

Relazione progetto di laboratorio
Cyber Security - A.A. 2023/2024
SecureShare

Chierchia Kevin - 147798 - chierchia.kevin@spes.uniud.it

10 giugno 2024

[GitHub Repository](#)

Indice

1	Introduzione	3
1.1	Descrizione dell'applicazione web	3
2	Registrazione, login, sessioni e DB utenti	3
2.1	Sistema di registrazione e creazione chiavi RSA	3
2.2	Sistema di Login	4
2.3	Sessioni	5
2.4	Database degli utenti	5
3	Upload ed encryption, download e decryption	6
3.1	Invio di file cifrati - Send	6
3.1.1	Ricerca utente	7
3.1.2	Selezione del file	8
3.2	Decifratura e scaricamento file - Inbox	8
3.2.1	Selezione della chiave privata	10
3.2.2	Decifratura della chiave AES	10
3.2.3	Decifratura e download del file	10
3.3	Pubblicazione file cifrati - Publish	10
3.3.1	Selezione di un file e scelta password	10
3.3.2	Decifratura e download dei file	11
4	Altre informazioni	12
4.0.1	Istruzioni per il setup del sistema	12
4.0.2	Problemi riscontrati	12

1 Introduzione

Questa relazione ha lo scopo di mostrare i dettagli architetturali e implementativi dell'applicazione web sviluppata, con un particolare focus sugli aspetti inerenti alla sicurezza.

1.1 Descrizione dell'applicazione web

L'applicazione web sviluppata è una semplice piattaforma di file sharing, con file hostati dal web server stesso. La feature chiave è che tutti i file caricati sul web server, vengono prima cifrati lato client, e poi caricati direttamente sotto forma di dati cifrati con AES.

Sono presenti due modalità principali di caricamento dei file:

- Invio a un utente specifico (più sicuro)
- Pubblicazione del file su dashboard pubblica

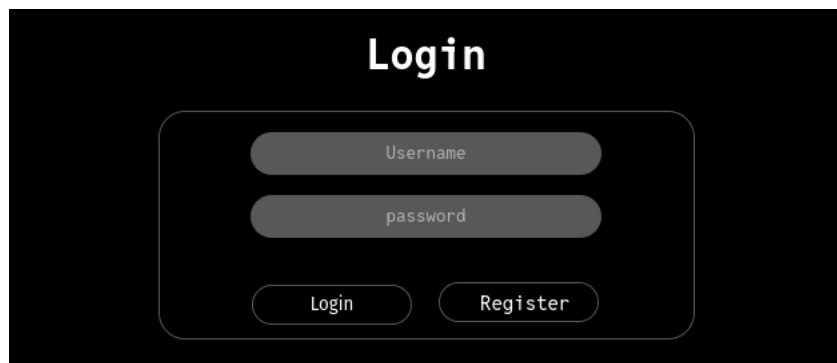
L'applicazione usa Apache2 per eseguire il webserver e PostgreSQL come base di dati. Offre altre feature secondarie come l'encryption del traffico web tramite TLS e l'utilizzo di sessioni per una maggiore confidenzialità dei file cosiddetti privati.

2 Registrazione, login, sessioni e DB utenti

In questa sezione si illustra la pagina di registrazione, quella di login, il metodo con cui viene verificata la corretta autenticazione di un utente lato server e il metodo con cui sono archiviati le credenziali degli utenti sul database.

2.1 Sistema di registrazione e creazione chiavi RSA

Per iscriversi all'applicazione è sufficiente accedere al sito, (**qualsiasi pagina si provi ad accedere, senza sessione attiva, si verrà sempre reindirizzati alla pagina di login**). Il prototipo dell'applicazione può essere acceduto dopo aver avviato apache2, al URL <https://project-domain.local/>, si visualizzerà il seguente form:

A screenshot of a login form titled "Login" in white text on a black background. The form is enclosed in a rounded rectangle with a thin white border. It contains two input fields: "Username" and "password", both with placeholder text in a light gray font. Below the input fields are two buttons: "Login" and "Register", both with white text on a dark gray background.

Una volta alla pagina di login, cliccando "Register" si accederà ad una pagina pressochè identica ma che ha lo scopo di registrare l'utente nel DB se l'username inserito non esiste.

