUP COMPLETADO EXITOSAMENTE
```

```
Carpetas respaldadas: 3
Archivos procesados: 1,247
Archivo final: backup_completo.zip.enc
Tamaño: 234.56 MB
Encriptación: AES-256 aplicada
```

Próximos pasos:

- El archivo está listo para usar
- Para restaurar: python -m src.main restore -i "backup\_completo.zip.enc" -o ./restaurado

---

## 9. Arquitectura Técnica Detallada

### 9.1. Módulos Core

#### 9.1.1. Scanner.py - Escaneo Inteligente

```
python

def scan_directories(directories, parallel=True):
    """
    ... Escanea múltiples directorios con paralelismo Dask
    ... - Procesamiento paralelo de múltiples directorios
    ... - Manejo robusto de errores
    ... - Fallback automático a modo secuencial
    """
```

## Características:

- **Escaneo recursivo** completo de directorios
- **Paralelismo con Dask** para directorios múltiples
- **Manejo de errores** gracioso con fallback
- **Validación de rutas** y permisos

### 9.1.2. Compressor.py - Compresión Paralela

python

```
def compress_files(files, algorithm='zip', encrypt=False, workers=4):
    """
    ... Comprime archivos con algoritmos múltiples y paralelismo
    ... - ZIP, GZIP, BZIP2 con workers configurables
    ... - Integración con encriptación automática
    ... - Manejo inteligente de rutas y conflictos
    """

```

## Innovaciones técnicas:

- **Cliente Dask optimizado** con configuración automática
- **Detección de conflictos** de rutas fuente/destino
- **Manejo de rutas relativas** inteligente
- **Integración transparente** con encriptación

### 9.1.3. Encryptor.py - Seguridad Empresarial

python

```
def encrypt_file(file_path, output_path, password, workers=4):
    """
    ... Encriptación AES-256 con paralelismo por chunks
    ... - PBKDF2 con 100,000 iteraciones
    ... - Procesamiento paralelo de chunks
    ... - Salt único por archivo
    """

```

## Seguridad implementada:

- **AES-256-CBC** con IV único por chunk
- **PBKDF2-SHA256** con 100,000 iteraciones
- **Salt aleatorio** de 16 bytes por archivo
- **Paralelismo seguro** por chunks

#### 9.1.4. Storage.py - Almacenamiento Flexible

python

```
def fragment_file(source_file, fragment_size_mb=1024, output_dir=None):
    """
    Fragmentación inteligente para dispositivos múltiples
    - Metadatos JSON completos
    - Scripts de reconstrucción automáticos
    - Verificación de integridad MD5
    """
    ...
```

#### Características avanzadas:

- **Fragmentación inteligente** con metadatos JSON
- **Verificación de espacio** disponible antes de fragmentar
- **Checksums MD5** para cada fragmento individual
- **Scripts multiplataforma** (Python, Batch, Bash)
- **Detección automática** de dispositivos externos

#### 9.1.5. Restore.py - Recuperación Robusta

python

```
def restore_backup(backup_path, output_dir, password=None):
    """
    Restauración universal de todos los tipos de backup
    - Detección automática de formato
    - Desencriptación transparente
    - Reconstrucción de fragmentos
    """
    ...
```

#### Capacidades de restauración:

- **Detección automática** de tipo de archivo (.zip, .enc, fragmentos)
- **Desencriptación transparente** con manejo de errores
- **Preservación de estructura** de directorios original
- **Validación de integridad** durante restauración

### 9.2. Utilidades Avanzadas

#### 9.2.1. Logger.py - Trazabilidad Completa

- **Niveles configurables:** DEBUG, INFO, WARNING, ERROR

- **Archivos rotatorios:** Logs con timestamp automático
- **Output dual:** Consola + archivo simultáneo
- **Thread-safe:** Seguro para operaciones paralelas

## 9.2.2. Error\_handler.py - Manejo Robusto

python

