

# Rapport de projet



**Nom du projet :** Data Shielder

**Membres du groupe :**

- Mustapha OUMEZIANE (chef de projet)
- Enzo FERNANDEZ
- Evan LEGOUEZ-RICOCE

**Durée du projet :** Janvier 2024 - Juin 2024

# **Sommaire**

I - Le but de Data Shielder.....	3
II - Tâches réalisées.....	5
● Pourquoi le RSA ?.....	6
● Extension .ds.....	9
● Gestionnaire de mot de passe.....	10
● Fonctions codées.....	10
● Chiffrement ECC.....	13
● Site web.....	15
● Fonctionnalité de l'interface Utilisateur.....	18
III - L'avancée du projet.....	31
● Tableau récapitulatif .....	31
IV - Ressentis positifs.....	32
● Evan.....	32
● Enzo.....	32
● Mustapha.....	33
V - Ressentis négatifs.....	34
● Evan.....	34
● Enzo.....	35
● Mustapha.....	35
VI - Problèmes rencontrés.....	37
● Evan.....	37
● Enzo.....	38
● Mustapha.....	39
VII - Conclusion.....	40

## **I - But du Projet**

Compte tenu de l'importance de la cybersécurité aujourd'hui, notre objectif est de créer des logiciels capables de protéger vos données. Le monde est de plus en plus numérisé, le stockage et la transmission des données sont essentiels à son bon fonctionnement. C'est pourquoi il est essentiel de protéger vos données afin d'éviter toute forme de corruption. Nous avons donc pensé à créer un logiciel qui crypte les données, par exemple un fichier contenant des mots de passe, et qui est capable de déchiffrer ces fichiers.

Notre projet vise à crypter les fichiers textes fournis par l'utilisateur. Il sera capable de renvoyer un fichier .ds (une extension que l'on détaillera plus tard) avec un mot de passe associé. Ensuite, notre projet aura également la possibilité de déchiffrer ces fichiers, ce qui ne sera possible qu'en fournissant le mot de passe correctement associé au fichier.

Les propriétés de l'extension sont fournis dans la partie dédiée

Nous souhaitons mettre en œuvre plusieurs méthodes de cryptage afin d'élargir la gamme des options disponibles ainsi qu'une interface intuitive et facile d'utilisation

## **II - Tâches réalisées**

### **Répartition des tâches principales:**

- Moustapha :
  - création de l'extension
  - fonctions de cryptage RSA
  - Site web
  - architecture logiciel
- Enzo :
  - Interface graphique fonctionnelle
  - Chiffrement par courbe elliptique
- Evan :
  - Fonctionnalité de l'interface Utilisateur

## POURQUOI LE RSA ?

Parlons de logarithme discret:

Le logarithme discret est un concept clé utilisé dans de nombreux systèmes de cryptographie, y compris le chiffrement RSA. Pour comprendre le logarithme discret, il est important de d'abord comprendre ce qu'est un groupe cyclique.

Un groupe cyclique est un ensemble dans lequel il existe un élément de base, appelé générateur, qui, lorsqu'il est combiné avec lui-même un certain nombre de fois, génère tous les autres éléments du groupe. Par exemple, dans l'arithmétique modulaire, un groupe cyclique est formé par les restes de la division euclidienne par un nombre premier  $p$ . Dans ce groupe, l'exponentiation modulaire ( $g^x \bmod p$ ) joue un rôle central.

Le problème du logarithme discret consiste à trouver  $x$  dans l'équation  $g^x \equiv y \pmod{p}$ , où  $g$  est le générateur,  $y$  est un élément donné du groupe et  $p$  est le modulo. Le problème est difficile car il est difficile de déterminer  $x$  même si  $g$ ,  $y$  et  $p$  sont connus. En d'autres termes, il est difficile de trouver l'exposant  $x$  à partir du résultat  $y$  et de la base  $g$ .

Dans le contexte de la cryptographie, le logarithme discret est utilisé pour rendre difficile la récupération de la clé privée à partir de la clé publique dans les systèmes de cryptographie asymétrique, tels que RSA et Diffie-Hellman. Dans RSA, par

exemple, la clé publique consiste en n (le produit de deux nombres premiers), qui est utilisé pour chiffrer les messages, tandis que la clé privée consiste en d, où d est l'inverse modulaire de l'exposant de chiffrement e dans l'espace de calcul de la fonction d'Euler. La sécurité de RSA repose sur le fait que calculer d à partir de e et n est équivalent à résoudre un problème de logarithme discret difficile.

RSA (Rivest-Shamir-Adleman) est un système de cryptage asymétrique largement utilisé dans lequel une paire de clés est utilisée : une clé publique pour le cryptage et une clé privée pour le décryptage.

Génération des clés :

1. Deux nombres premiers distincts, p et q, sont choisis.
2. Leur produit n est calculé :  $n=p \times q$ .
3. La fonction d'Euler  $\phi(n)$  est calculée :

$$\phi(n)=(p-1) \times (q-1)$$

4. Un exposant de chiffrement e est choisi, habituellement petit et premier avec  $\phi(n)$ .
5. L'exposant de déchiffrement d est calculé de sorte que  $d \times e \equiv 1 \pmod{\phi(n)}$

Chiffrement :

1. Un message  $M$  est représenté par une séquence de nombres.
2. Chaque nombre est chiffré avec la clé publique  $(n,e)$  en utilisant la formule  $C \equiv M^e \pmod{n}$
3. Les nombres chiffrés forment le message chiffré.