Al momento del caricamento della pagina viene generata una **coppia di chiavi RSA** tramite la libreria `subtle.crypt` di JavaScript (direttamente sul browser dell'utente). Una volta verificato che l'utente non esiste, al DB verrà aggiunto un record nella tabella `users` con l'username inserito e un'hash della password (il web server non salva la vera password dell'utente). Maggiori informazioni sul Database SQL verranno illustrati più avanti in questo documento.

Nella tabella `pkey` invece verrà inserito un record con il nome dell'utente e la chiave pubblica generata.

Se la registrazione ha successo, il client viene notificato e sia la chiave pubblica che privata vengono salvate sulla `localStorage` del browser (valida solo all'interno del dominio dell'applicazione).

Questa scelta implementativa comporta delle feature di sicurezza ma anche delle limitazioni lato utente. Se un utente per esempio si registra dal suo laptop, le chiavi RSA saranno salvate sulla `localStorage` del browser su quello specifico laptop, e anche se dovesse effettivamente riuscire ad autenticarsi con le sue credenziali

da un altro dispositivo, non potrebbe scaricare i file privati che gli vengono mandati da altri utenti. Questo implica che anche in caso di violazione del DB e fuga di dati, gli utenti non rischiano che i propri file privati personali vengano acceduti da persone indesiderate in quanto solo loro stessi possiedono la chiave privata RSA.

2.2 Sistema di Login

Una volta registrato, l'utente può effettivamente effettuare il login, inserendo le credenziali che vengono mandate tramite POST, con traffico crittografato tramite TLS:

Status	Meth...	Domain	File	Initiator	Type	Transferred	Size
200	POST	project_do...	login.php	document	html	583 B	660 B
404	GET	project_do...	favicon.ico	FaviconLoade...	html	cached	283 B

2 requests | 943 B / 583 B transferred | Finish: 64 ms | DOMContentLoaded: 27 ms | load: 30 ms

Headers | Cookies | Request | Response | Timings | **Security**

▼ Connection:

Protocol version: "TLSv1.3"

Cipher suite: "TLS_AES_128_GCM_SHA256"

Key Exchange Group: "x25519"

Signature Scheme: "RSA-PSS-SHA256"

▼ Host project_domain.local:

HTTP Strict Transport Security: "Disabled"

Public Key Pinning: "Disabled"

▼ Certificate:

▼ Issued To

Common Name (CN): "CybersecurityLab"

Organization (O): "Certification authority"

Organizational Unit (OU): "<Not Available>"

▼ Issued By

Common Name (CN): "CybersecurityLab"

Organization (O): "Certification authority"

Organizational Unit (OU): "CybersecurityLab"

▼ Period of Validity

Begins On: "Mon, 03 Jun 2024 08:01:40 GMT"

Expires On: "Tue, 03 Jun 2025 08:01:40 GMT"

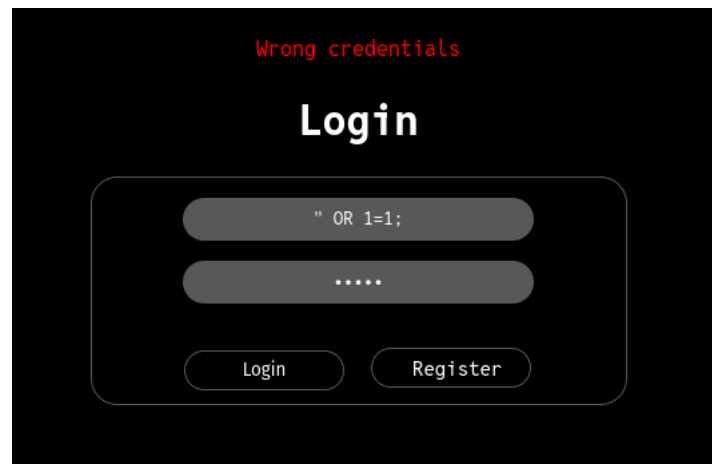
▼ Fingerprints

Il controllo per l'autenticazione sul DB viene fatto tramite una funzione SQL (non una semplice query), il che rende molto più difficile la possibilità di inserire comandi malevoli all'interno del form, per esempio nel tentativo di iniettare query SQL. La funzione in questione effettua il controllo, e se l'utente esiste restituisce un valore booleano (0 o 1). Oltre a ciò viene comunque effettuata una validazione dell'input prima del controllo, ma da diversi test è risultata non necessaria data la presenza della funzione SQL.

