Documentazione Web File Transfer

[1 Introduzione 3](#_Toc217048587)

[1.1 Informazioni sul progetto 3](#_Toc217048588)

[1.2 Abstract 3](#_Toc217048589)

[1.3 Scopo 3](#_Toc217048590)

[2 Analisi 4](#_Toc217048591)

[2.1 Analisi del dominio 4](#_Toc217048592)

[2.2 Analisi e specifica dei requisiti 4](#_Toc217048593)

[2.2.1 Spiegazione elementi tabella dei requisiti: 6](#_Toc217048594)

[2.3 Use case 6](#_Toc217048595)

[2.4 Pianificazione 1](#_Toc217048596)

[2.5 Analisi dei mezzi 1](#_Toc217048597)

[2.5.1 Software 1](#_Toc217048598)

[2.5.2 Hardware 1](#_Toc217048599)

[3 Progettazione 2](#_Toc217048600)

[3.1 Design dell’architettura del sistema 2](#_Toc217048601)

[3.1.1 Architettura Generale 2](#_Toc217048602)

[3.1.2 Schema di rete 2](#_Toc217048603)

[3.2 Design dei dati e database 3](#_Toc217048604)

[3.3 Design delle interfacce 4](#_Toc217048605)

[3.4 Design procedurale 5](#_Toc217048606)

[3.4.1 Diagramma di flusso 5](#_Toc217048607)

[4 Implementazione 6](#_Toc217048608)

[4.1 Architettura sito web 6](#_Toc217048609)

[4.2 Generazione Token e Derivazione Chiave 6](#_Toc217048610)

[4.3 Crittografia Client-Side 7](#_Toc217048611)

[4.4 Upload File con crittografia E2E 9](#_Toc217048612)

[4.5 Hash Token sul server 11](#_Toc217048613)

[4.6 Download e Decrittografia Client-Side 13](#_Toc217048614)

[4.7 Validazione Download e gestione limiti 15](#_Toc217048615)

[4.8 Rate limiting 16](#_Toc217048616)

[4.9 Implementazione Postfix e Dovecot 17](#_Toc217048617)

[5 Test 18](#_Toc217048618)

[5.1 Protocollo di test 18](#_Toc217048619)

[5.2 Risultati test 22](#_Toc217048620)

[5.3 Mancanze/limitazioni conosciute 30](#_Toc217048621)

[6 Consuntivo 1](#_Toc217048622)

[7 Conclusioni 1](#_Toc217048623)

[7.1 Sviluppi futuri 1](#_Toc217048624)

[7.2 Considerazioni personali 1](#_Toc217048625)

[8 Glossario 2](#_Toc217048626)

[9 Indice delle figure 3](#_Toc217048627)

[10 Bibliografia 4](#_Toc217048628)

[10.1 Sitografia 4](#_Toc217048629)

[11 Allegati 5](#_Toc217048630)

# Introduzione

## Informazioni sul progetto

* Titolo: Web File Transfer con crittografia end-to-end
* Scuola/sezione: Scuola Arti e Mestieri Informatica
* Allievo: Armir Cetaj
* Docente: Mariano Fasano
* Periodo: 05.09.2025 – 19.12.2025

## Abstract

L’aumento del numero di servizi cloud per la condivisione di file ha portato con sé importanti rischi legati alla privacy e alla sicurezza. Molte di queste piattaforme, infatti, possono accedere ai contenuti caricati dagli utenti, compromettendo la riservatezza delle informazioni.

Questo progetto propone lo sviluppo di una piattaforma web per il trasferimento sicuro di file con crittografia end-to-end. Il file viene cifrato nel browser del mittente prima dell’upload, e decifrato solo nel browser del destinatario al momento del download. In questo modo il server funge solo da contenitore di file cifrati senza possibilità di accedere ai dati in chiaro.

Sono incluse funzionalità aggiuntive come link effimeri, limite di download e notifica opzionale al mittente in caso di avvenuto scaricamento. Il progetto mira a dimostrare come sia possibile realizzare una soluzione semplice e user-friendly, ma al tempo stesso sicura, per la protezione dei dati condivisi online.

## Scopo

Lo scopo del progetto è sviluppare un’applicazione web che garantisca:

* la condivisione sicura di file senza esposizione dei dati in chiaro,
* la generazione di link temporanei per il download,
* la possibilità di impostare scadenze o limiti di accesso,
* un’opzione di notifica per il mittente al momento del primo download,
* un’interfaccia semplice e intuitiva utilizzabile da utenti non esperti.

# Analisi

## Analisi del dominio

Il contesto è quello della condivisione di file tra utenti via internet. Attualmente il problema viene affrontato da servizi come WeTransfer e SwissTransfer, che offrono praticità ma non garantiscono totale riservatezza, poiché i file vengono salvati in chiaro sui loro server.

L’obiettivo del sistema è offrire un’alternativa privacy-first, in cui:

* Il server non può leggere né modificare i contenuti.
* Gli utenti (mittente e destinatario) sono gli unici a possedere le chiavi di decrittazione.
* I link generati hanno validità limitata per ridurre i rischi di accesso non autorizzato.

Gli utenti principali sono:

* Mittente, che carica i file e decide eventuali opzioni (scadenza, limite download, notifica).
* Destinatario, che riceve il link e decritta i file.

## Analisi e specifica dei requisiti

|  |  |
| --- | --- |
| **ID: REQ-001** | |
| **Nome** | Upload e cifratura lato client |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** | Richiede l’utilizzo di librerie come libsodium |
| **Sotto requisiti** | |
| **001** | L’utente deve poter selezionare uno o più file dal browser |
| **002** | I metadati devono essere anch’essi cifrati (nome file, dimensione) |

|  |  |
| --- | --- |
| **ID: REQ-002** | |
| **Nome** | Generazione link univoco |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** | Il token deve essere impossibile da indovinare (es. 256-bit random) |
| **Sotto requisiti** | |
| **001** | Creazione URL di download contenente il token |
| **002** | Il token deve avere validità temporale |
| **003** | Il token deve essere a un massimo di download configurabile |

|  |  |
| --- | --- |
| **ID: REQ-003** | |
| **Nome** | Download e decifratura del file |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** | La chiave di cifratura deve essere ricavata dal link |
| **Sotto requisiti** | |
| **001** | Il file deve essere decifrato automaticamente nel browser |
| **002** | Il file deve essere scaricato in chiaro solo localmente |

|  |  |
| --- | --- |
| **ID: REQ-004** | |
| **Nome** | Opzioni di scadenza e limiti |
| **Priorità** | 2 |
| **Versione** | 1.0 |
| **Note** | Il mittente decide durata del link o numero massimo di download |
| **Sotto requisiti** | |
| **001** | Possibilità di impostare un ora/data di scadenza |
| **002** | Possibilità di impostare |
| **003** | Eliminazione automatica dei file scaduti |

|  |  |
| --- | --- |
| **ID: REQ-005** | |
| **Nome** | Notifica opzionale al mittente |
| **Priorità** | 2 |
| **Versione** | 1.0 |
| **Note** | L’email è facoltativa e serve solo per avvisi di download |
| **Sotto requisiti** | |
| **001** | Un mail server |

### Spiegazione elementi tabella dei requisiti:

**ID**: identificativo univoco del requisito

**Nome**: breve descrizione del requisito

**Priorità**: indica l’importanza di un requisito nell’insieme del progetto, definita assieme al committente. Ad esempio, poter disporre di report con colonne di colori diversi ha priorità minore rispetto al fatto di avere un database con gli elementi al suo interno. Solitamente si definiscono al massimo di 2-3 livelli di priorità.

**Versione**: indica la versione del requisito. Ogni modifica del requisito avrà una versione aggiornata.

Sulla documentazione apparirà solamente l’ultima versione, mentre le vecchie dovranno essere inserite nei diari.