Déchiffrement :

1. Le message chiffré est déchiffré avec la clé privée  $(n,d)$  en utilisant la formule  $M \equiv C^d \pmod{n}$
2. Les nombres déchiffrés sont convertis en caractères pour obtenir le message d'origine.

La sécurité de RSA repose sur la difficulté de factoriser en ses facteurs premiers  $p$  et  $q$ . La clé privée  $d$  ne peut être trouvée efficacement sans connaître  $p$  et  $q$ , ce qui rend le système robuste contre les attaques de force brute. et c'est ce que l'on cherche avec notre logiciel

## **Extension .ds:**

L'idée même de ce projet est l'extension .ds comme data shilder. A l'intérieur, plusieurs éléments sont organisés d'une manière bien précise pour le bon fonctionnement du processus. les arguments sont séparé par “///” : tout d'abord la signature DATA\_SHIELDER, si le fichier ne commence pas par ces caractères, il n'est pas reconnu comme étant au bon format.Puis le nom du fichier original, ce qui permet de retrouver le nom original même si le nom sous .ds a été modifié. Ensuite le hash du mot de passe entré au moment du chiffrement, l'extension du fichier, ici txt, par la suite il y a le mode de cryptage (dans l'exemple “mode” non spécifié). Le nombre qui commence par 25477 est utilisé dans le chiffrement RSA, et celui commençant par 158698 est en réalité la clé privée mais chiffré elle aussi avec cette fois le mot de passe. Cela permet donc de ne pas pouvoir déchiffrer le fichier en entrant dans l'extension sinon il n'y aurait plus de sécurité. Enfin le reste des éléments est le message crypté. Pour le RSA les clé publiques et privé sont générées aléatoirement donc pas de déterminisme avec le mot de passe comme dans la version précédente.

`DATA_SHIELDER///test///4da3376323046a3bb6759f0a3f4ae7100a0567950c53ee42d2e19201baaa6dfc///txt///mode///`

`//mode///25477306558557943///158698748033629202578503495996365822984///86812553324672///207616015289871///194871710000000//150363025899136//20761601528`

## **Gestionnaire de mot de passe:**

Le mot de passe a une place importante dans le processus. Tout d'abord il va être stocké dans le fichier .ds pour pouvoir être comparé avec le hash du mot de passe entré au moment du déchiffrement. Bien évidemment, si on lis le fichier .ds on accède au hash mais cela ne nous permet pas de déchiffrer le fichier ou de retrouver le mot de passe d'origine. Le mot de passe sera également utilisé pour brouiller la clé privé qui est visible dans le .ds

## **Les fonctions codées:**

Les fonctions propres au projet sont fournies en annexe.

### **FONCTIONS :**

**hash password** : hash le mot de passe

**valid\_passeword** : vérifie si l'utilisateur a entré le bon mot de passe pour le déchiffrement

**cyclone** : prend en paramètre la clé privé et le mot de passe mot pour retourner une clé non utilisable. le concept est inspiré de césar cette fonction nous permet de transférer la clé privé qui est

indispensable pour le déchiffrement sans pour autant compromettre la sécurité du logiciel

**antyclone** : permet de retrouver la clé privée avec le mot de passe et la clé brouillé

**is\_prime** : vérifie si un nombre entier est premier

**generate\_random\_prime** : génère aléatoirement un nombre premier

**generate\_two\_distinct\_prime** : génère deux nombres premiers aléatoirement et différents

**gcd** : plus grand diviseur commun entre deux nombres

les fonction **find\_e**, **mod\_inverse**, **mod\_exp** suivent le processus de création de clé publique et privé RSA

Le type d'entier est fixé à i128 pour pouvoir utiliser de grands chiffres afin d'exploiter au maximum l'algorithme RSA.

**get\_file\_name\_and\_extension** : prend le chemin d'un fichier et retourne son nom et son extension

**read\_file\_to\_string** : lis un fichier et retourne le contenu sous forme de string

**create\_ds\_file** : crée le fichier .ds et regroupe tous les éléments à mettre à l'intérieur.

```
let content: String = format!(
    "DATA_SHIELDER///{}///{}///{}///{}///{}///{}///{}",
    file_name, hashed_password, extension, constant, private_key.0, private_key.1, encrypted_message_joined
);
```

**builder** : prend en paramètre le mot de passe l'input du fichier à crypter, le mode de cryptage, le chemin de sorti  
c'est cette fonction qui fera le lien avec l'interface graphique

**collector** : elle collecte le fichier . ds , le mot de passe et vérifie si le fichier est sous le bon format

**recreate\_file** : retourne le fichier originale

**encrypt** : chiffre le message avec la clé publique.

**decrypt** : cette fonction décrypte le message crypté avec la clé privé

**process** : génère une clé publique et privé RSA

## **Chiffrement ECC:**

### **Fonctionnement**

On commence par choisir une courbe elliptique et un point de base G sur cette courbe. Le point de base est un point fixe utilisé pour les calculs.

Une courbe elliptique est une courbe définie par une équation mathématique de la forme  $y^2=x^3+ax+b$ . où a et b sont des constantes qui déterminent la forme de la courbe. Ces courbes ont des propriétés mathématiques spéciales qui les rendent utiles pour le chiffrement.

Ici, nous avons choisi d'utiliser la courbe elliptique associée à la fonction X25519 Diffie-Hellman de forme :  
 $v^2=u^3+486662u^2+u$

### **Implémentation**

Comme nous l'avions prévu, nous avons implémenté une seconde méthode de chiffrement de fichier par la méthode d'utilisation des courbes elliptiques.