Un estratto di login_check.php:

```
1 $username = $_POST['username'];
2 $pwd = $_POST['password'];
3
4 $stmt = $pdo->prepare('SELECT verify(:username, :password)');
5 $stmt->execute(['username' => $username, 'password' => $pwd]);
6
7 $result = $stmt->fetch();
8 // $result['verify'] can be only 0 or 1
9 if($result['verify']){
10     $authenticated = $result['verify']==1;
11
12     if ($authenticated) {
13         $_SESSION["session_username"] = $username;
14     }
15 }
```

Un esempio pratico di SQL injection inefficace (vale lo stesso anche per gli apici singoli):



2.3 Sessioni

Una volta validate le credenziali dell'utente, l'accesso viene garantito creando nella struttura `_SESSION` una variabile contenente il nome dell'utente (come si può vedere dall'estratto di codice nella sezione precedente a riga 13). Questa variabile viene controllata prima del caricamento di ogni pagina per verificare che vi sia una sessione valida. **Le sessioni durano massimo 24 minuti**, dopodichè sarà necessario autenticarsi nuovamente.

2.4 Database degli utenti

La parte di database che si occupa dell'archiviazione dei dati degli utenti è molto semplice:

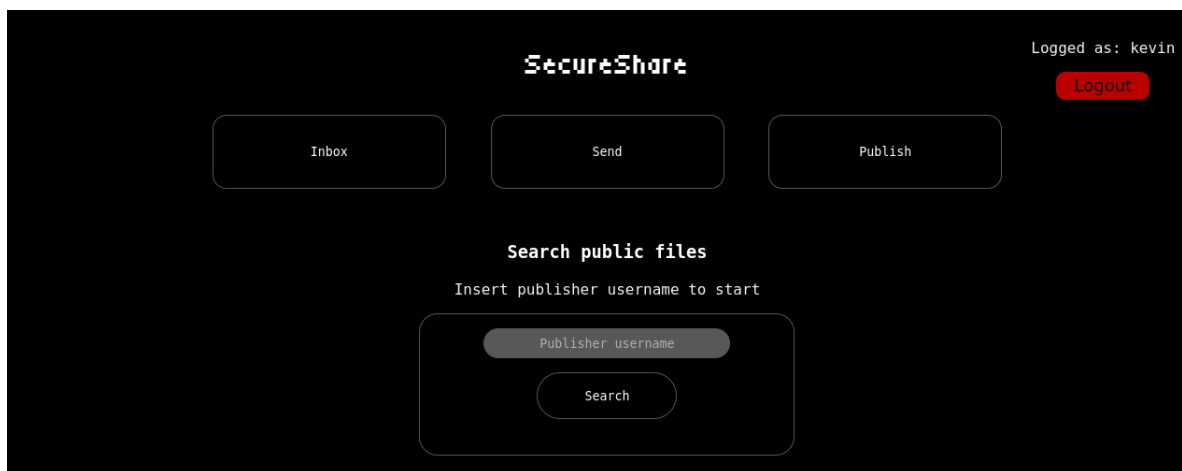
```
1 CREATE TABLE users (
2 username varchar(20) NOT NULL PRIMARY KEY,
3 pswd varchar(255) NOT NULL,
4 );
5
6
7 CREATE TABLE pkey (
8 user varchar(20) NOT NULL,
9 pkey TEXT PRIMARY KEY,
10 CONSTRAINT foreign_pkey_key FOREIGN KEY (user) REFERENCES users(username) ON DELETE CASCADE
11 );
```