**Note**: eventuali osservazioni importanti o riferimenti ad altri requisiti.

**Sotto requisiti**: elementi che compongono il requisito.

## Use case

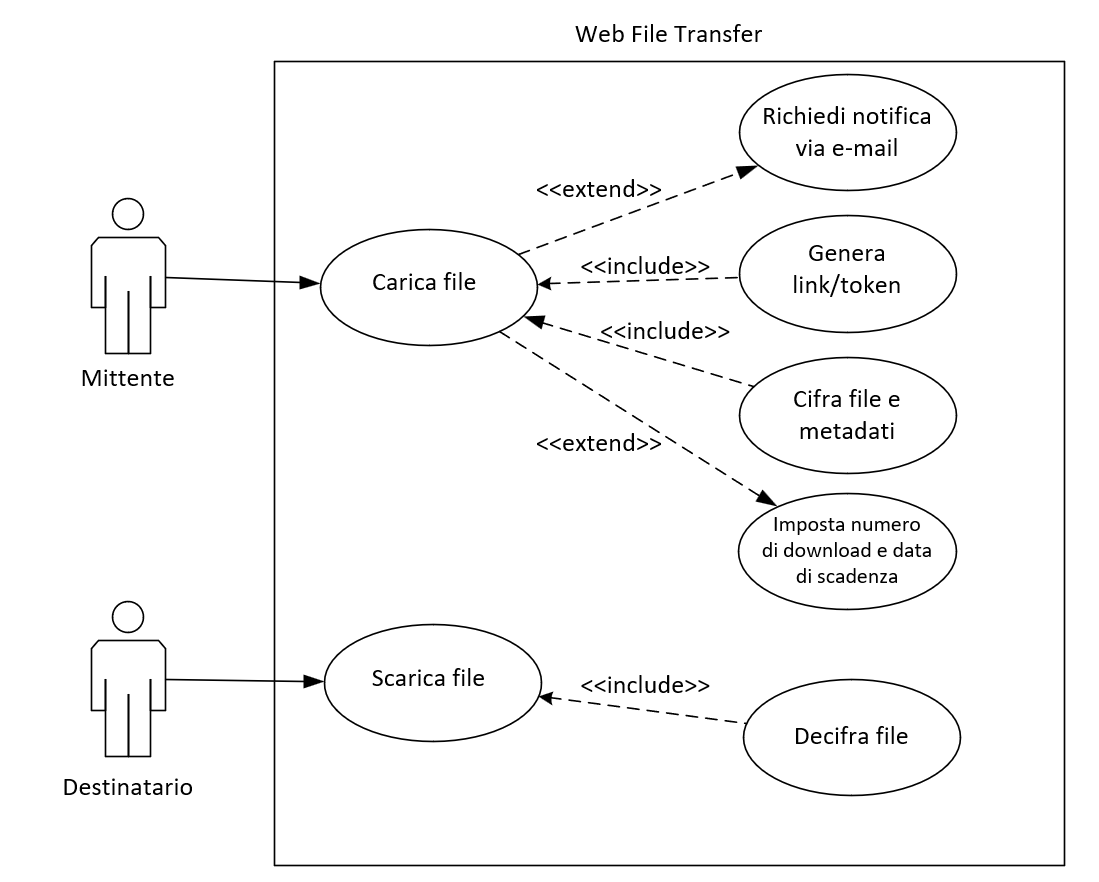


Figura 1: Diagramma Use case

## Pianificazione

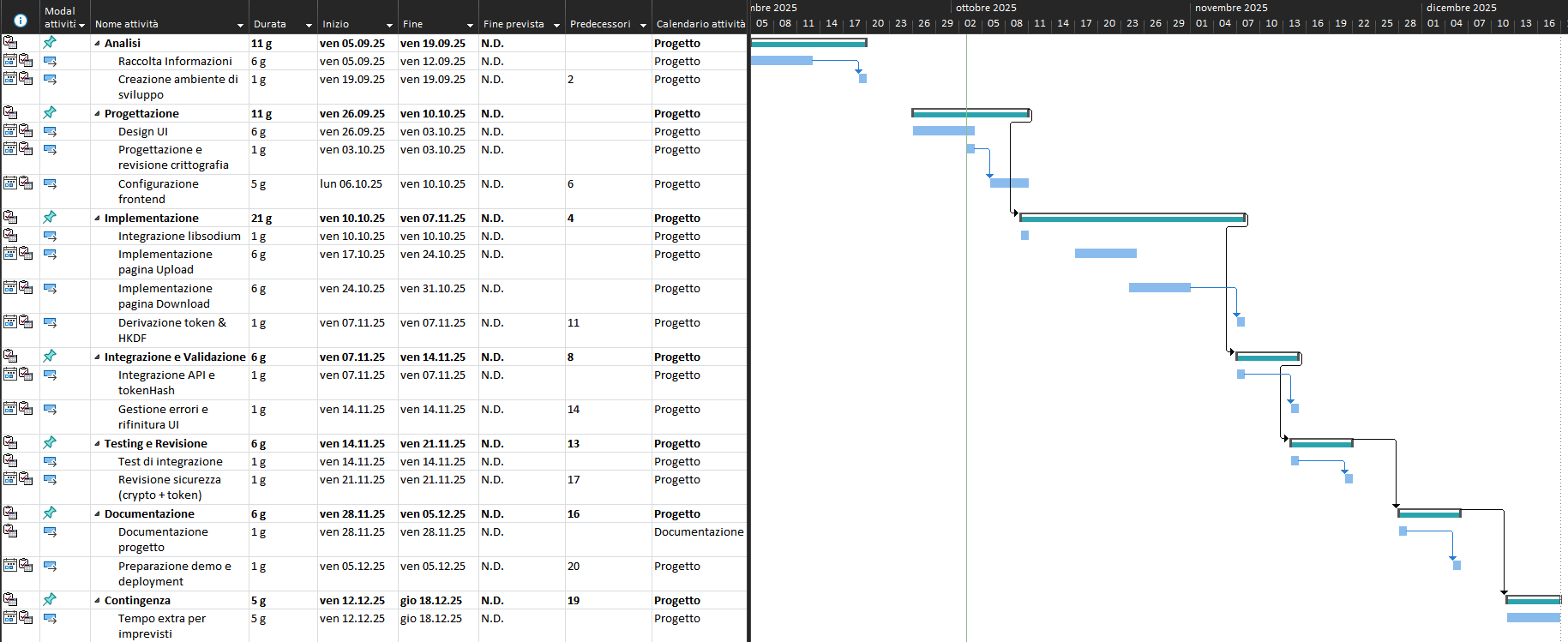


Figura 2: Diagramma di Gantt

## Analisi dei mezzi

### Software

**Linguaggi e runtime**

* JavaScript (ES6+): linguaggio principale, usato sia lato client che lato server.
* CSS 5.1: linguaggio utilizzato per definire lo stile del sito web.
* Node.js v22 LTS: runtime per eseguire codice JavaScript lato server.

**Framework e librerie backend**

* Express.js v5.1: framework leggero per la gestione delle API REST e del routing.
* Multer v2.2: middleware per la gestione degli upload di file (multipart/form-data).
* libsodium v1.0.20: libreria crittografica per la cifratura e decifratura sicura dei file lato server.
* pg (node-postgres) v8.16: driver ufficiale Node.js per interfacciarsi con PostgreSQL.

**Database**

* PostgreSQL v18: database relazionale utilizzato per memorizzare i metadati dei file condivisi: token temporanei, date di scadenza, indirizzi email opzionali del mittente (per notifiche).

**Mail server**

* Postfix v3.10: gestisce direttamente l’invio delle email di notifica.
  + Configurato con Dovecot per servizio IMAP.
  + Node.js non usa librerie intermedie: le notifiche vengono passate al mail server tramite SMTP locale.

**Strumenti di sviluppo**

* Visual Studio Code v1.93: IDE principale.
* Vim: IDE secondario usato sulla macchina server.
* Git v2.51 + GitHub: Version control del codice.
* curl / wget: test basilari degli endpoint senza necessità di strumenti esterni.

**Browser (client)**

* Compatibilità con:
  + Google Chrome v141+
  + Mozilla Firefox v143+
  + Microsoft Edge v138+

### Hardware

* Per accedere al sito si ha bisogno di una piattaforma che garantisca una connessione stabile all’internet.
* Per replicare il server si ha bisogno di una macchina che consente la virtualizzazione (creare macchine virtuali Linux)

# Progettazione

## Design dell’architettura del sistema

### Architettura Generale

Il sistema è organizzato secondo un’architettura classica client-server:

Client:

* Cifra i file e i metadati localmente.
* Genera token e deriva la chiave per la cifratura.
* Effettua upload dei blob cifrati e avvia decrittazione al download.

Server (Node.js + Express):

* Espone API HTTPS per upload, download e stato.
* Memorizza solo blob cifrati e metadati cifrati; salva token\_hash e parametri (scadenza, limiti).
* Invoca il mail server locale per notifiche (se richiesto).

Database (PostgreSQL):

* Conserva record con token\_hash, path del blob, limiti, timestamp.

Mail server (Postfix/Dovecot):

* Gestisce la consegna delle email di notifica al mittente.

### Schema di rete

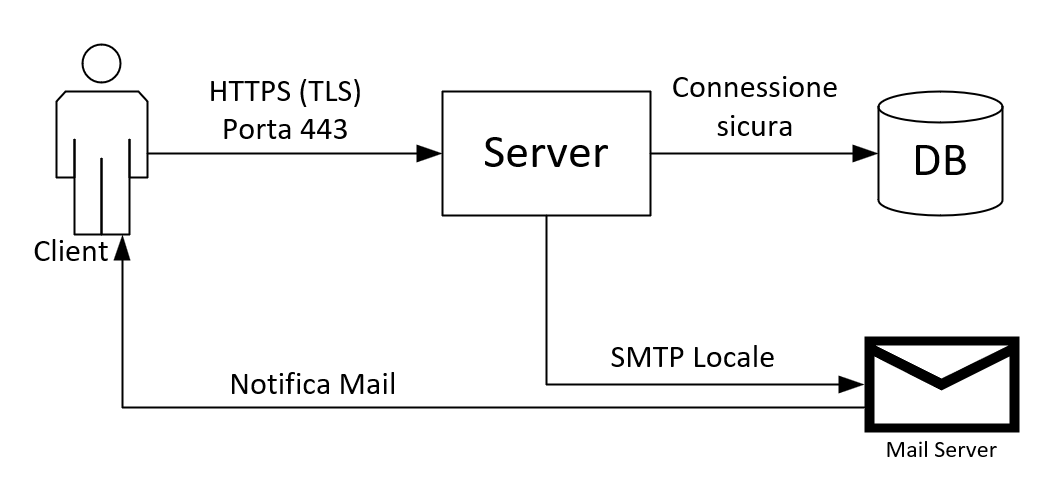


Figura 3: Schema di rete

Il client si connette al server stabilendo una connessione HTTPS sulla porta 443, il server comunica con il DB e il Mail Server, il Mail Server spedisce un’email al client.

