INFORME TECNICO - SAFEKEY VAULT+

Date	@December 6, 2025
Status	Done

1. INTRODUCCIÓN.

SAFEKEY VAULT+ es un sistema de gestion de contrasenas desarrollado en Python que implementa principios de seguridad informaica, diseno modular y programación recursiva. El sistema permite almacenar, cifrar y gestionar contrasenas de multiples servicios de forma segura mediante consola.

Caracteristicas principales que tiene:

- Cifrado de contraseñas con dos metodos diferentes
- Análisis de fortaleza de contraseñas
- Generador de contrasenas seguras
- Sistema de auditoría completo
- Búsqueda recursiva inteligente
- Validación de integridad

2. DISEÑO MODULAR.

El sistema está organizado en 7 modulos independientes que trabajan en conjunto:

Modulo 1: Cifrado y Descifrado.

Responsabilidad: Implementar algoritmos de cifrado para proteger contraseñas

Funciones principales:

- `cifrado_cesar()` - Cifrado por desplazamiento de caracteres.
- `descifrado_cesar()` - Descifrado del método César.
- `invertir_texto_recursivo()` - Inversión recursiva de texto.
- `cifrado_recursivo()` - Cifrado combinado (inversión + desplazamiento).
- `descifrado_recursivo()` - Descifrado del método recursivo.

Importaciones: Ninguna (Use funciones base de Python).

Modulo 2: Validacion y Analisis de Contrasenas.

Responsabilidad: Evaluar la fortaleza de las contraseñas

Funciones principales:

- `analizar_fuerza_contrasena()` - Analiza y clasifica contraseñas según criterios de seguridad

Criterios de evaluación:

- Longitud mínima (8+ caracteres)
- Presencia de mayúsculas y minúsculas
- Inclusión de números
- Uso de símbolos especiales
- Detección de patrones prohibidos

Importaciones: Ninguna

Modulo 3: Generador de Contraseñas.

Responsabilidad: Crear contraseñas aleatorias seguras

Funciones principales:

- `generar_contrasena()` - Genera contrasenas personalizables con diferentes caracteres

Importaciones: `random`

▼ Modulo 4: Busqueda Recursiva.

Responsabilidad: Implementar búsqueda eficiente en registros

Funciones principales:

- `buscar_recursivo()` - Búsqueda recursiva por servicio o usuario

Importaciones: Ninguna

▼ Modulo 5: Validacion Recursiva de Integridad.

Responsabilidad: Verificar la consistencia de los datos almacenados

Funciones principales:

- `validar_entrada_recursiva()` - Valida campos requeridos recursivamente
- `revisar_integridad_recursiva()` - Revisa todos los registros de forma recursiva

Importaciones: Ninguna

▼ Modulo 6: Gestion de Archivos.

Responsabilidad: Persistencia de datos y registro de auditoría

Funciones principales:

- `guardar_contrasenas()` - Guarda registros en archivo TXT
- `cargar_contrasenas()` - Carga registros desde archivo
- `registrar_log()` - Registra acciones en log de auditoría
- `guardar_contrasena_maestra()` - Almacena contraseña maestra cifrada
- `cargar_contrasena_maestra()` - Recupera contraseña maestra

Importaciones: `os` , `datetime` **Dependencias:** Módulo 1 (cifrado_cesar, descifrado_cesar)

▼ Modulo 7: Interfaz y Menú Principal.

Responsabilidad: Interacción con el usuario y coordinación del sistema

Funciones principales:

- `main()` - Función principal del programa
- `verificar_contrasena_maestra()` - Autenticación de usuario
- `mostrar_menu_principal()` - Interfaz del menú
- `agregar_contrasena()` - Añadir nuevas contraseñas
- `consultar_contrasenas()` - Ver contraseñas guardadas
- `editar_contrasena()` - Modificar registros
- `eliminar_contrasena()` - Borrar registros
- `buscar_contrasenas()` - Búsqueda de contraseñas
- `generar_contrasena_interactivo()` - Generador con interfaz
- `revisar_integridad()` - Verificación del sistema
- `ver_log()` - Visualización de auditoría

Importaciones: `os` , `datetime` **Dependencias:** Todos los modulos anteriores (1-6)

▼ 3. ESTRUCTURAS DE DATOS Y ARCHIVOS.

Estructura de Registro de Contrasena.