Sono inoltre presenti due funzioni che servono a verificare le credenziali (username e password) per il login, e a verificare che un utente esista prima di procedere all'invio di un file:

```
1 CREATE or REPLACE FUNCTION verify(user_name varchar(20), passwd varchar(255))
2 RETURNS int
3 language plpgsql;
4 as
5 $$
6 DECLARE
7 verified int;
8 BEGIN
9 select 1 into verified from users where username=user_name and passwd=crypt(passwd, passwd);
10 return coalesce(verified, 0);
11 END;
12 $$
13
14 CREATE or REPLACE FUNCTION does_exist(user_name varchar(20))
15 RETURNS int
16 language plpgsql
17 as
18 $$
19 DECLARE
20 exists int;
21 BEGIN
22 SELECT 1 into exists FROM users WHERE username=user_name;
23 return coalesce(exists, 0);
24 END;
25 $$;
```

3 Upload ed encryption, download e decryption

Superato il login, ci si troverà nella pagina **Home**:



I tre pulsanti mostrano le principali feature dell'applicazione:

- Ricevere o cercare file cifrati
- Inviare file cifrati
- Pubblicare file cifrati

Inoltre è possibile cercare i file che un utente ha pubblicato.

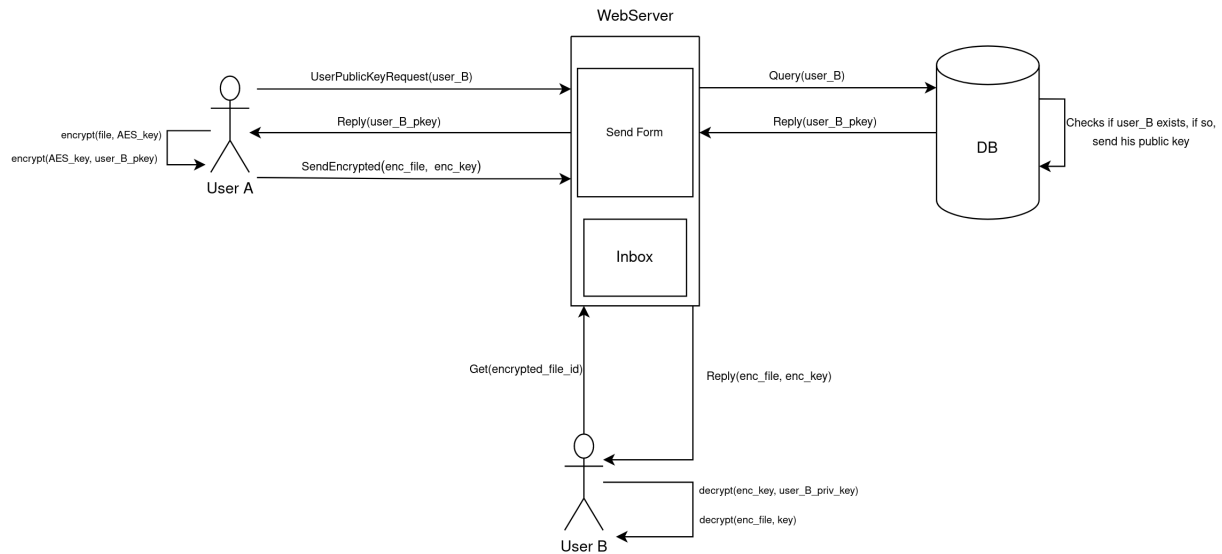
3.1 Invio di file cifrati - Send

Questa feature è la più sicura, e la più complessa. Prevede l'utilizzo di cifratura **AES256** per il file, tramite una chiave generata randomicamente lato client. Dopodichè **la chiave viene cifrata con la chiave pubblica del utente destinatario**, questo implica che solo il destinatario stesso, unico detentore della

chiave privata, potrà decifrare la chiave AES, e dunque il file. Nemmeno il server stesso possiede le chiavi per poter effettuare la decifratura.

Prima però è appunto necessario verificare l'esistenza dell'utente, e se esiste ottenere la sua chiave pubblica.

Segue un diagramma che descrive il processo:



3.1.1 Ricerca utente

Cliccando sul pulsante "Send", si viene reindirizzati nella pagina **search_user**:

The screenshot shows a web interface for searching a recipient's username. At the top right, it says "Logged as: kevin" with a red "Logout" button. The main heading is "Search recipient username". Below this is a form with a text input field labeled "Recipient username", a "Search" button, and a "Cancel" button.

Il form richiede l'inserimento di un utente, e se l'username esiste, si verrà reindirizzati alla pagina per la selezione del file.