## Design dei dati e database

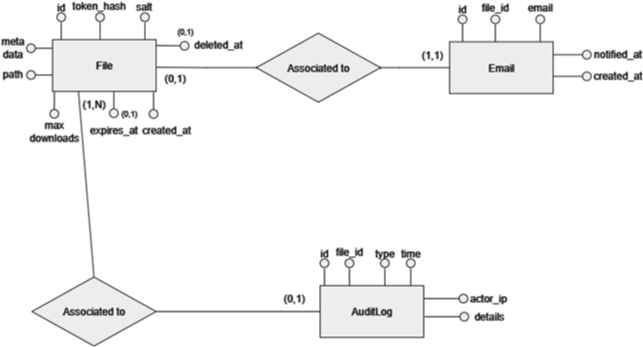


Figura 4: Schema ER

**files**

* id SERIAL PRIMARY KEY
* token\_hash BYTEA NOT NULL UNIQUE
* salt BYTEA NOT NULL
* metadata BYTEA NOT NULL
* path TEXT NOT NULL
* ciphertext\_length BIGINT NOT NULL
* max\_downloads INT NOT NULL DEFAULT 1
* download\_count INT NOT NULL DEFAULT 0
* expires\_at TIMESTAMP WITH TIME ZONE
* created\_at TIMESTAMP WITH TIME ZONE NOT NULL DEFAULT NOW()
* deleted\_at TIMESTAMP WITH TIME ZONE

**notifications**

* id SERIAL PRIMARY KEY
* file\_id INTEGER NOT NULL REFERENCES files(id) ON DELETE CASCADE
* email BYTEA
* notified\_at TIMESTAMP WITH TIME ZONE NULL
* created\_at TIMESTAMP WITH TIME ZONE NOT NULL DEFAULT NOW()

**audit\_logs**

* id SERIAL PRIMARY KEY
* file\_id INTEGER NULL REFERENCES files(id) ON DELETE SET NULL
* type TEXT NOT NULL
* time TIMESTAMP WITH TIME ZONE NOT NULL DEFAULT NOW()
* actor\_ip INET
* details TEXT

## Design delle interfacce

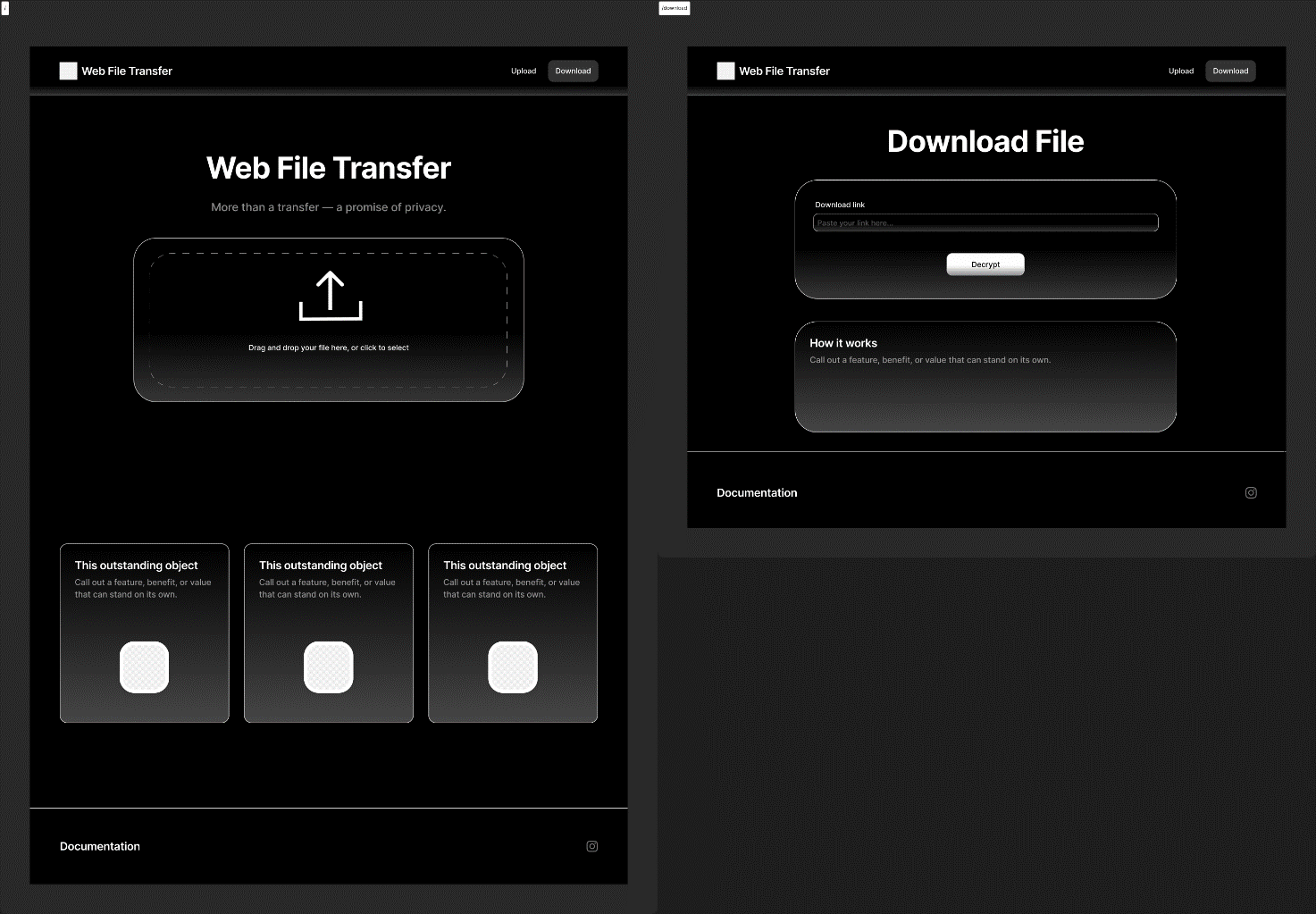


Figura 5: Mockup sito web

Pagina di Upload

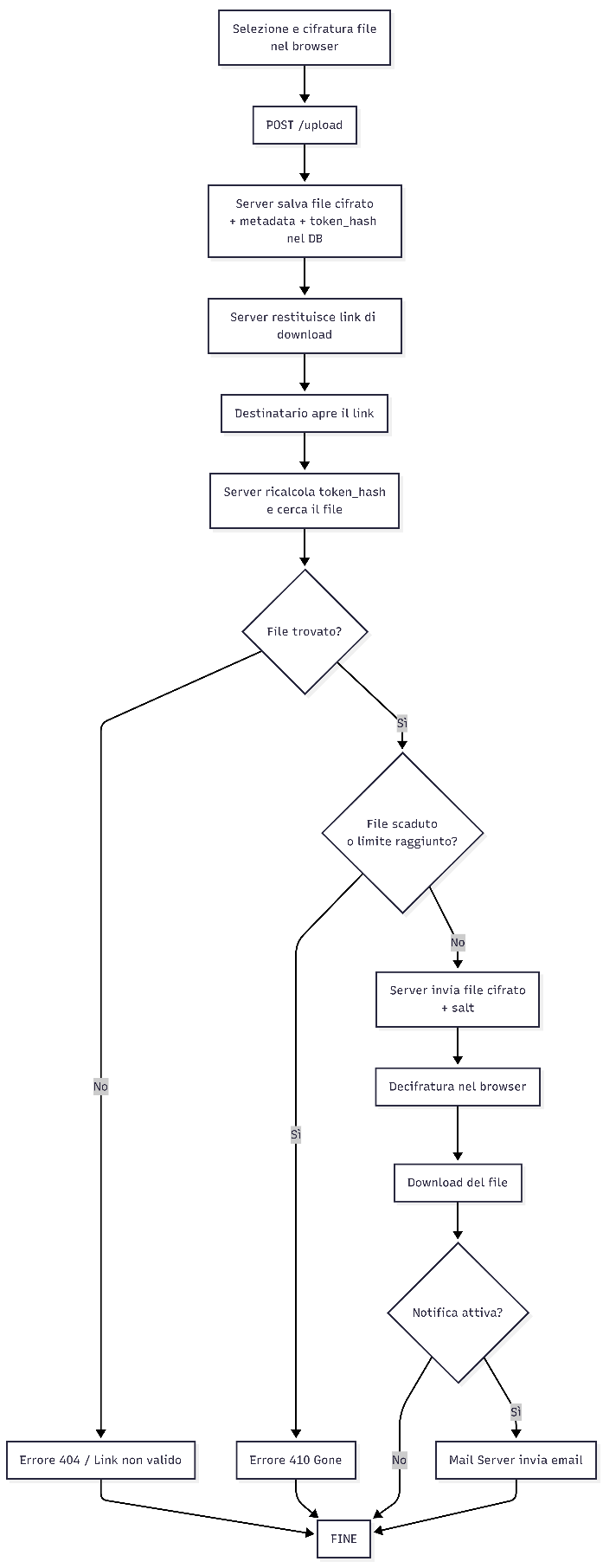
* Titolo e sottotitolo: comunicano la funzione principale del servizio e l’enfasi sulla privacy.
* Area drag & drop: permette all’utente di trascinare o selezionare un file da caricare.
* Pulsanti di navigazione: accesso rapido alla sezione di download e alle altre funzionalità.
* Sezione informativa: tre riquadri sottostanti illustrano caratteristiche e vantaggi principali del servizio (es. sicurezza, velocità, anonimato).
* Footer: link a documentazione o informazioni aggiuntive.

Pagina di Download

* Campo link di download: l’utente incolla l’URL ricevuto.
* Pulsante “Decrypt”: avvia il processo di recupero e decifratura del file, tutto lato client per garantire la privacy.
* Sezione “How it works”: descrizione sintetica del funzionamento per aiutare l’utente a comprendere il processo.
* Footer: coerente con la pagina di upload.