Pour cela nous avons implémenté les fonctions nécessaires à la génération d'une clé ECC (Elliptic Curve Cryptography) : nous avons utilisé la librairie rust-crypto pour les opérations de courbes elliptiques et la librairie aes-gcm pour le chiffrement des données.

AES-GCM (Advanced Encryption Standard - Galois/Counter Mode) est un mode de fonctionnement pour le chiffrement par bloc qui relie les avantages de l'AES avec les techniques d'authentification pour plus de sécurité durant le chiffrement.

La clé privée est générée à l'aide d'une fonction auxiliaire (voir **generate\_private\_key** en annexe) qui la crée en utilisant une valeur fixe. Dans un meilleur cas, la clé privée serait générée aléatoirement à chaque chiffrement pour garantir plus de sécurité. La clé privée obtenue est ensuite multipliée par le point de base de la courbe elliptique ( ici, le point de base de la courbe elliptique de la) pour générer une clé publique.

Après génération des clés publique et privée, le programme créera ensuite un “secret partagé”, obtenu en multipliant les clé publique et privée, convertie en octets pour l'utiliser comme clé AES.

Pour chaque opération de chiffrement, un nonce aléatoire de 12 octets est généré. Le nonce est essentiel pour garantir que chaque texte chiffré est unique, ce qui empêche les attaques par texte chiffré identique.

Le contenu du fichier, combiné a est ensuite chiffré à l'aide de la clé AES. Le résultat du chiffrement, comprenant le nonce et le texte chiffré, est ensuite combiné et encodé en base64 pour produire une chaîne de caractères. L'encodage en base64 facilite le stockage et le transfert du texte chiffré sous forme de chaîne de caractères.

## **Site web :**

Le site web de Data Shielder est une composante essentielle de notre projet, conçue pour être le visage accueillant de notre solution de cybersécurité. À travers une interface intuitive et une esthétique moderne, nous avons cherché à offrir une expérience utilisateur fluide et engageante. Voici une présentation détaillée de notre site web, de ses fonctionnalités, et de la vision qui a guidé sa conception.

### **Conception et Design**

Dès le départ, nous avons voulu que le design du site reflète les valeurs fondamentales de Data Shielder : sécurité, fiabilité et simplicité. Notre palette de couleurs est sobre, avec des teintes de bleu et de blanc qui évoquent la confiance et la clarté. Le logo de Data Shielder, visible en haut de chaque page, symbolise la protection et la robustesse de notre solution de chiffrement.

### **Accueil**

La page d'accueil est la première impression que les visiteurs auront de notre site. Elle est conçue pour être à la fois informative et attrayante. Dès leur arrivée, les utilisateurs sont accueillis par une bannière accrocheuse présentant notre logiciel et ses

principales fonctionnalités. Un bouton "Commencer" invite les visiteurs à explorer davantage le site.

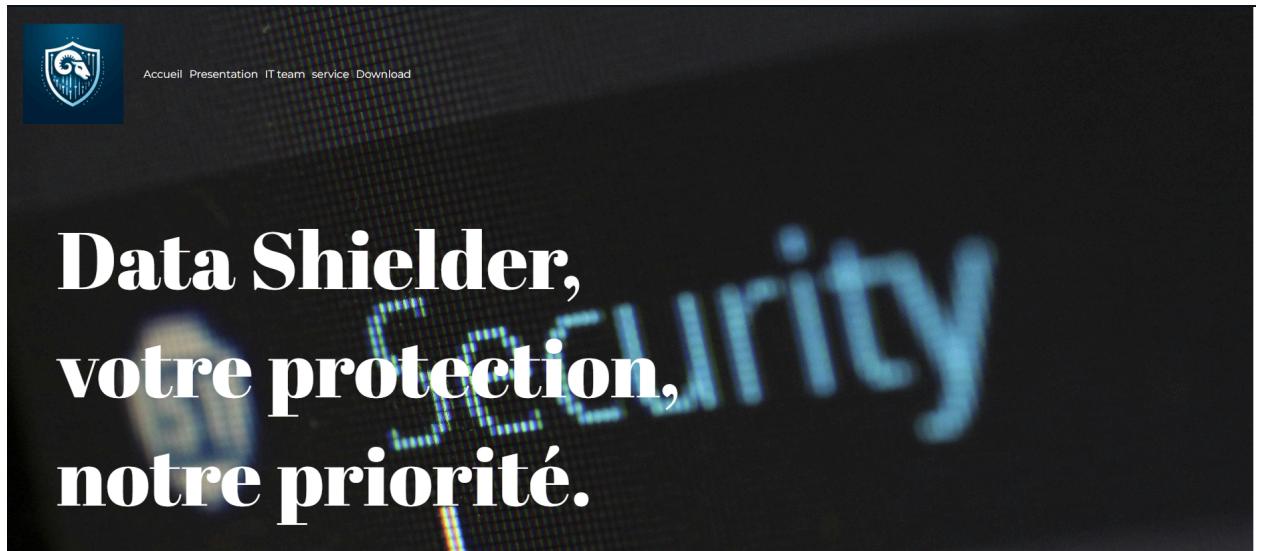
Sous la bannière, nous avons intégré une section de présentation de notre projet. Elle explique brièvement le but de Data Shielder, la problématique de la cybersécurité que nous cherchons à résoudre, et les avantages de notre solution. Cette section est accompagnée de graphiques et d'illustrations pour rendre l'information plus accessible et engageante.