Anche in questo caso l'input viene validato prima di effettuare la query. La query in ogni caso restituisce solamente chiavi pubbliche, dunque nessuna informazione confidenziale dell'destinatario.

Lato client la chiave viene recuperata tramite una variabile in `_SESSION` chiamata "serialized_key". La variabile viene istanziata lato server al momento della validazione dell'username inserito nella pagina precedente.

La chiave verrà poi deserializzata perchè per l'effettivo caricamento della chiave sul DB è necessario esportarla dalla struttura dati che la contiene e serializzarla.

Per informazioni più dettagliate si allega la documentazione di [subtle.cryptkey](#), libreria con la quale sono state generate le chiavi RSA.

```

1 <?php
2     $_SESSION['serialized_key'] = serialize($_SESSION['recipient_pkey']);
3     $session_value=(isset($_SESSION['serialized_key']))?$_SESSION['serialized_key']:'';
4     preg_match('/\{[^{}]*\}/', $session_value , $matches);
5     ?>
6
7     <script type="text/javascript">
8
9         var cleaned = '<?php echo $matches[0];?>';
10        var keyObj = JSON.parse(cleaned);
11        var key;
12
13        window.crypto.subtle.importKey(
14            'jwk', keyObj,
15            { name: 'RSA-OAEP', hash: { name: 'SHA-256' } },
16            true, ['encrypt']
17        )
18        .then(function(cryptoKey) {
19            key = cryptoKey;
20        })
21        .catch(function(error) {
22            console.error("Error while creating AES key:", error);
23        });
24    </script>

```

3.1.2 Selezione del file

Se l'utente inserito nel form esiste, oltre al recupero della chiave pubblica, si viene reindirizzati nella pagina **select_file**. Dopo la selezione del file, se si clicca su Send, la pagina verrà ricaricata e si visualizzerà una notifica di invio completato con successo:



Un esempio di come compaiono i dati cifrati sul database inviando un semplice file di testo contenente 13 caratteri:

```

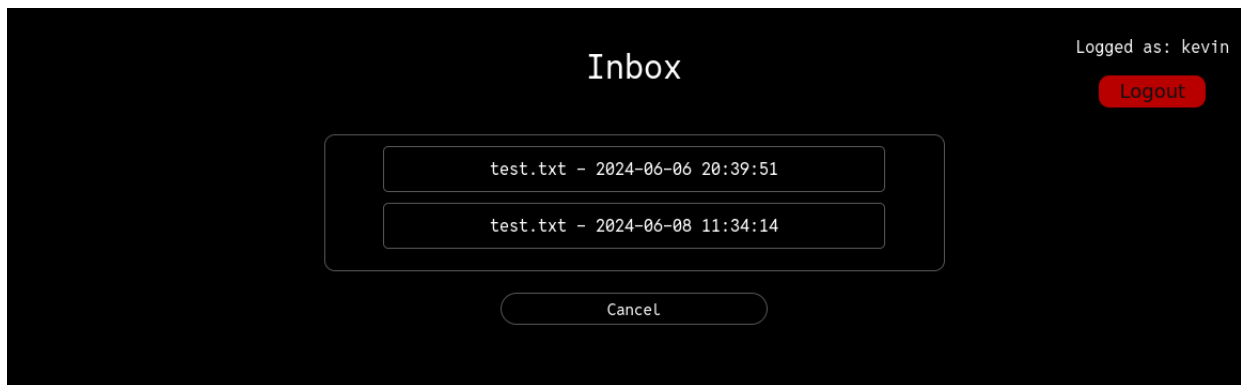
cysec=# select encrypted_data from encrypted_files where id = 64;
          encrypted_data
-----
U2FsdGVkX18h/du/j7IxCukP4zRXN2JXCa46DwBDd2boh5c8aTQR02/r2MLo0eF
(1 row)

cysec=#

```

3.2 Decifratura e scaricamento file - Inbox

Per scaricare i file che vengono inviati dagli altri utenti al proprio username, è sufficiente cliccare sul pulsante inbox nella home, per venir reindirizzati alla pagina **Inbox**:



Cliccando su ciascun file, se si possiede la chiave privata dell'utente con cui si è loggati, verrà scaricato il file decifrato.