## Design procedurale

### Diagramma di flusso



# Implementazione

## Architettura sito web

Il progetto è organizzato in una struttura client-server con separazione tra frontend e backend:

**Backend (Node.js/Express):**

server/app.js - Server principale Express con gestione route, rate limiting e logica di upload/download

server/db/connection.js - Configurazione connessione PostgreSQL

server/db/models.js - Modelli per le varie query FileModel, NotificationModel e AuditLogModel

**Frontend (JS):**

client/index.html - Pagina principale per l'upload dei file

client/download.html - Pagina per il download e decrittografia dei file

client/js/crypto-client.js - Classe CryptoClient per operazioni crittografiche (generazione token, derivazione chiavi, crittografia/decrittografia)

client/js/uploader.js - Gestione upload file con crittografia client-sided

client/js/downloader.js - Gestione download e decrittografia file client-sided

client/js/main.js - Funzioni di utility e gestione UI

client/js/animation.js – Animazioni UI con Motion.js

**Database:**

sql/schema.sql - Schema database PostgreSQL con tabelle files, notifications e audit\_logs

**Script:**

scripts/cleanup.js - Script per rimozione automatica dei file scaduti

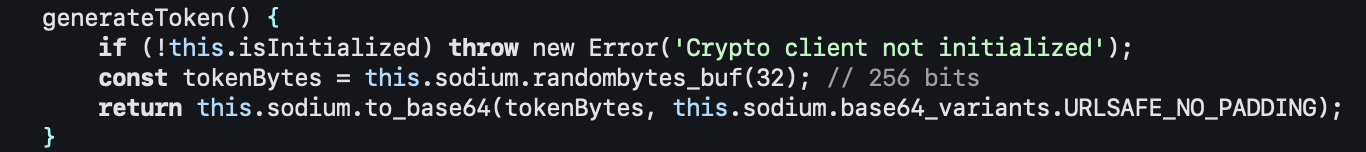
**Storage:**

storage/ - Directory contenente i file crittografati salvati con estensione .enc (encrypted)

## Generazione Token e Derivazione Chiave

La sicurezza del sistema si basa sulla generazione di token crittograficamente sicuri e sulla derivazione di chiavi di crittografia senza che il server possa mai accedere al token originale, essendo che il token viene generato sul client.

crypto-client.js – Funzione generazione token



Il token è l'unico segreto necessario per decrittografare il file. Viene generato lato client e non viene mai inviato al server in forma leggibile (solo durante l'upload per creare l'hash).

Genera un token di 256 bit (32 byte) utilizzando il generatore di numeri casuali di libsodium (randombytes\_buf). Il token viene codificato in base64 URL-safe senza padding per essere utilizzabile direttamente nell’URL.

La scelta di 256 bit garantisce 2^256 possibili combinazioni, rendendo il brute-forcing quasi impossibile. Base64 URL-safe (- e \_ invece di + e /) evita problemi di encoding nell’URL.

crypto-client.js – Funzione derivazione chiave

Immagine che contiene testo, schermata, Carattere

Il contenuto generato dall'IA potrebbe non essere corretto.

Fa derivare una chiave di crittografia da 32 byte (256 bit) dal token utilizzando la Key Derivation Function (KDF) di libsodium, poi applica un'operazione XOR bitwise con un salt casuale per aumentare la sicurezza e garantire che ogni file abbia una chiave unica anche con lo stesso token.

**Parametri**

* token: token base64 URL-safe generato da GenerateToken()
* salt: buffer casuale di 32 byte
* context: stringa di contesto di 8 caratteri per il KDF

L'operazione XOR con il salt (unico per file) aggiunge un layer di sicurezza: anche se un attaccante conosce il token, senza il salt specifico non può ricostruire la chiave. Il salt viene memorizzato sul server ma è inutile senza il token.

## Crittografia Client-Side

Tutti i file vengono crittografati nel browser prima dell'invio al server, garantendo che il server non veda mai i dati in chiaro.

crypto-client.js – Funzione crittografia file

Immagine che contiene testo, schermata, Carattere

Il contenuto generato dall'IA potrebbe non essere corretto.

Crittografa i dati del file utilizzando ChaCha20-Poly1305, un cifrario AEAD (Authenticated Encryption with Associated Data) che fornisce sia confidenzialità che autenticazione. Genera un nonce casuale di 12 byte (96 bit) e lo prepone al ciphertext per permettere la decrittografia senza informazioni aggiuntive.

**Parametri**

* data: Uint8Array contenente i dati del file in chiaro
* key: chiave di crittografia da 32 byte derivata dal token

ChaCha20-Poly1305 è stato scelto perché è veloce su CPU moderne, resistente a timing attacks, e standardizzato (RFC 8439).

Il nonce di 12 byte è sufficiente per garantire unicità: con 2^96 possibili valori, anche con miliardi di file crittografati, la probabilità di collisione è trascurabile.

I parametri null indicano che non ci sono "associated data" (AAD) - Poly1305 autentica solo il ciphertext. Il nonce viene preposto al ciphertext (nonce || ciphertext) per semplicità: il client può estrarre i primi 12 byte durante la decrittografia senza dover gestire header separati.

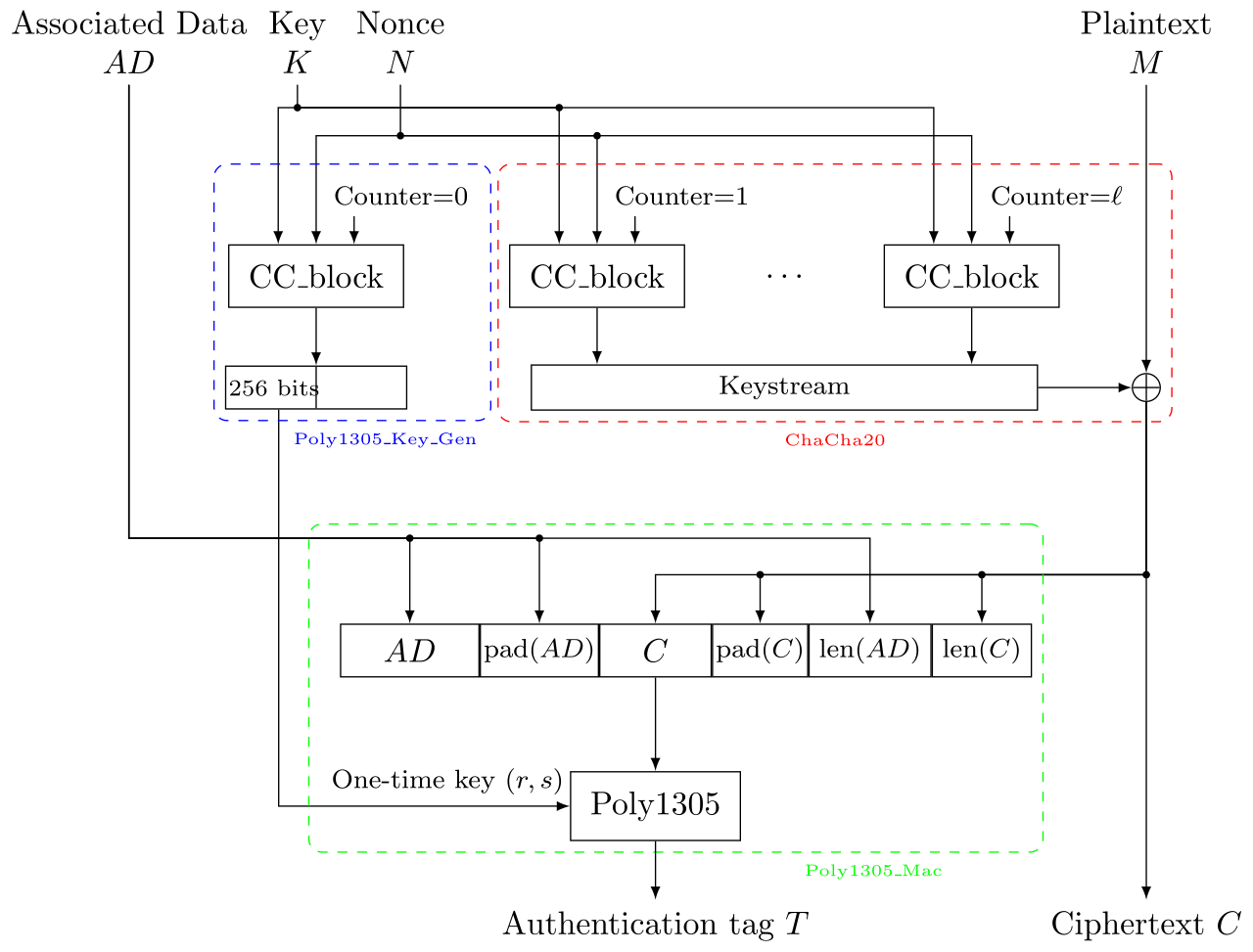
Ho considerato di utilizzare AES-GCM, ma ho scartato l’opzione perché ChaCha20 è più veloce su CPU senza accelerazione hardware AES e offre migliore resistenza ad attacchi side-channel.

Figura 6: Procedura di crittografia ChaCha20Poly1305

crypto-client.js – Funzione crittografia metadati file

Immagine che contiene testo, schermata, Carattere

Il contenuto generato dall'IA potrebbe non essere corretto.

Lo scopo della funzione è di serializzare i metadati del file (nome, tipo MIME, dimensione, data modifica) in JSON, li codifica in UTF-8, e li crittografa utilizzando la stessa funzione encryptFile con la stessa chiave del file principale.

**Parametri**

* metadata: oggetto JS con proprietà filename, mimeType, size, lastModified
* key: stessa chiave utilizzata per crittografare il file

I metadati vengono crittografati separatamente per permettere al client di recuperare il nome file originale durante il download senza dover decrittografare l'intero file. L'uso della stessa chiave semplifica la gestione: un solo segreto (il token) permette di decrittografare sia file che metadati. Ho scelto di utilizzare JSON per facilità di parsing e compatibilità.