## Fonctionnalités

La page des fonctionnalités est dédiée à une description détaillée des différentes capacités de Data Shielder. Nous avons divisé cette page en plusieurs sections, chacune mettant en lumière un aspect spécifique du logiciel :

- **Gestionnaire de mots de passe** : Explication de la manière dont notre gestionnaire de mots de passe sécurise les informations sensibles.
- **Chiffrement RSA** : Détails techniques sur l'utilisation de l'algorithme RSA, accompagné d'exemples et d'illustrations pour aider à comprendre son fonctionnement.
- **Chiffrement par courbes elliptiques (ECC)** : Présentation de cette méthode de chiffrement avancée, ses avantages et son application dans notre logiciel.

- **Interface utilisateur** : Aperçu de notre interface graphique, montrant comment elle permet une utilisation intuitive et efficace de notre logiciel.



#### Presentation

DataShielder est un logiciel de chiffrement de fichier, il intègre des algorithme tel que le RSA. Le fichier chiffrer est généré sous l'extension .ds pour une sécurité maximale

®  
logiciel codé  
en rust

⌚  
Leger et  
rapide

🛡️  
garantie une  
protection  
prétiquement  
inviolable

#### Les Developpeurs épitéens



Mustapha Oumeziane (chef de projet)  
Diriger ce projet fut pour moi une expérience enrichissante



Evan  
Je me suis chargé d'implémenter les algorithmes.



Enzo Fernandez  
Je suis responsable de l'interface graphique et du design

#### Télécharger le projet

## **Interface Utilisateur :**

### Première soutenance:

Lors de la première soutenance nous n'avions pas commencé à concevoir d'interface mais nous nous sommes rapidement aperçus qu'elle allait être indispensable.

### Deuxième soutenance:

Pour la deuxième soutenance nous nous sommes réorientés vers l'utilisation de Iced et nous avons abandonné l'idée d'utiliser GTK.

Iced est une bibliothèque graphique multiplateforme pour Rust et est plutôt bien documentée notamment avec des git ou des vidéos youtube.

Le but principal pour cette 2e soutenance est de vous proposer une interface graphique qui fait entrer en jeu nos premières fonctions de chiffrement, déchiffrement et hachage dans une interface utilisateur ou du moins une première version qui se doit d'être agréable et fluide.

Nous avons développé une interface utilisateur intuitive, principalement centrée sur les fonctions de chiffrement et de déchiffrement. De plus, notre interface propose une fonctionnalité supplémentaire permettant à l'utilisateur de sécuriser ses fichiers chiffrés en saisissant un mot de passe unique. Ce mot de passe est utilisé pour générer les clés privées et publiques nécessaires au chiffrement du fichier. Par ailleurs, plusieurs modes de

chiffrement sont proposés, offrant à l'utilisateur la possibilité de choisir entre des algorithmes symétriques et asymétriques. Bien que la fonctionnalité de sélection des modes de chiffrement ne soit pas encore opérationnelle, elle sera implémentée pour notre dernière présentation. En résumé, l'ensemble du processus de chiffrement dépend du mot de passe saisi par l'utilisateur.

Lors du lancement de l'application, l'utilisateur rencontrera d'abord une page d'accueil l'invitant à commencer l'utilisation.



Bienvenu sur DataShield !

Start

En cliquant sur le bouton 'start', l'interface changera vers l'interface principale contenant les fonctionnalités de l'application.

Voici un aperçu de la page principale de notre application:



Le premier élément de l'interface est le bouton pour sélectionner un fichier. Comme son nom l'indique, cliquer sur ce bouton invitera l'utilisateur à choisir un fichier à chiffrer/déchiffrer.

Le deuxième élément est un champ dédié au mot de passe.

Le mot de passe a une importance cruciale dans notre application, étant donné que nous n'avons actuellement implémenté qu'un seul algorithme, le RSA original. Ce dernier englobe la génération de clés, ainsi que le chiffrement et le déchiffrement, reposant sur des opérations mathématiques impliquant des grands nombres premiers et de l'exponentiation modulaire.

Notre approche de gestion du mot de passe vise à faciliter la génération de clés publiques et privées. Pour rappel, cette génération repose sur la sélection de deux grands nombres premiers entre eux. Pour choisir ces deux nombres, nous avons mis en place un fonction de hachage. Cette dernière prend en entrée le mot de passe de l'utilisateur, le hache, puis retourne une chaîne de caractères représentant le résultat de ce processus.

```
// Function for hashing a password without using an external library
pub fn hash_password(password: &str) -> String {
    // Calculate the SHA-256 hash
    let hashed = sha256(password.as_bytes());

    // Convert the hash to a hexadecimal string
    let mut hashed_hex = String::new();
    for byte in hashed {
        write!(hashed_hex, "{:02x}", byte).expect("Failed to write to string");
    }

    // Truncate the hash string so that it is exactly 30 characters long
    if hashed_hex.len() > 30 {
        hashed_hex[..30].to_string()
    } else {
        hashed_hex
    }
}
```

Une fois ce hachage effectué, nous avons implémenté une fonction qui prend en paramètre ce message haché, et qui retourne un couple de nombres entier et premiers entre eux:

```

pub fn hash_to_primes(hash: &str) -> (i128, i128) {
    // Convert the first digits of the hash into a number
    let num = i128::from_str_radix(&hash[0..8], 16).unwrap_or(0);

    let mut prime1 = num;
    // Find the first prime number after 'num'
    while !is_prime(prime1) {
        prime1 += 1;
    }

    let mut prime2 = prime1 + 1;
    // Find the next prime after 'prime1'
    while !is_prime(prime2) {
        prime2 += 1;
    }
}

```

Grâce à ces deux grands nombres nous pouvons générer les deux clés requises pour chiffrer le fichier.