Tutto il codice per invertire il processo di cifratura si trova all'interno di uno script JavaScript nel file inbox.php ed è abbastanza complesso, qui viene illustrata una piccola parte che mostra la decifratura della chiave AES:

```
1  async function downloadFile(encryptedData, encryptedKey) {
2      let privateKey;
3      try {
4          let currentUsername = sessionStorage.getItem("currentUsername");
5          if (!currentUsername) {
6              console.error("Current user not found in sessionStorage");
7              return;
8          }
9
10         // Get the right key from localStorage
11         for (let i = 0; i < localStorage.length; i++) {
12             let key = localStorage.key(i);
13             if (key.startsWith('privateKey') && key.includes(currentUsername)) {
14                 let privateKeyData = JSON.parse(localStorage.getItem(key));
15                 privateKey = await window.crypto.subtle.importKey(
16                     'jwk', privateKeyData,
17                     { name: 'RSA-OAEP', hash: { name: 'SHA-256' } },
18                     true, ['decrypt']
19                 );
20                 break;
21             }
22         }
23
24         if (!privateKey) {
25             console.error('No key found for current user.');
```

```
26             return;
27         }
28
29         // decrypt AES key
30         let decodedKey = base64ToArrayBuffer(encryptedKey);
31         let decryptedKeyBuffer = await decryptMessage(privateKey, decodedKey);
32         let decryptedKey = new TextDecoder().decode(decryptedKeyBuffer);
33
34         // Decrypt the file data with the AES key
35         let decryptedData = atob(CryptoJS.AES.decrypt(encryptedData, decryptedKey).
36             toString(CryptoJS.enc.Utf8));
37
38         // Create a blob and initiate the download
39         let blob = new Blob([decryptedData], { type: 'text/plain' });
40         let url = URL.createObjectURL(blob);
41         let a = document.createElement('a');
42         a.href = url;
43         a.download = "<?php echo $file['file_name']; ?>";
44         document.body.appendChild(a);
45         a.click();
46         window.URL.revokeObjectURL(url);
47     } catch (err) {
48         console.error('Errore durante la decifratura:', err);
49     }
50 }
```

All'interno del codice vi sono alcune sezioni da commentare per una migliore comprensione del funzionamento.

3.2.1 Selezione della chiave privata

All'interno del for **da riga 10 a riga 22** viene cercata, all'interno di localStorage del browser, una chiave privata che contenga il nome dell'utente della sessione attiva. Questo avviene poichè **è possibile registrarsi con utenti multipli da un unico dispositivo**, e per evitare di sovrascrivere le chiavi degli utenti con cui ci si è registrati in precedenza, le chiavi contengono il nome dell'utente. Questo crea il problema di selezionare la chiave corretta in fase di decifratura. Il ciclo for risolve questo problema.

3.2.2 Decifratura della chiave AES

Da riga 30 a riga 32 avviene l'effettiva decifratura della chiave AES, utilizzando la chiave privata ottenuta nella sezione precedente. La chiave viene innanzitutto **trasformata da base64 a arrayBuffer** in quanto, per essere salvata sul DB in maniera consistente ed evitare perdita di dati a causa di caratteri non validi, è stata precedentemente codificata in base64. Avendo la chiave come arrayBuffer è possibile utilizzare la funzione **crypto.subtle.decrypt** per decifrarla, la funzione asincrona decryptMessage si occupa proprio di questo. Essendo una funzione asincrona, bisogna attenderne il completamento tramite la keyword **await** (poichè il valore ritornato altrimenti sarebbe una **Promise**).

3.2.3 Decifratura e download del file

I dati cifrati vengono poi decifrati a **riga 35** con la funzione CryptoJS.AES.decrypt, non è la stessa libreria usata per la generazione delle chiavi RSA ma quella usata per la cifratura dei file durante l'invio descritto nelle sezioni precedenti. **Da riga 38 a riga 45** si gestisce l'effettivo download del file.