## Upload File con crittografia E2E

Il processo di upload garantisce che tutti i dati siano crittografati prima di lasciare il browser del client, implementando un'architettura zero-knowledge.

uploader.js – Funzione upload del file

Immagine che contiene testo, schermata

Il contenuto generato dall'IA potrebbe non essere corretto.

La funzione orchestra l'intero processo di upload: inizializza il client crittografico, genera token e salt casuali, deriva la chiave di crittografia, legge il file come ArrayBuffer, crittografa file e metadati, serializza tutto in FormData con encoding base64, e invia al server.

Il token viene salvato in sessionStorage per riferimento locale (non necessario per il download).

**Parametri**

* file: oggetto File del browser (da <input type="file"> o drag-and-drop)
* options: oggetto opzionale con maxDownloads (intero), expiresAt (Date), senderEmail (stringa)

Il file viene letto come ArrayBuffer per accedere ai byte raw senza interpretazione.

La conversione in Uint8Array è necessaria per l'API di libsodium, tutti i dati binari vengono codificati in base64 per la trasmissione HTTP (FormData accetta stringhe).

Il token viene inviato al server solo durante l'upload per permettere la creazione dell'hash; successivamente il server non ha mai bisogno del token originale.

Ho scelto di usare FormData invece di JSON per supportare potenzialmente upload multipart in futuro e per semplicità. Base64 aumenta la dimensione del payload di ~33%, ma è accettabile per file fino a 100MB.

Ho considerato di usare Streaming encryption per file grandi, ma l’ho scartata per complessità e perché 100MB è gestibile in memoria nei browser moderni.

app.js – Gestione upload sul server

Immagine che contiene testo, schermata, Carattere

Il contenuto generato dall'IA potrebbe non essere corretto.

La funzione riceve i dati crittografati dal client, valida i campi obbligatori, decodifica i buffer base64, genera un UUID v4 per il nome file, salva il file crittografato su disco, calcola l'hash HMAC-SHA256 del token, memorizza i metadati nel database, e restituisce l'URL di download contenente il token originale.

Il file viene salvato con estensione .enc per indicare che è crittografato.

L'UUID v4 garantisce unicità e previene collisioni, il token viene hashato immediatamente dopo la validazione e mai memorizzato in chiaro.

La validazione token.length < 32 è un controllo minimo (un token base64 di 32 byte è ~43 caratteri).

Il salt e i metadati vengono memorizzati come BYTEA nel database per preservare i dati binari.

Ho scelto di usare un UUID invece di hash del token come nome file per evitare problemi se il token hash cambia (es. cambio chiave HMAC).

Il token viene incluso nell'URL di download per semplicità: l'utente può copiare l'URL completo senza dover gestire token separati.

## Hash Token sul server

Il server memorizza solo l'hash del token per sicurezza, rendendo impossibile recuperare il token originale anche in caso di compromissione del database.

app.js – Gestione upload sul server

Immagine che contiene testo, schermata, Carattere

Il contenuto generato dall'IA potrebbe non essere corretto.

Decodifica il token base64 URL-safe in buffer binario (normalizzando i caratteri -/\_ in +// e aggiungendo padding se necessario), poi calcola l'hash HMAC-SHA256 utilizzando una chiave segreta memorizzata nella variabile d'ambiente TOKEN\_HASH\_SECRET. L'output è un digest di 32 byte (256 bit).

**Parametri**

* token: token originale in base64 URL-safe (stringa)

HMAC (Hash-based Message Authentication Code) viene utilizzato invece di un semplice hash (es. SHA-256) per prevenire attacchi di estensione e garantire che solo chi possiede la chiave segreta possa verificare l'hash.

La normalizzazione base64 è necessaria perché il client usa variante URL-safe (-/\_) mentre Buffer.from() si aspetta standard base64 (+//).

Il padding viene aggiunto dinamicamente perché base64 URL-safe spesso omette il padding =.

Ho scelto di usare HMAC invece di un hash semplice perché anche se un attaccante ottiene accesso al database, senza la chiave TOKEN\_HASH\_SECRET non può creare hash validi per token arbitrari. La chiave deve essere mantenuta segreta e cambiata periodicamente.

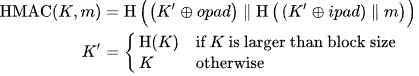
Dove:

Figura 7: Definizione di HMAC (RFC 2104)

**H**: funzione hash crittografica (nel nostro caso SHA256)

*m*: messaggio da autenticare (il token utente dopo decodifica Base64)

*K*: chiave segreta (Variabile d'ambiente TOKEN\_HASH\_SECRET)

*K'*: chiave derivata da K, portata alla dimensione del blocco

*opad*: padding esterno (0x5c ripetuto)

*ipad*: padding interno (0x36 ripetuto)

app.js – Validazione token durante il download

Immagine che contiene testo, schermata, Carattere

Il contenuto generato dall'IA potrebbe non essere corretto.

Estrae il token dalla URL path parameter, valida la lunghezza minima, calcola l'hash HMAC del token, e cerca il file corrispondente nel database utilizzando l'hash come chiave di ricerca.

**Parametri**

* token: token dato dall’URL del download (path parameter)

La validazione token.length < 32 è un controllo minimo per evitare query inutili al database.

Il lookup nel database utilizza un indice unico su token\_hash per performance ottimali, se il file non viene trovato, viene restituito errore 404 per non rivelare se un token è valido o meno (security through obscurity secondaria).

Ho scelto di usare Path parameter invece di query parameter per permettere validazione URL più semplice e per evitare problemi con caratteri speciali nelle URL.

L'hash viene ricalcolato ad ogni richiesta invece di essere memorizzato in cache per garantire che cambiamenti alla chiave HMAC siano immediatamente effettivi.

## Download e Decrittografia Client-Side

l processo di download garantisce che la decrittografia avvenga esclusivamente nel browser del client,

mantenendo la privacy E2E.

downloader.js – Download e decrittografia file

Immagine che contiene testo, schermata, Carattere, design

Il contenuto generato dall'IA potrebbe non essere corretto.

La funzione orchestra il download completo: verifica lo status del file (senza consumare un download), recupera e decrittografa i metadati per ottenere il nome file originale, scarica il file crittografato, estrae il salt dall'header HTTP, deriva la chiave di decrittografia, decrittografa il file nel browser, crea un blob temporaneo, e avvia il download del browser con il nome file originale.

**Parametri**

* url: URL complete dei download contente il token
* token: token estratto dall’URL

Il processo è diviso in fasi per efficienza e UX. Prima si verifica lo status tramite endpoint /status che non incrementa il contatore download, permettendo all'utente di vedere se il file è disponibile prima di consumare un download, poi si recuperano i metadati crittografati tramite endpoint /metadata separato, che permette di ottenere il nome file senza scaricare l'intero file.

Il salt viene inviato nell'header HTTP X-File-Salt invece che nel body per permettere al client di derivare la chiave prima di processare il file. Il file viene scaricato come ArrayBuffer per accesso ai byte raw.

La decrittografia avviene completamente in memoria. URL.createObjectURL crea un URL blob temporaneo che punta ai dati decrittografati in memoria.

Il download viene triggerato programmaticamente creando un elemento <a> temporaneo. URL.revokeObjectURL libera la memoria del blob dopo il download.

crypto-client.js – Decrittografia file

Immagine che contiene testo, schermata, Carattere

Il contenuto generato dall'IA potrebbe non essere corretto.

Estrae il nonce dai primi 12 byte del payload crittografato, estrae il ciphertext dal resto, e decrittografa utilizzando ChaCha20-Poly1305. Poly1305 verifica automaticamente l'autenticazione durante la decrittografia: se il ciphertext è stato modificato, la funzione lancia un'eccezione.

**Parametri**

* encryptedData: Uint8Array contenente nonce (12 byte) + ciphertext
* key: chiave di decrittografia di 32 byte derivata dal token e salt

Il metodo slice(0, nonceLength) estrae i primi 12 byte (nonce), slice(nonceLength) estrae il resto (ciphertext). I parametri null indicano che non ci sono associated data da verificare.

Se l'autenticazione Poly1305 fallisce (ciphertext modificato o chiave errata), libsodium lancia un'eccezione automaticamente.

## Validazione Download e gestione limiti

Il server valida ogni richiesta di download controllando scadenze, limiti di download e stato del file prima di permettere l'accesso.

app.js – Validazione prima del download

Immagine che contiene testo, schermata, Carattere

Il contenuto generato dall'IA potrebbe non essere corretto.

Valida il token, cerca il file nel database, verifica che il file non sia scaduto, non abbia raggiunto il limite di download, e non sia stato eliminato, poi incrementa il contatore download e invia il file crittografato.

**Parametri**

* token: token dato dall’URL path parameter

La validazione avviene prima dell'incremento del contatore per evitare race conditions: se due richieste simultanee arrivano quando rimane 1 download, entrambe vedrebbero downloads\_remaining = 1, ma solo la prima passerebbe la validazione e incrementerebbe il contatore.

La seconda fallirebbe alla validazione successiva. L'errore 410 (Gone) invece di 404 indica che la risorsa esisteva ma non è più disponibile (scaduta o limite raggiunto), utile per distinguere errori di autenticazione da errori di disponibilità.

Ho scelto di eseguire la validazione prima dell'incremento per atomicità logica. Status code 410 per distinguere file non disponibili da file non esistenti.

models.js – Metodo validazione download

Immagine che contiene testo, schermata, Carattere

Il contenuto generato dall'IA potrebbe non essere corretto.

Esegue una query SQL atomica che verifica tre condizioni simultaneamente: il file non è stato eliminato (deleted\_at IS NULL), non è scaduto (expires\_at IS NULL OR expires\_at > NOW()), e non ha raggiunto il limite di download (download\_count < max\_downloads).