Cada contrasena se almacena como un diccionario con los siguientes campos:

```
{
   'servicio': 'Gmail',          # Nombre del servicio
   'usuario': 'usuario@email.com', # Usuario o correo
   'contrasena': 'dI8fKw...',     # Contraseña cifrada
   'metodo_cifrado': 'cesar',     # Método usado (cesar/recursivo)
   'fecha': '2025-12-09 14:30:00' # Timestamp de creación
}
```

Archivos del Sistema.

1. contrasenas.txt.

- Formato: Texto plano con separador de pipe (|)
- Estructura: `servicio|usuario|contrasena_cifrada|metodo|fecha`
- Ejemplo:

```
Gmail|usuario@gmail.com|dI8fKwLqX|cesar|2025-12-09 14:30:00
Discord|player123|oWdFjK9pL|recursivo|2025-12-09 14:31:15
```

2. maestra.txt

- Formato: Texto plano
- Contenido: Contraseña maestra cifrada con César (desplazamiento 7)
- Propósito: Autenticación del usuario

3. audit_log.txt

- Formato: Texto plano con timestamps
- Estructura: `[YYYY-MM-DD HH:MM:SS] Descripción de acción`
- Ejemplo:

```
[2025-12-09 14:30:00] Acceso exitoso al sistema
[2025-12-09 14:30:45] Añadida contraseña para 'Gmail'
[2025-12-09 14:31:20] Consultada contraseña de 'Discord'
```

▼ 4. FUNCIONES RECURSIVAS IMPLEMENTADAS.

▼ 4.1 Inversión de Texto Recursiva.

```
def invertir_texto_recursivo(texto, indice=0):
    if indice >= len(texto):
        return ""
    return invertir_texto_recursivo(texto, indice + 1) + texto[indice]
```

Proposito: Invierte una cadena de texto carácter por carácter
Aplicación: Parte del cifrado recursivo

▼ 4.2 Búsqueda Recursiva.

```
def buscar_recursivo(lista, termino, indice=0, resultados=None):
    if indice >= len(lista):
        return resultados
    # Buscar coincidencias
    return buscar_recursivo(lista, termino, indice + 1, resultados)
```

Proposito: Busca coincidencias en una lista de registros
Aplicación: Búsqueda de contraseñas por servicio o usuario

▼ 4.3 Validación Recursiva de Campos.

```
def validar_entrada_recursiva(entrada, campos_requeridos, indice=0):
    if indice >= len(campos_requeridos):
        return True, []
    # Validar campo actual
    return validar_entrada_recursiva(entrada, campos_requeridos, indice
+ 1)
```

Proposito: Valida que un registro tenga todos los campos requeridos
Aplicación: Verificación de integridad de datos

▼ 4.4 Revisión de Integridad Recursiva.

```
def revisar_integridad_recursiva(registros, indice=0, errores_encontrad
os=None):
    if indice >= len(registros):
        return errores_encontrados
    # Validar registro actual
    return revisar_integridad_recursiva(registros, indice + 1, errores_enco
ntrados)
```

Proposito: Revisa todos los registros del sistema
Caso base: Cuando se procesan todos los registros
Aplicación: Auditoría completa del sistema

▼ 5. SEGURIDAD IMPLEMENTADA.

▼ Capas de Proteccion.

1. **Autenticacion:** Contrasena maestra con bloqueo tras 3 intentos
2. **Cifrado:** Dos métodos de cifrado disponibles
3. **Almacenamiento:** Contraseñas nunca se guardan en texto plano
4. **Auditoría:** Registro de todas las acciones del usuario
5. **Validación:** Análisis de fortaleza de contraseñas
6. **Integridad:** Verificación recursiva de datos

▼ Métodos de Cifrado.

Cifrado Cesar:

- Desplazamiento de 5 posiciones
- Rápido y simple
- Adecuado para uso básico

Cifrado Recursivo:

- Inversión recursiva del texto
- Aplicación de desplazamiento César
- Mayor complejidad

▼ 6. CASOS DE USO.

Caso 1: Primer Uso.

- Usuario inicia el programa por primera vez
- Sistema solicita crear contraseña maestra
- Usuario configura su contraseña de acceso
- Sistema queda listo para usar

Caso 2: Agregar Contrasena Debil.

- Usuario intenta guardar "12345"

- Sistema analiza: "Débil (1 punto)"
- Muestra problemas detectados
- Usuario decide si continuar o generar una mejor

Caso 3: Busqueda de Servicio.

- Usuario busca "mai"
- Sistema encuentra "Gmail" y "Hotmail"
- Muestra resultados con búsqueda recursiva

---

7. REPOSITORIO Y DOCUMENTACION.

Repositorio GIT:
https://github.com/CarloGennaro18/Safekey