```
@retry(max_attempts=3, exceptions=(ConnectionError,), delay=2.0)
@safe_cloud_operation
def upload_to_cloud(file_path, service):
    """Decoradores para manejo automático de errores"""

```

### Patrones implementados:

- **Decorador @retry** para operaciones de red
- **Excepciones específicas** por tipo de error
- **Fallback automático** para operaciones críticas
- **Logging contextual** de todos los errores

## 9.2.3. Rebuild\_generator.py - Generación de Scripts

- **Scripts Python:** Lógica principal de reconstrucción
- **Batch Windows:** Interfaz amigable para Windows
- **Bash Unix:** Compatibilidad Linux/macOS completa
- **README automático:** Documentación incluida

## 9.3. Arquitectura de Datos

### 9.3.1. Formato de Metadatos JSON

```

json

{
    "original_file": "/path/to/source.zip",
    "file_size": 1073741824,
    "fragment_size": 1073741824,
    "fragment_size_mb": 1024,
    "num_fragments": 3,
    "created_by": "Sistema de Backup Seguro v1.0",
    "fragments": [
        "backup.part000": {
            "size": 1073741824,
            "checksum": "d41d8cd98f00b204e9800998ecf8427e",
            "index": 0
        }
    ]
}

```

### 9.3.2. Estructura de Logs

```

2025-01-20 15:30:45 - secure_backup - INFO - Iniciando backup de 3 directorios
2025-01-20 15:30:46 - secure_backup - DEBUG - Escaneando directorio: ./documentos
2025-01-20 15:30:47 - secure_backup - INFO - Encontrados 1247 archivos
2025-01-20 15:30:48 - secure_backup - INFO - Iniciando compresión ZIP con 6 workers
2025-01-20 15:31:15 - secure_backup - INFO - Compresión completada: 234.56 MB
2025-01-20 15:31:16 - secure_backup - INFO - Aplicando encriptación AES-256
2025-01-20 15:31:45 - secure_backup - INFO - Backup completado exitosamente

```

---

## 10. Análisis de Seguridad

### 10.1. Implementación Criptográfica

#### 10.1.1. AES-256-CBC

```

python

# Configuración de encriptación
algorithm = algorithms.AES(key) # 256-bit key
mode = modes.CBC(iv) # 16-byte IV único por chunk
cipher = Cipher(algorithm, mode, backend=default_backend())

```

#### Fortalezas implementadas:

- **Clave de 256 bits:** Estándar NSA para información "Top Secret"
- **IV único por chunk:** Previene ataques de análisis de patrones

- **Padding PKCS7:** Estándar de la industria
- **Backend certificado:** Cryptography library con validación FIPS

### 10.1.2. Derivación de Claves PBKDF2