Restituisce true solo se tutte le condizioni sono soddisfatte.

**Parametri**

* fileId: ID del file nel database (intero)

## Rate limiting

Il sistema implementa rate limiting per prevenire abusi e attacchi di forza bruta, limitando il numero di richieste per indirizzo IP.

app.js – Implementazione rate limiting

Immagine che contiene testo, schermata, Carattere

Il contenuto generato dall'IA potrebbe non essere corretto.

Implementa un rate limiter in-memory che limita ogni indirizzo IP a massimo 10 richieste ogni 15 minuti. Utilizza una Map JavaScript per memorizzare lo stato di ogni IP (contatore richieste e tempo di reset). Se un IP supera il limite, viene restituito errore HTTP 429 (Too Many Requests).

**Parametri**

* req: oggetto request Express (contiene req.ip o req.connection.remoteAddress)
* res: oggetto response Express
* next: funzione middleware Express per continuare la catena

Lo status code 429 è standard HTTP per rate limiting.

La finestra di 15 minuti è un compromesso tra sicurezza e usabilità: troppo corta blocca utenti legittimi, troppo lunga permette più tentativi di brute-force.

## Implementazione Postfix e Dovecot

Il sistema utilizza un mail server locale per inviare la notifica opzionale al mittente dopo il primo download del file.  
La configurazione è minima, pensata per un ambiente scolastico / di sviluppo, senza dominio pubblico e senza necessità di DKIM/DMARC.

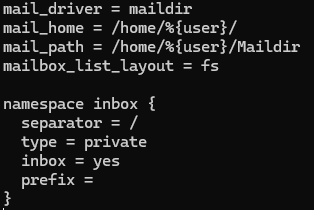
Installiamo i due programmi con il seguente comando:

sudo apt install postfix dovecot-core dovecot-imapd

Iniziamo modificando la configurazione di Postfix (/etc/postfix/main.cf), dovrebbe essere cosi:



La configurazione di Dovecot (/etc/dovecot/conf.d/10-mail.conf) deve essere cosi:



Questa configurazione ci permette di mandare e ricevere email localmente, con utenti locali della macchina.

Per far in modo che le email inviate dal server non appaiano come example@localhost oppure example@hostname ma come indirizzi reali, come [example@webfiletransfer.local](mailto:example@webfiletransfer.local), dobbiamo abilitare i **sender canonical maps**, per far ciò assicuriamoci di avere la stessa configurazione di Postfix come riportata sopra, dopodiché dobbiamo creare il file /etc/postfix/sender\_canonical e inserire il seguente contenuto:

noreply noreply@webfiletransfer.local

# Test

## Protocollo di test

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case:**  **Riferimento**: | TC-001.001  REQ-001 e 002 | **Nome:** | Selezione di un file dal browser |
| **Descrizione:** | Verificare che l’utente possa selezionare uno o più file tramite l’interfaccia web. | | |
| **Prerequisiti:** | |  | | --- | |  |  |  | | --- | | Browser moderno, funzionalità di upload attiva. | | | |
| **Procedura:** | 1. Aprire la pagina di upload (index). 2. Trascinare o selezionare un file nell’area di rilascio. 3. Selezionare un file. 4. Confermare l’upload senza opzioni aggiuntive. | | |
| **Risultati attesi:** | L’interfaccia web deve ritornare un link con il token per scaricare il file, il file deve essere aggiunto al database e dovrei vedere il file .enc dentro storage/ | | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test Case:**  **Riferimento**: | TC-001.002  REQ-001 e 002 | **Nome:** | |  | | --- | |  |  |  | | --- | | Cifratura dei metadati (nome file, dimensione) | |
| **Descrizione:** | |  | | --- | |  |  |  |  |  | | --- | --- | --- | | |  | | --- | |  |  |  | | --- | | Verificare che nome e dimensione del file siano cifrati lato client usando libsodium o equivalente. | | | | |
| **Prerequisiti:** | |  | | --- | |  |  |  | | --- | | Browser moderno, funzionalità di upload attiva. | | | |
| **Procedura:** | 1. Aprire la pagina di upload (index). 2. Trascinare o selezionare un file nell’area di rilascio. 3. Selezionare un file. 4. Confermare l’upload senza opzioni aggiuntive. 5. Verificare nell’endpoint metadata/ se vengono salvati i metadati | | |
| **Risultati attesi:** | Verificare nell’endpoint se viene ritornato il nome del file e altri metadati in formato JSON. | | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test Case:**  **Riferimento**: | TC-002  REQ-002 | **Nome:** | |  | | --- | |  |  |  | | --- | | Generazione URL contenente token univoco | |
| **Descrizione:** | |  | | --- | |  |  |  |  |  | | --- | --- | --- | | |  | | --- | |  |  |  | | --- | | Verificare che venga creato un link contenente un token casuale non prevedibile. | | | | |
| **Prerequisiti:** | |  | | --- | |  |  |  |  |  | | --- | --- | --- | | |  | | --- | |  |  |  | | --- | | File già caricato. |   (Test case TC-001.001 e TC-001.002 devono funzionare) | | | |
| **Procedura:** | 1. Caricare un file. 2. Generare link di condivisione. 3. Analizzare il token generato. | | |
| **Risultati attesi:** | L’URL contiene un token con entropia elevata (es. 256 bit), non raccorciabile o prevedibile. | | |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Test Case:**  **Riferimento**: | TC-003  REQ-003 | **Nome:** | |  | | --- | |  |  |  |  |  | | --- | --- | --- | | |  | | --- | |  |  |  | | --- | | Decifratura automatica file nel browser | | |
| **Descrizione:** | |  | | --- | |  |  |  |  |  | | --- | --- | --- | | |  | | --- | |  |  |  | | --- | | Verificare che, aprendo il link, il file venga decifrato usando il token incorporato nel link. | | | | |
| **Prerequisiti:** | |  | | --- | |  |   I test case TC-001 devono funzionare correttamente. | | |
| **Procedura:** | 1. Aprire il link di download. 2. Monitorare l’operazione lato browser. 3. Verificare che il file venga decifrato prima del salvataggio locale. | | |
| **Risultati attesi:** | Il file decifrato è identico all’originale. Nessun dato in chiaro transita sui server. | | |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Test Case:**  **Riferimento**: | TC-004.001  REQ-004 | **Nome:** | |  | | --- | |  |  |  |  |  | | --- | --- | --- | | |  | | --- | |  |  |  | | --- | | Configurazione scadenza link | | |
| **Descrizione:** | |  | | --- | |  |  |  |  |  | | --- | --- | --- | | |  | | --- | |  |  |  | | --- | | Verificare che il link non sia più valido dopo la scadenza configurata. | | | | |
| **Prerequisiti:** | |  | | --- | |  |   I test case TC-001 devono funzionare correttamente.  Assegnare una scadenza di 1 ora prima della conferma dell’upload del file. | | |
| **Procedura:** | 1. Aprire il link entro il limite: deve funzionare. 2. Attendere oltre la scadenza. 3. Riprovare a scaricare dal link. 4. Provare se l’endpoint metadata/ funzioni | | |
| **Risultati attesi:** | Dopo la scadenza il server risponde con errore (es. 410 Gone / link scaduto) sia per il download che l’endpoint metadata/  Anche dall’endpoint status/ si dovrebbe vedere che è scaduto il file | | |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Test Case:**  **Riferimento**: | TC-004.002  REQ-004 | **Nome:** | |  | | --- | |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | | |  | | --- | |  |  |  |  |  | | --- | --- | --- | | |  | | --- | |  |  |  | | --- | | Limite massimo di download configurabile | | | |
| **Descrizione:** | |  | | --- | |  |  |  | | --- | | Verificare che il token diventi invalido dopo un certo numero di download. | | | |
| **Prerequisiti:** | |  | | --- | |  |   I test case TC-001 devono funzionare correttamente.  Assegnare un limite di 3 download prima della conferma dell’upload del file. | | |
| **Procedura:** | |  | | --- | |  |  |  | | --- | | * 1. Effettuare 3 download: OK.   2. Provare un quarto download. | | | |
| **Risultati attesi:** | Dopo aver raggiunto il limite il server risponde con sia per il download che l’endpoint metadata/  L’endpoint status/ dovrebbe dire che rimangono 0 download (scaduto) | | |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Test Case:**  **Riferimento**: | TC-005  REQ-005 | **Nome:** | |  | | --- | |  |  |  |  |  | | --- | --- | --- | | |  | | --- | |  |  |  | | --- | | Notifica email al mittente | | |
| **Descrizione:** | |  | | --- | |  |  |  |  |  | | --- | --- | --- | | |  | | --- | |  |  |  | | --- | | Verificare che, se il mittente inserisce un’email, venga inviata una notifica al primo download. | | | | |
| **Prerequisiti:** | |  | | --- | |  |  |  |  |  | | --- | --- | --- | | |  | | --- | |  |  |  | | --- | | Mail server configurato; indirizzo email inserito dall’utente. | | | | |
| **Procedura:** | |  | | --- | |  |  |  | | --- | | * 1. Caricare file.   2. Generare link con email di notifica.   3. Effettuare download dal link.   4. Controllare la cassetta postale del mittente. | | | |
| **Risultati attesi:** | Un’email viene ricevuta indicando che il file è stato scaricato. | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test Case:**  **Riferimento**: | TC-006  / | **Nome:** | |  | | --- | |  |   Verifica attivazione del rate-limiting dopo più di 10 richieste |
| **Descrizione:** | |  | | --- | |  |   Il client invia più di 10 richieste consecutive allo stesso endpoint. Dopo la 10ª richiesta, il server deve rispondere con errore 429. | | |
| **Prerequisiti:** | |  | | --- | |  |  |  |  |  | | --- | --- | --- | | |  | | --- | |  |  |  | | --- | | Server avviato con middleware rateLimit applicato agli endpoint /api/upload, /download, /status, /metadata | | | | |
| **Procedura:** | |  | | --- | |  |  |  | | --- | |  |  |  | | --- | | 1. Avviare uno script bash che esegue un curl di un endpoint 15 volte. 2. Registrare il codice di risposta di ogni richiesta | | | |
| **Risultati attesi:** | Le prime 10 richieste -> 200 OK Le richieste dalla 11a in poi -> 429 Too Many Requests con JSON { error: "Rate limit exceeded. Please try again later." } | | |