3.3 Pubblicazione file cifrati - Publish

Questa feature è simile all'invio di file cifrati ad utenti specifici ma con alcuni passaggi in meno. Il file viene anche in questo caso cifrato con AES, ma la chiave usata viene scelta arbitrariamente dall'utente e non viene cifrata, poichè non verrà salvata sul server. Questa chiave dovrà essere immessa come password dagli utenti che vorranno poi scaricare il file.

3.3.1 Selezione di un file e scelta password

Se dalla **Home** si clicca sul pulsante Publish si viene reindirizzati alla pagina **Publish**:



La pagina è molto simile alla pagina **select.file** della feature Send in quanto il funzionamento è molto simile, ma è presente un textfield nel form per l'inserimento della password con cui il file verrà cifrato.

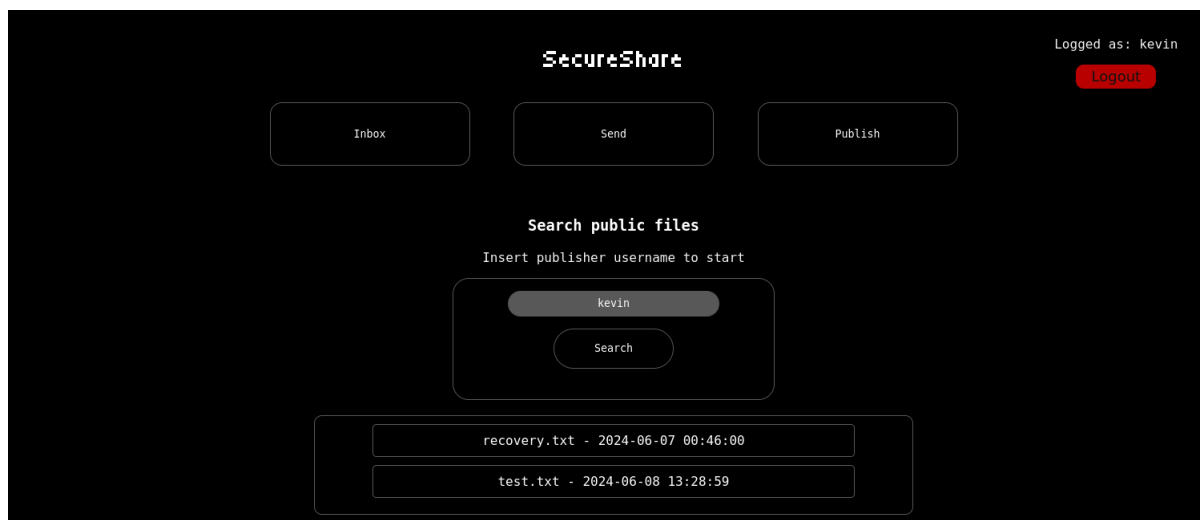


Una volta selezionato il file e inserita una password, il file verrà caricato e si visualizzerà una notifica che confermerà o meno l'avvenuto upload (in caso di errore si visualizzerà una notifica simile ma rossa e con un messaggio di esito negativo):

