DOCUMENTO TÉCNICO

Sistema RedPluma – Infraestructura de Certificación de Integridad de Datos

Versión: 2.0
Fecha: 24/02/2026
Autor: RedPluma / Ing. Benjamín Abraham
Tipo de documento: Documento Técnico

Índice de contenido

1. Introducción

El presente documento describe la arquitectura técnica y los procesos internos del sistema **RedPluma**, basándose en su implementación real distribuida en múltiples servicios backend y componentes frontend.

El sistema está diseñado para la certificación de integridad de datos digitales mediante:

* funciones hash criptográficas
* estructuras de Merkle
* anclaje en blockchain Bitcoin vía OpenTimestamps

2. Arquitectura real del sistema

El sistema se organiza en tres bloques principales:

2.1. Frontend (WordPress)

Ubicación: /wordpress/

Componentes relevantes:

* plugins personalizados (ej: `redpluma-certificados`)
* generación de PDFs (dompdf)
* panel de usuario
* endpoints REST (`/wp-json/...`)

Funciones:

* interfaz de usuario
* carga de archivos / datasets
* visualización de resultados
* almacenamiento de metadata en base de datos

2.2. Backend de servicios (VPS)

Ubicación: /backend/

Dividido en múltiples servicios:

2.2.1. Servicio OTS (sellado)

Ubicación: /backend/ots_service/app.py

Funciones:

* recibe hashes
* genera pruebas OpenTimestamps
* interactúa con `ots` CLI
* devuelve archivos `.ots`

Dockerizado: /backend/ots_service/Dockerfile

2.2.2. Servicio de upgrade OTS

Ubicación: /backend/ots_upgrade_service/app_upgrade.py

Funciones:

* actualiza pruebas OTS
* verifica anclaje en Bitcoin
* consulta estado de confirmación

2.2.3. Servicio de email ingest

Ubicación: /backend/email_ingest/

Archivos clave:

* `imap_ingest.py`
* `raw_server.py`

Funciones:

* ingestión de emails vía IMAP
* procesamiento de emails crudos (RAW)
* preparación para certificación

2.3. Motor Chat Forensic (VPS)

Ubicación: /backend/chat_forensic/

Este es uno de los componentes más importantes del sistema.

Archivos clave

* `api_server.py` → API principal (Flask)
* `process_chat.py` → procesamiento principal
* `parser.py` → parsing de chats
* `merkle.py` → construcción de árbol
* `merkle_proofs.py` → generación de pruebas
* `verify_case.py` → verificación
* `upgrade_worker.py` → upgrade OTS

3. Flujo real de certificación

3.1. Ingreso de datos

Entrada vía:

* upload directo
* URL (ej: Google Drive)
* email ingest

API relevante:

/api/chat/create
/api/chat/create-from-url

3.2. Procesamiento asincrónico

El sistema utiliza: *subprocess.Popen* en *process_chat.py*

Esto permite:

* ejecución en background
* no bloqueo del API

3.3. Generación de hash

Implementación real:

* hash del dataset completo
* hash individual de cada archivo

Algoritmo: *doble SHA-256*

Esto se aplica en:

* dataset.zip
* archivos internos

3.4. Parsing de datos

Archivo: *parser.py*

Funciones:

* extracción de mensajes
* identificación de participantes
* conteo de elementos multimedia

Salida: estructura que luego se guarda en `report.json`

3.5. Construcción del Merkle Tree

Archivo: *merkle.py*

Características reales:

* orden determinístico por nombre de archivo
* duplicación del último nodo si cantidad impar
* concatenación de hashes
* doble SHA-256 en cada nivel

Salida: *merkle_root*

3.6. Generación de pruebas Merkle

Archivo: *merkle_proofs.py*

Genera:

* proof por archivo
* estructura de validación

Salida: `merkle_proofs.json`

3.7. Sellado OTS

Proceso:

1. se genera: *merkle_root.txt*
2. se ejecuta: *ots stamp*
3. se obtiene: *merkle_root.txt.ots*

3.8. Upgrade y verificación

Archivos: *upgrade_worker.py , verify_case.py*

Proceso:

* ejecución de: ots verify –no-bitcoin

* parsing de salida

detección de:

* block height
* estado de anclaje

3.9. Almacenamiento de caso

Ubicación: /opt/redpluma/chat_forensic/cases/<case_id>/

Archivos generados:

* `dataset.zip`
* `report.json`
* `merkle_root.txt`
* `merkle_root.txt.ots`
* `merkle_proofs.json`

3.10. Generación de reportes

Archivo: *report.json*

Contiene:

* case_id
* hash del dataset
* tamaño
* participantes
* métricas (mensajes, multimedia)
* fechas

4. API del sistema

4.1. Endpoints principales

Desde `api_server.py`:

* `/api/chat/create`
* `/api/chat/create-from-url`
* `/api/chat/list`
* `/api/chat/status/<case_id>`
* `/api/chat/<case_id>/report-json`
* `/api/chat/verify`

4.2. Descargas

* report
* ots
* merkle proofs
* dataset

5. Verificación técnica real

5.1. Verificación de dataset

* recalcular SHA-256 doble del zip
* comparar con `zip_sha256`

5.2. Verificación Merkle

* usar `merkle_proofs.json`
* reconstruir root

5.3. Verificación OTS

Comando real:

ots verify merkle_root.txt.ots

Resultado esperado:

* referencia a bloque Bitcoin

6. Seguridad y propiedades

6.1. Hashing

* SHA-256 doble
* resistencia a colisiones
* sensibilidad total a cambios

6.2. Determinismo

* ordenamiento de archivos
* reglas claras de construcción

 evita inconsistencias

6.3. No almacenamiento de contenido persistente (en capa blockchain)

* solo hashes se anclan
* datos permanecen locales

6.4. Procesamiento aislado

* ejecución por caso
* logs independientes

7. Limitaciones reales del sistema

* depende de calidad del dataset
* parsing de chats puede variar según formato
* requiere integridad del archivo inicial

8. Escalabilidad

* arquitectura desacoplada
* servicios independientes
* dockerización parcial (OTS)

9. Conclusión técnica

RedPluma implementa una arquitectura distribuida que combina:

* procesamiento asincrónico
* hashing criptográfico robusto
* estructuras Merkle determinísticas
* integración con OpenTimestamps

Esto permite generar pruebas de integridad verificables, independientes y escalables.