## Risultati test

|  |  |  |
| --- | --- | --- |
| **Test Case:**  **Riferimento**: | TC-001.001   |  | | --- | | REQ-001 e 002 | |
| **Risultato** | |  | | --- | |  |  |  |  |  | | --- | --- | --- | | |  | | --- | |  |  |  | | --- | | Figura 8: Pagina di upload con file caricato | | | Il file viene correttamente caricato nel sito, significa anche libsodium è caricato correttamente.    Figura 9: Messaggio di successo dell'upload file |   Il link viene generato correttamente insieme al token, significa che il server ha ricevuto il file crittografato nel browser.    Figura 10: Query SQL backend inserimento file  Query nel DB eseguita correttamente.  Il test è passato. |

|  |  |  |
| --- | --- | --- |
| **Test Case:**  **Riferimento**: | TC-001.002   |  | | --- | | REQ-001 e 002 | |
| **Risultato** | |  | | --- | |  |   Usando il file caricato prima, se visitiamo l’endpoint metadata viene ritornato questo:    Figura 11: Contenuto endpoint metadata  Il test è passato. |

|  |  |  |
| --- | --- | --- |
| **Test Case:**  **Riferimento**: | TC-002   |  | | --- | | REQ-002 | |
| **Risultato** | |  | | --- | |  |   Usando il file caricato prima, analizziamo il token (qFCXmCkDfwuOjvdIczjJvEIIzmESubvxTGYqXhy5QnA) con CyberChef:  Innanzitutto possiamo vedere che il token è lungo 43 caratteri (giusto)    Figura 12: Conversione token da base64  Il token viene decodificato da base64 correttamente, non è ASCII leggibile, che è un ottimo segno.    Figura 13: Conversione raw byte in hex  Nell’output di prima trasformato in hex, ci sono esattamente 32 byte (256 bit) ogni coppia è 1 byte.  L’entropia TEORICA del token è ~7.9 nella scala di Shannon.  Il test è passato. |

|  |  |  |
| --- | --- | --- |
| **Test Case:**  **Riferimento**: | TC-003   |  | | --- | | REQ-003 | |
| **Risultato** | |  | | --- | |  |   Usando il file caricato prima, mettiamo il link nella pagina di download per eseguire la decifratura.    Figura 14: Pagina download con file scaricato  Il file viene correttamente decifrato, questo perché viene scaricato il file con il nome e estensione giusta, il file d’esempio era un video mp4 di 30 secondi del globo che gira, se apro il file dovrei vedere lo stesso video identico senza perdita di qualità ecc.    Figura 15: File video d'esempio  Il test è passato. |

|  |  |  |
| --- | --- | --- |
| **Test Case:**  **Riferimento**: | TC-004.001   |  | | --- | | REQ-004 | |
| **Risultato** | |  | | --- | |  |   Carichiamo un nuovo file nel sito, e impostiamo la scadenza tra 1 ora:    Figura 16: Impostazioni d'upload con scadenza  Carichiamo il file e Andiamo a vedere nell’endpoint status/ quando scadrà esattamente il file:    Figura 17: Endpoint status con scadenza  Per ora si può scaricare correttamente il file, vediamo tra 1 ora se il file scade correttamente (11:16)    Figura 18: Messaggio d'errore file scaduto  Il test è passato. |

|  |  |  |
| --- | --- | --- |
| **Test Case:**  **Riferimento**: | TC-004.002   |  | | --- | | REQ-004 | |
| **Risultato** | |  | | --- | |  |   Carichiamo un nuovo file nel sito, e impostiamo il limite di 1 download:    Figura 19: Impostazioni upload con limite  Carichiamo il file e Andiamo a vedere nell’endpoint status/ quanti download rimangono:    Figura 20: Download rimanenti prima  Scarichiamo il file una volta e andiamo a ricontrollare l’endpoint:    Figura 21: Download rimanenti dopo  Il file non è più scaricabile.    Figura 22: Messaggio d'errore limite raggiunto  Il test è passato. |

|  |  |  |
| --- | --- | --- |
| **Test Case:**  **Riferimento**: | TC-005   |  | | --- | | REQ-005 | |
| **Risultato** | |  | | --- | |  |   Carichiamo un nuovo file nel sito, e impostiamo la notifica opzionale prima del caricamento (si utilizza un email locale perché il mail server NON è impostato a mandare email a domini esterni):    Figura 23: Impostazioni upload con notifica  Carichiamo il file e andiamo ad aprire la casella delle email (in questo caso ho usato Thunderbird), scarichiamo il file e andiamo a vedere la casella:    Figura 24: Notifica Thunderbird dell'email    Figura 25: Contenuto email  Il test è passato. |

|  |  |  |
| --- | --- | --- |
| **Test Case:**  **Riferimento**: | TC-006   |  | | --- | | / | |
| **Risultato** | |  | | --- | |  |   Usando il seguente file bash (ratelimitingTest.sh) potremmo mandare multiple richieste allo stesso endpoint e attivare il sistema di ratelimiting:    Figura 26: Script bash per ratelimiting  Il risultato dopo aver eseguito lo script è il seguente:    Figura 27: Risultato script bash  Il test è passato. |
| **Commento extra** | Le prime 10 richieste danno 404 e non 200 questo perché l’endpoint si aspettava un parametro (token) nel path parameter che non li abbiamo dato in questo caso, quindi è normale.  Lo script bash è stato generato dall’IA con il seguente prompt:  «Generami uno script minimale in bash per eseguire 15 richieste curl a questo url 'http://localhost/status’ Deve anche stampare il numero della richiesta e lo status code, questo per testare il ratelimiting di un endpoint. » |

## Mancanze/limitazioni conosciute

* **Upload di più file**  
  Al momento si può caricare solo un file alla volta. Non ho implementato l’upload multiplo perché avrebbe richiesto un po' più di gestione lato frontend/backend e non era una priorità per questo progetto.
* **Nessun dominio per configurare DKIM/DMARC/SPF**  
  Non avendo un dominio vero, non posso configurare le protezioni necessarie per far funzionare l’invio di email a domini esterni.  
  Questo significa che le email possono funzionare in locale o in rete interna, ma non posso garantire l’invio verso caselle come Gmail, perché senza configurare i protocolli DKIM/DMARC le email non vengono mai inviate.

# Consuntivo

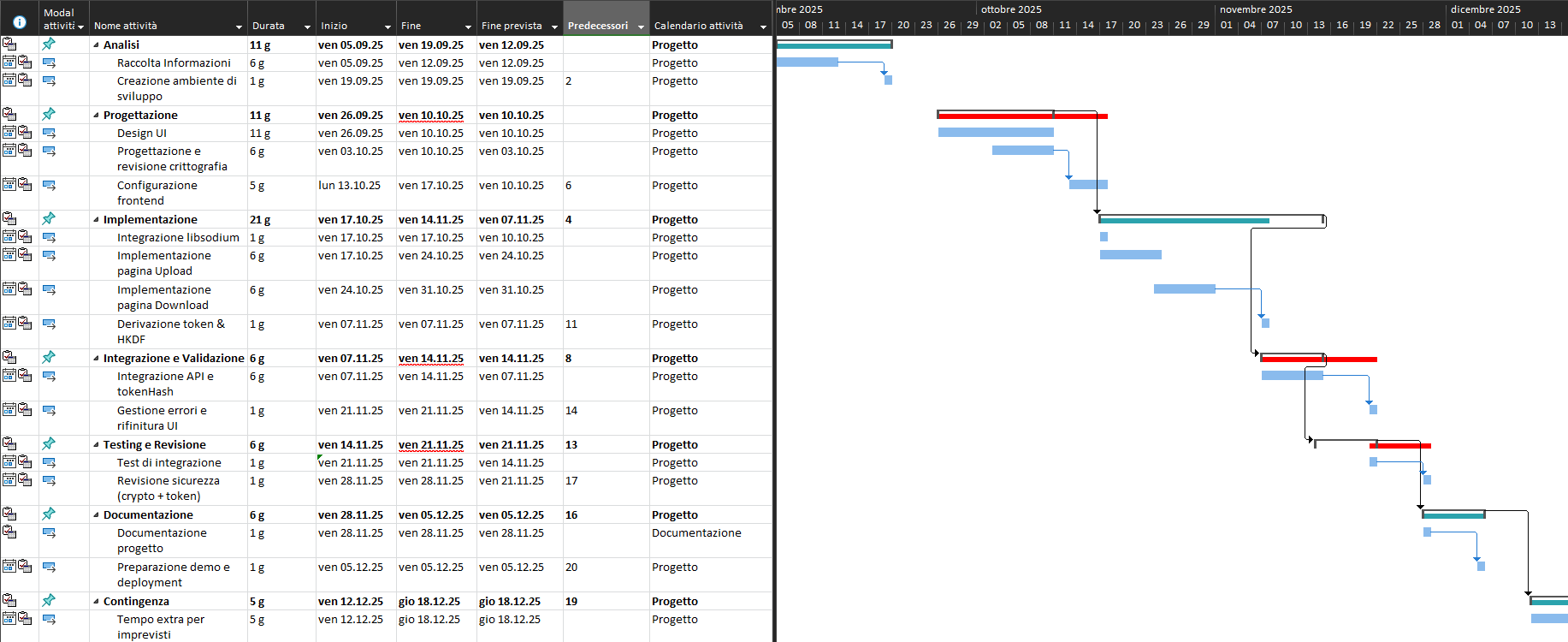


Figura 28: Gantt consuntivo

# Conclusioni

Il progetto Web File Transfer dimostra che è possibile realizzare un servizio di condivisione file che rispetti la privacy utilizzando la **crittografia E2E lato client**.  
Grazie all’architettura **zero-knowledge,** il server non può accedere ai contenuti dei file, nemmeno in caso di compromissione.