Lorsque l'utilisateur appuie sur le bouton 'Chiffrer', qui déclenche la fonction de chiffrement avec la clé publique, plusieurs vérifications sont effectuées. Tout d'abord, le système vérifie si le mot de passe a été saisi et si un fichier a été sélectionné. Si l'une de ces conditions n'est pas remplie, un message d'alerte est affiché pour informer l'utilisateur de l'erreur.

Sélectionner un fichier

Entrez votre mot de passe

Chiffrer

Déchiffrer

Sauvegarder le fichier chiffré

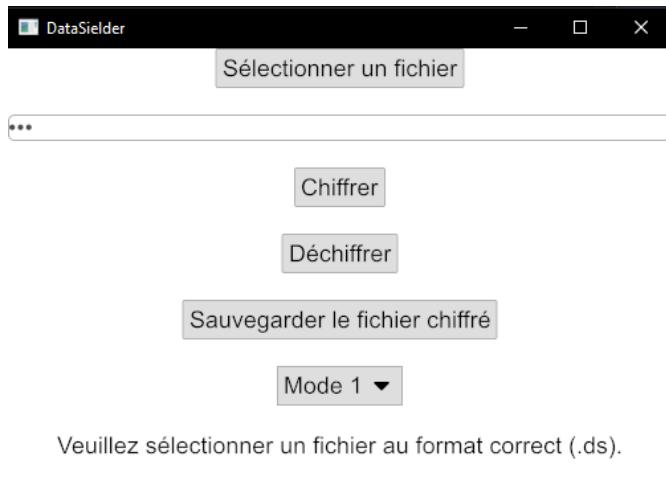
Mode 1 ▾

Veuillez entrer un mot de passe avant de chiffrer.

Le bouton 'sauvegarder' invite l'utilisateur à sauvegarder le fichier chiffré. Ce dernier sera automatiquement enregistré en

format .ds, une extension propre à notre application qui sera développée lors de la dernière soutenance.

Le bouton ‘déchiffrer’ ne peut accepter que les fichiers sous format .ds sinon un message d’avertissement se lance:

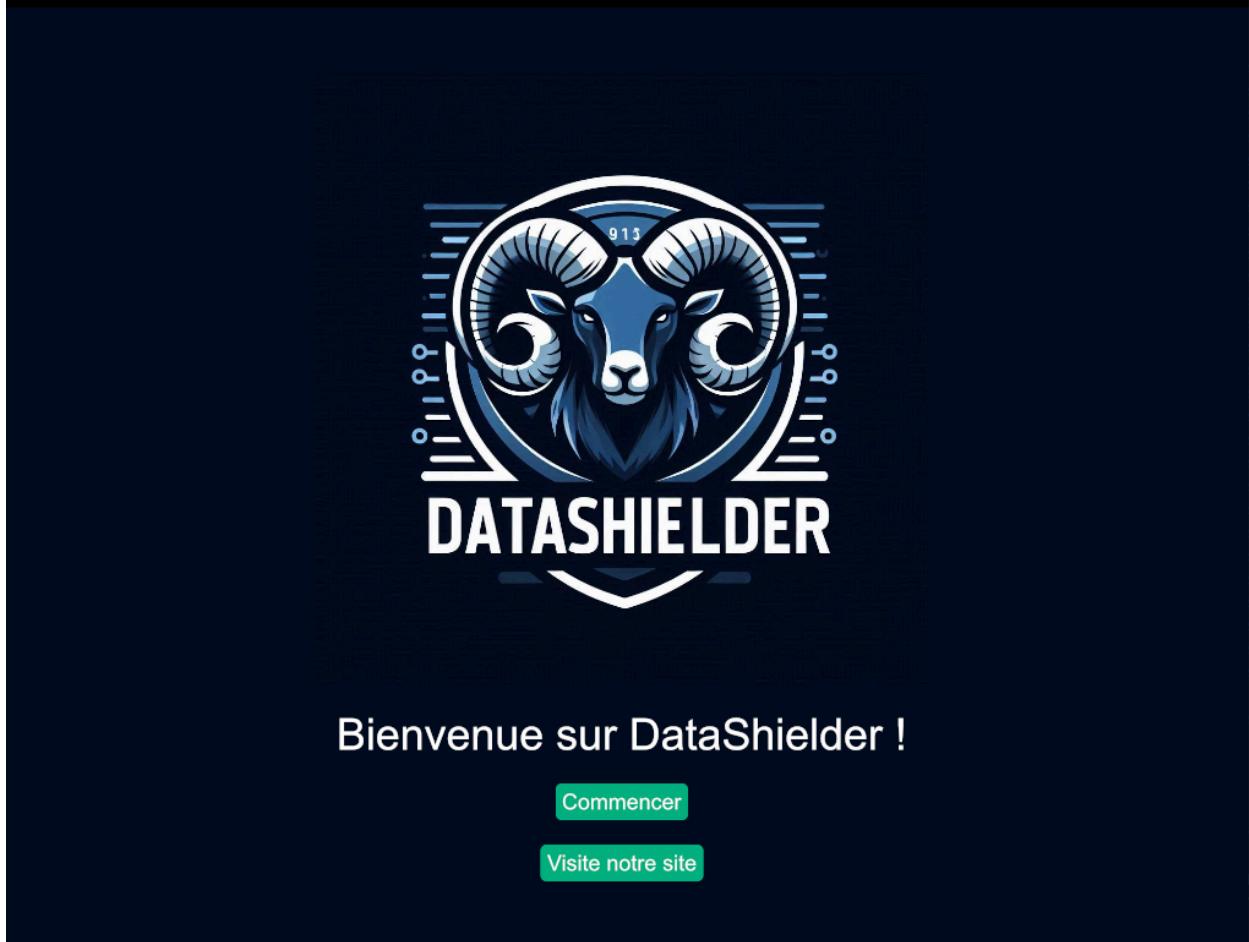


Il y a également un menu déroulant contenant deux options , mode 1 et mode 2 qui plus tard représenteront les deux méthodes de chiffrage. Nous ne savons pas encore si on laisse le choix à l'utilisateur entre un algorithme symétrique ou asymétrique mais dans tous les cas il aura le choix entre différents algorithmes.

### Troisième soutenance:

Enfin lors de cette troisième soutenance nous sommes repartis sur la base que nous avons créé pour la développer et la rendre plus attractive. Nous avons gardé les fonctionnalités principales telles que chiffrer, déchiffrer et sauvegarder et nous avons implémenté les modes pour proposer plus d'options.

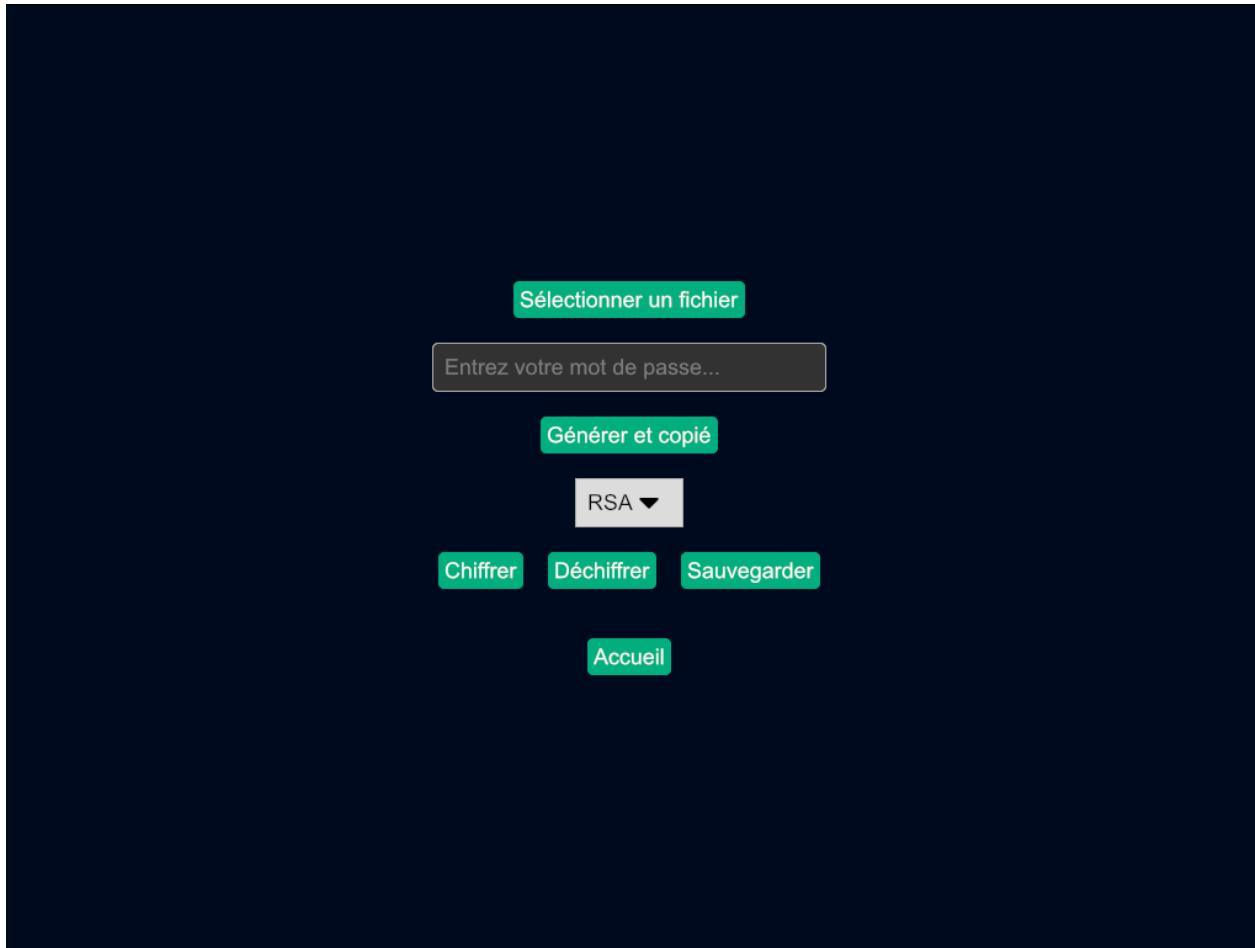
Premièrement, la page d'intro, nous avons décidé de partir sur un thème plutôt sombre car ça fait beaucoup plus propre plutot que de rester sur un theme blanc plutot basique.