```
python

kdf = PBKDF2HMAC(
    algorithm=hashes.SHA256(),
    length=32,           # 256 bits
    salt=os.urandom(16),      # 128-bit random salt
    iterations=100000,       # Protección contra ataques de diccionario
    backend=default_backend()
)
```

#### Protecciones implementadas:

- **100,000 iteraciones:** Resistente a ataques de fuerza bruta
- **Salt aleatorio:** Único por archivo, previene rainbow tables
- **SHA-256:** Hash criptográficamente seguro
- **Entropía de OS:** Salt generado por os.urandom()

## 10.2. Análisis de Vulnerabilidades

### 10.2.1. Protecciones Implementadas

- **Password timing attacks:** Validación constante en tiempo
- **IV reutilización:** IV único generado por chunk
- **Salt collision:** Generación criptográficamente aleatoria
- **Memory leaks:** Limpieza explícita de variables sensibles

### 10.2.2. Consideraciones de Seguridad

- **Password storage:** No almacenado, solo en memoria durante ejecución
- **Temp files:** Limpieza automática de archivos temporales
- **Key derivation:** PBKDF2 podría actualizarse a Argon2 en versiones futuras
- **Side-channel:** Implementación resistente a ataques de timing básicos

## 11. Conclusiones del Proyecto

### 11.1. Objetivos Alcanzados

#### 11.1.1. Requisitos Funcionales Completos

El proyecto cumple exitosamente con todos los requisitos establecidos:

1. **Selección de múltiples carpetas**: Implementado con escaneo recursivo paralelo
2. **Encriptación AES-256 opcional**: Implementado con PBKDF2 y validación robusta
3. **Algoritmos de compresión**: ZIP, GZIP, BZIP2 con optimización automática
4. **Paralelismo**: Dask integrado con configuración inteligente de workers
5. **Interface CLI**: Sistema de ayuda completo y ejemplos contextuales

### 11.1.2. Innovaciones Más Allá de Requisitos

El proyecto supera las expectativas originales con características avanzadas:

- **Fragmentación automática**: Sistema completo para distribución en USBs múltiples
- **Scripts de reconstrucción**: Generación automática multiplataforma
- **Validación de integridad**: MD5 checksums automáticos por fragmento
- **Almacenamiento en nube**: Arquitectura preparada para servicios cloud reales
- **Suite de testing**: 35+ tests automatizados con validación de requisitos

## 11.2. Aprendizajes Técnicos Clave

### 11.2.1. Desarrollo de Software

- **Paralelismo real**: Implementación práctica con Dask para operaciones CPU-intensivas
- **Criptografía aplicada**: AES-256 y PBKDF2 en contexto real de seguridad de datos
- **Arquitectura modular**: Diseño que facilita testing, mantenimiento y extensión
- **Error handling**: Patrones robustos para operaciones de archivo y red

### 11.2.2. Sistemas Operativos

- **File I/O optimizado**: Manejo eficiente de operaciones de archivo a gran escala
- **Resource management**: Gestión inteligente de CPU, memoria y almacenamiento
- **Cross-platform development**: Compatibilidad real entre sistemas operativos
- **Process coordination**: Coordinación efectiva de múltiples workers

### 11.2.3. Seguridad Informática

- **Encryption best practices**: Implementación correcta de algoritmos estándar
- **Key management**: Derivación y manejo seguro de claves criptográficas
- **Threat modeling**: Análisis y mitigación de vectores de ataque comunes
- **Security testing**: Validación automática de características de seguridad

## 11.3. Reflexión Final

El **Sistema de Backup Seguro** representa más que un proyecto académico: es una herramienta funcional que combina múltiples disciplinas de la informática en una solución coherente y práctica.

### Logros destacados:

- **Implementación completa** de todos los requisitos con calidad profesional
- **Innovaciones técnicas** que superan las expectativas originales
- **Arquitectura escalable** preparada para evolución futura
- **Documentación exhaustiva** que facilita uso y mantenimiento
- **Testing riguroso** que garantiza confiabilidad

### Competencias demostradas:

- **Análisis de requisitos** y traducción a implementación técnica
- **Diseño de arquitectura** modular y mantenable
- **Implementación técnica** con herramientas modernas de la industria
- **Testing y validación** automatizada de calidad
- **Documentación técnica** completa y profesional

El proyecto establece una base sólida para futuras innovaciones en el área de backup y almacenamiento de datos, mientras demuestra competencias técnicas avanzadas en desarrollo de software, seguridad informática y sistemas operativos.

---

## 12. Referencias y Recursos

### 12.1. Documentación Técnica

- [Python Cryptography Library](#) - Implementación de AES-256
- [Dask Documentation](#) - Paralelismo y computación distribuida
- [NIST SP 800-132](#) - Recomendaciones PBKDF2
- [RFC 3394](#) - AES Key Wrap Algorithm

### 12.2. Estándares de Seguridad

- [FIPS 140-2](#) - Security Requirements for Cryptographic Modules
- [NIST SP 800-38A](#) - Block Cipher Modes of Operation

### 12.3. Herramientas y Librerías

- [Python unittest](#) - Framework de testing
- [argparse](#) - CLI argument parsing

- pathlib - Path manipulation
- 

© 2025 - Sistema de Backup Seguro

Proyecto Final - Sistemas Operativos (ST0257)

Universidad EAFIT