La soluzione utilizza un sistema di accesso basato su **token**, senza autenticazione utente, in cui solo chi possiede il token può decrittografare il file. Questo approccio riduce la fiducia richiesta nel server ed è un’alternativa ai servizi cloud tradizionali.

Il progetto non rappresenta una rivoluzione, ma una **buona implementazione pratica** di concetti noti come E2EE e zero-knowledge in un contesto web.  
I risultati sono in parte specifici del file sharing, ma i principi utilizzati sono**facilmente riutilizzabili** in altri ambiti che richiedono privacy, come messaggistica o storage sicuro.  
Una limitazione è la dipendenza da JavaScript e dal supporto alle API crittografiche del browser.

## Sviluppi futuri

Tra i possibili miglioramenti:

* aumento della sicurezza (Perfect Forward Secrecy, firme per integrità, autenticazione opzionale)
* nuove funzionalità (upload multipli, preview file, password aggiuntiva, API REST)
* migliori performance e scalabilità (crittografia in streaming, CDN, compressione)
* miglioramenti UX (barra di progresso, notifiche, QR code, temi)

## Considerazioni personali

Questo progetto mi ha permesso di applicare concretamente concetti di **crittografia e sicurezza**, andando oltre la teoria.  
Ho approfondito aspetti come key management, nonce handling e la progettazione di sistemi zero-knowledge.

Dal punto di vista dello sviluppo, ho migliorato il coordinamento tra frontend e backend, l’organizzazione del codice e l’attenzione ai dettagli, soprattutto per quanto riguarda la sicurezza.  
Le principali difficoltà hanno riguardato la gestione dei file, le **race conditions** e alcuni edge cases.

Nel complesso, il progetto è stato utile e formativo, fornendo una base solida per futuri lavori legati a **privacy, sicurezza e applicazioni web.**

# Glossario

|  |  |
| --- | --- |
| **Termine** | **Descrizione** |
| AEAD | **Authenticated Encryption Associated Data:** tipo di cifratura che garantisce sia la protezione dei dati sia l’autenticità del messaggio. |
| ArrayBuffer | Struttura dati JS che rappresenta byte grezzi (dati binari). |
| Audit Log | Registro che traccia le operazioni effettuate sul sistema (upload, download, ecc.). |
| Base64 / Base64 URL-safe | Metodo di codificare dati binari in testo: la versione URL-safe evita caratteri problematici nell’URL |
| Bruteforce | Tentativo di indovinare una chiave provando tutte le combinazioni possibili. |
| ChaCha20-Poly1305 | Algoritmo di crittografia moderno, veloce e molto sicuro, usato anche in TLS/HTTPS. |
| Ciphertext | Testo cifrato, cioè il risultato della crittografia. |
| Client-side | Operazioni svolte dal browser dell’utente, non dal server |
| Context (KDF) | Piccola stringa usata per ‘contestualizzare’ una derivazione chiave e prevenire collisioni. |
| Digest | Risultato finale dell’esecuzione di una funzione hash (es. HMAC SHA256). |
| E2E | **End-to-End Encryption**: crittografia dove solo il mittente e il destinatario possono leggere i dati. |
| FormData | API web che permette di inviare dati e file tramite richieste HTTP multipart. |
| HMAC-SHA256 | Funzione di hash autenticata che usa una chiave segreta per verificare integrità e autenticità. |
| KDF | **Key Derivation Function**: funzione che genera una chiave sicura partendo da un segreto iniziale. |
| MIME type | Tipo di file (es.: image/png, text/plain) usato dal browser per capire di che file si tratta. |
| Nonce | Valore casuale usato una sola volta per garantire sicurezza nella crittografia. |
| Open-source | Software che rende il codice sorgente liberamente accessibile a chiunque. |
| Path parameter | Parte di un URL che rappresenta un valore (es. /download/<token>). |
| Payload | Dati trasmessi o contenuti all’interno di una richiesta o di un file. |
| Salt | Valore casuale aggiunto a un segreto per renderlo più difficile da indovinare. |
| SessionStorage | Memoria del browser che conserva dati finché la scheda resta aperta |
| Side-channel attack | Tipo di attacco informatico che sfrutta tempi di esecuzione o consumo di energia per ottenere informazioni. |
| Token | Codice segreto che permette di decrittografare un file. |
| Uint8Array | Array di byte (0-255) usato in JS per manipolare dati binari |
| UUID v4 | Identificatore univoco generato casualmente, usato ad esempio come nome file. |
| Zero-knowledge | Architettura in cui il server non conosce mai i dati sensibili dell’utente. |

# Indice delle figure

[Figura 1: Diagramma Use case 6](#_Toc217048471)

[Figura 2: Diagramma di Gantt 1](#_Toc217048472)

[Figura 3: Schema di rete 2](#_Toc217048473)

[Figura 4: Schema ER 3](#_Toc217048474)

[Figura 5: Mockup sito web 4](#_Toc217048475)

[Figura 6: Procedura di crittografia ChaCha20Poly1305 8](#_Toc217048476)

[Figura 8: Definizione di HMAC (RFC 2104) 12](#_Toc217048477)

[Figura 9: Pagina di upload con file caricato 22](#_Toc217048478)

[Figura 10: Messaggio di successo dell'upload file 22](#_Toc217048479)

[Figura 11: Query SQL backend inserimento file 22](#_Toc217048480)

[Figura 12: Contenuto endpoint metadata 23](#_Toc217048481)

[Figura 13: Conversione token da base64 24](#_Toc217048482)

[Figura 14: Conversione raw byte in hex 24](#_Toc217048483)

[Figura 15: Pagina download con file scaricato 25](#_Toc217048484)

[Figura 16: File video d'esempio 25](#_Toc217048485)

[Figura 17: Impostazioni d'upload con scadenza 26](#_Toc217048486)

[Figura 18: Endpoint status con scadenza 26](#_Toc217048487)

[Figura 19: Messaggio d'errore file scaduto 26](#_Toc217048488)

[Figura 20: Impostazioni upload con limite 27](#_Toc217048489)

[Figura 21: Download rimanenti prima 27](#_Toc217048490)

[Figura 22: Download rimanenti dopo 27](#_Toc217048491)

[Figura 23: Messaggio d'errore limite raggiunto 27](#_Toc217048492)

[Figura 24: Impostazioni upload con notifica 28](#_Toc217048493)

[Figura 25: Notifica Thunderbird dell'email 28](#_Toc217048494)

[Figura 26: Contenuto email 28](#_Toc217048495)

[Figura 27: Script bash per ratelimiting 29](#_Toc217048496)

[Figura 28: Risultato script bash 29](#_Toc217048497)

[Figura 29: Gantt consuntivo 1](#_Toc217048498)

# Bibliografia

## Sitografia

<https://expressjs.com/>, Express - Sito ufficiale, 19-09-2025

[developer.mozilla.org](https://developer.mozilla.org/en-US/docs/Learn_web_development/Extensions/Server-side/Express_Nodejs/Introduction?utm_source=chatgpt.com), Express/Node introduction - MDN Web Docs, 19-09-2025

[doc.libsodium.org](https://doc.libsodium.org/secret-key_cryptography/aead/chacha20-poly1305?utm_source=chatgpt.com), ChaCha20-Poly1305 | Libsodium documentation, 03-10-2025

[doc.libsodium.org](https://doc.libsodium.org/secret-key_cryptography/aead/chacha20-poly1305?utm_source=chatgpt.com), XChaCha20-Poly1305 construction | Libsodium documentation, 03-10-2025

<https://en.wikipedia.org/wiki/Express.js>, Express.js - Wikipedia, 26-09-2025

<https://codegrind.it/documentazione/expressjs/introduzione>, Introduzione a Express.js, 26-09-2025

<https://www.youtube.com/watch?v=GRpS-1OKmXE>, *Deep Dive into ChaCha20Poly1305*, 10-10-2025

<https://www.youtube.com/watch?v=UeIpq-C-GSA>, *ChaCha Chiper*, 10-10-2025

<https://en.wikipedia.org/wiki/Key_derivation_function>, *KDF - Wikipedia*, 17-10-2025

[developer.mozilla.org](https://developer.mozilla.org/), *ArrayBuffer – MDN Web Docs*, 17-10-2025

<https://www.youtube.com/watch?v=ScZZoHj7mqY>, *Deep Dive into Blobs, Files and ArrayBuffer*, 3-10-2025

<https://www.youtube.com/watch?v=28VZ-AE2t2I&t=147s>, *Videotutorial Postfix e Dovecot*, 14-11-2025

# Allegati

Lista degli allegati:

* Diagramma di flusso --- DiagrammaDiFlusso.png
* Gantt --- Gantt.mpp
* Logo --- Logo.png
* Mockup --- Mockup.png
* Schema di rete --- SchemaDiRete.png
* Schema ER --- SchemaER.png
* UseCase -- UseCase.vsdx