Voici notre page d'intro, lorsqu'on lance l'application on tombe sur cette page. Nous avons ajouté la possibilité de visiter notre site web pour proposer plus de détails et d'informations sur notre application.

Un fond sombre qui colle parfaitement avec l'image et un style de bouton plutôt opposé avec des couleurs froides pour contraster ce côté sombre.

Une fois être prêt à cliquer sur le bouton commencer on tombe sur cette page:



Une page très simple, toujours avec un thème sombre et des boutons qui contrastent. La barre de saisie du mot de passe est maintenant beaucoup plus jolie et attrayante que précédemment, vous pouvez voir que nous avons implémenté plusieurs boutons, d'abord une nouvelle fonctionnalité, la possibilité de générer un mot de passe aléatoirement si l'utilisateur veut quelque chose de sécurisé, il s'agit d'un mot de passe à 12 caractères voici la fonction associée:

```
fn generate_random_password(length: usize) -> String {
    const CHARSET: &[u8] = b"ABCDEFGHIJKLMNOPQRSTUVWXYZ\
                           abcdefghijklmnopqrstuvwxyz\
                           0123456789)(*^%$#@!~";
    let mut rng = rand::thread_rng();

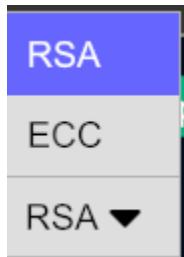
    (0..length)
        .map(|_| {
            let idx = rng.gen_range(0..CHARSET.len());
            CHARSET[idx] as char
        })
        .collect()
}
```

Une fonction assez simple qui permet de choisir 12 caractères parmi ceux proposés. Une fois le mot de passe généré, il est directement copié dans le presse papier de l'utilisateur ce qui lui permet de pouvoir le réutiliser.



Il est directement mis dans la barre de saisie du mot de passe et s'affiche en dessous au cas où il ne s'est pas copié.

Juste en dessous nous avons le menu déroulant permettant de changer de mode de chiffrement si nous le souhaitons:

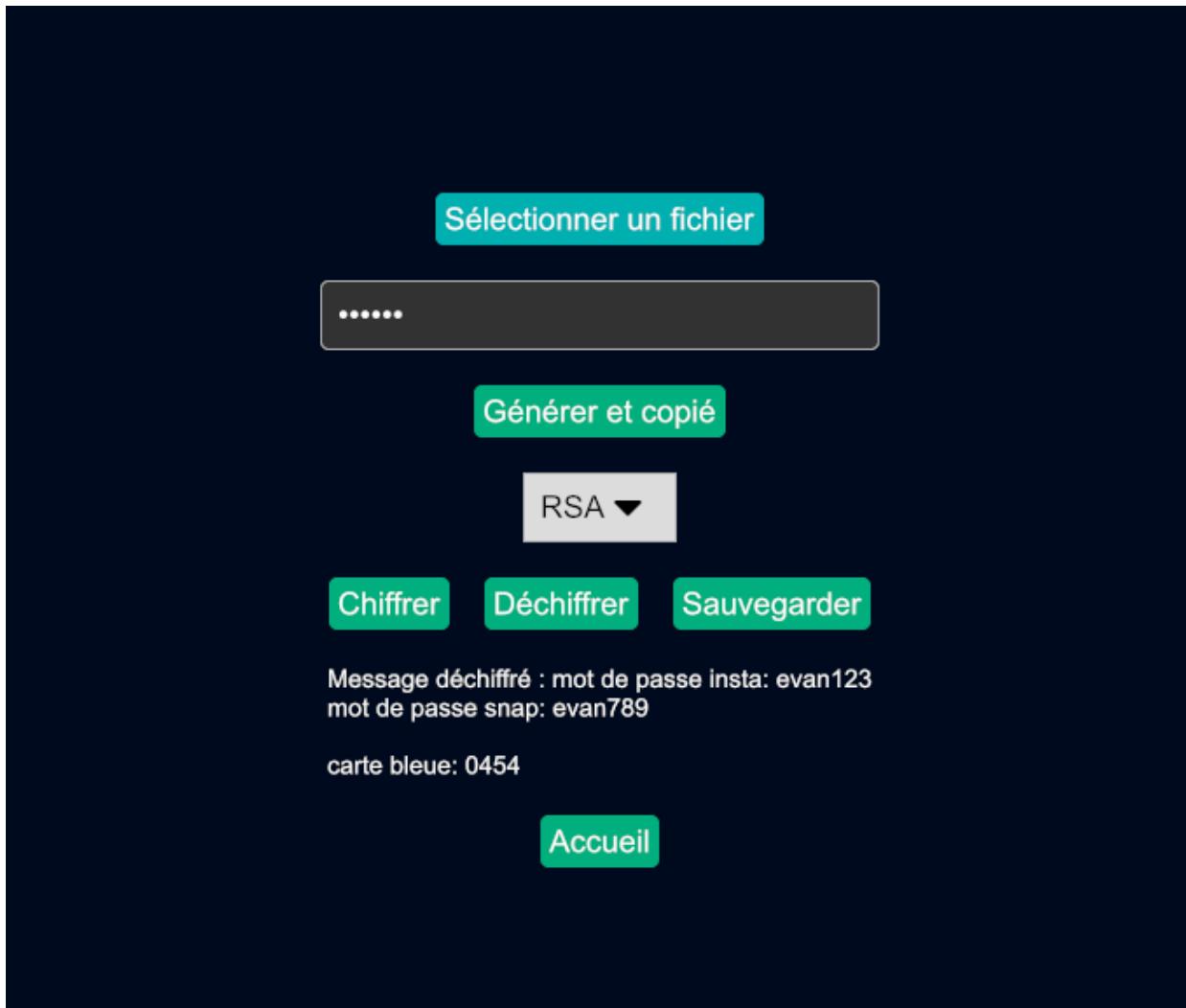


Nous avons implémenté 2 modes, le chiffrement RSA et le chiffrement ECC (nous ne rentrerons pas en détail sur leur fonctionnement ici puisqu'ils ont été expliqués plus haut dans le rapport).