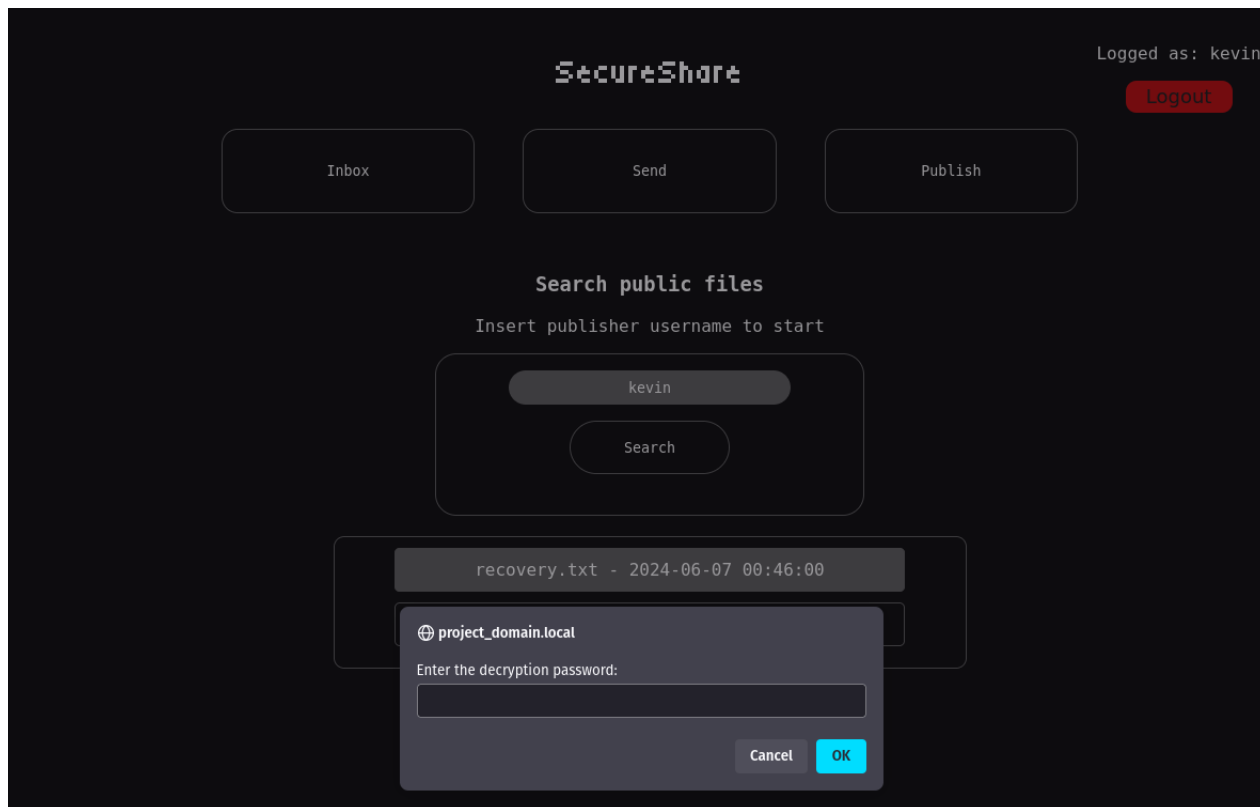


3.3.2 Decifratura e download dei file

Infine, per visualizzare i file pubblicati da un utente, dalla **Home** è possibile compilare il form in basso inserendo il nome di un publisher, e verranno caricati tutti i file che ha pubblicato:



Cliccando sui file presenti in questa vista, un pop up comparirà e verrà richiesto l'inserimento della password. L'idea che c'è alla base di questa feature, è la semplificazione dell'invio di file cifrati a destinatari multipli, ai quali **la password può essere comunicata su un canale esterno**, considerato sicuro, per esempio, via email.



Un vantaggio di questo approccio è che si può accedere ai file pubblicati da un utente anche senza disporre della propria chiave privata, per esempio se si vuole accedere a dei file da un dispositivo diverso da quello con cui ci si è registrati. **Come nel caso del send, la decifratura del file è a carico del client.**

4 Altre informazioni

4.0.1 Istruzioni per il setup del sistema

All'interno del file zip si trovano i file **userManual.pdf** e **instructions.txt** (contengono le stesse informazioni, ma il txt viene fornito per copiare in maniera più veloce i comandi) contenenti le istruzioni per una rapida configurazione del sistema, in particolare per configurare Apache2 e Postgres. Oltre ai 2 file è presente un file **sql_commands.sql** contenente una transazione per creare tutto il database, con le relazioni e le funzioni usate. All'interno della transazione è possibile vedere anche come sono definite tutte le tabelle.

4.0.2 Problemi riscontrati

L'idea iniziale era quella di creare un sistema per l'encryption di qualsiasi tipo di file, ma a causa di un problema riscontrato con la libreria CryptoJS e con l'encoding di file con un formato MIME diverso da plain/text, il risultato finale funziona solo con file di testo. La decifratura in linea teorica funziona con i file di formato diverso, tuttavia la codifica dei file, per esempio di un png, sembra non essere più recuperabile, o per lo meno, non con l'attuale implementazione. Se si prova a inviare un png, e a decifrarlo poi, si noterà che l'header del file è quasi leggibile (si riescono a intravedere dei caratteri validi, come la data) mentre il contenuto vero e proprio del file, viene codificato probabilmente con UNICODE e l'immagine viene aperta come un file di testo, ovviamente incomprensibile.