Nous avons beaucoup plus guidé l'utilisateur en mettant des messages pour lui indiquer ce qu'il doit faire si cela ne fonctionne pas par exemple si il veut chiffrer un fichier sans avoir entré de mot de passe, un message va le guider, si il veut déchiffrer un fichier et qu'il n'est pas en .ds il le guide etc.

Revenons à notre barre de mot de passe, désormais elle est réellement fonctionnelle, avant même si le mot de passe n'était pas le bon, on pouvait déchiffrer notre fichier ce qui posait problème puisque le fichier de sortie ne ressemblait à rien. Maintenant si le mot de passe n'est pas le bon, il est strictement impossible pour l'utilisateur de déchiffrer son fichier.

Si le contenu du fichier déchiffré n'est pas trop long, il est affiché:



Un message vous indique si votre fichier a été chiffré ou déchiffré, on a maintenant la possibilité de revenir à la page d'accueil.

Une fois qu'un fichier a été chiffré voici un exemple de comment il ressort:

```
DATA_SHIELDER///message///91a73fd806ab2c005c13b4dc19130a884e909dea3f72d46e30266fe1a1f588d8
///txt///Mode 1///68259570131747077///73947868188116561
///39313162803115704///4598173355879026///31865077226682774///67449789737055854///61144781014308853///
///20897799137208962///64149805649321093///47339948526210473///47339948526210473///12765264922050054///
///27695398225041652///47339948526210473///31865077226682774///64149805649321093///7604488536574808///
///67571370919887914///64149805649321093///27695398225041652///20314770921236382///12960978470576364///
///10000000000000///39313162803115704///4598173355879026///31865077226682774///67449789737055854///61:
///67449789737055854///20897799137208962///64149805649321093///47339948526210473///47339948526210473///
///47339948526210473///27695398225041652///64149805649321093///20897799137208962///7604488536574808///
///67571370919887914///64149805649321093///27695398225041652///53839450838885768///55734257323311628///
///10000000000000///302875106592253///10000000000000///66867654691718806///64149805649321093///470454:
///12765264922050054///67449789737055854///35899963506179094///4862166185221929///12765264922050054///
///7604488536574808///67449789737055854///2363378469454230///71443494727724449///47275911585650678///
```

Comme expliqué plus haut dans le rapport, si le fichier .ds ne commence pas par DATA\_SHIELDER il ne pourra pas être déchiffré ce qui ajoute une sécurité. L'utilisateur peut s'il le souhaite retirer ce tag temporairement comme ça si il se fait voler son fichier le voleur ne pourra pas le déchiffrer, il ne faudra pas qu'il oublie de le remettre s'il veut pouvoir déchiffrer son fichier. Dans ce fichier est indiqué dans l'ordre: le nom du fichier avant d'être chiffré, le mot de passe haché, le type du fichier avant d'être chiffré, le mode utilisé pour le chiffrer et la clé privée, tout le reste est le contenu chiffré.

Pour le chiffrement ECC la méthode est un peu différent, le fichier de sortie ne possède pas toute cette sécurité mais uniquement le contenu chiffré, voici un exemple de fichier en sortie:

---

UhtCR4pF8UBAi3yi8T3odNAJhGNR/iUT9RrpW4ps0VKW  
3F4CxzXp0kbJt2NM12CoHgGlxFdiuagjMCfmhrAMUfXX  
1km348wh2PYHZCPzAga2nC/C7pj3SnLg5/s/qHeJvn+Zp  
/o0Qks=f" h5spjhPgEzqP8oiU1ijio5ksjkn,xS|

### **III - Etat du projet**

**Tableau récapitulatif :**

Membre impliqué	Tâches	Réalisation (en %)
Mustapha	Gestionnaire de mots de passe	100%
	RSA	100%
	extension	100%
	Site web	75%
Enzo	Interface graphique	100%
	Chiffrement par courbe elliptique	90%
Evan	Fonctionnalités de l'interface utilisateur	90%
	Algorithmes de chiffrement, déchiffrement, signature cryptographiques	80%

## **IV - Ressentis Positifs**

### **Evan:**

Ce projet m'a beaucoup apporté sur la gestion des ressources pour en créer quelque chose de concret, étant chargé de créer l'application, j'ai pu faire parler ma créativité pour rendre l'application simple d'utilisation et attractive pour qu'elle puisse plaire au grand public. J'ai développé mes compétences en Rust et notamment en iced, je suis maintenant capable de gérer une application, ce qui devrait m'apporter beaucoup pour les futures années à EPITA.

J'ai adoré voir l'évolution de l'application au fur et à mesure, j'ai pu voir ce que je devais améliorer ou refaire, quand je vois l'évolution de notre application depuis la deuxième soutenance je ne peux qu'être fier de ce qu'on a réalisé.

J'ai adoré la liberté offerte lors de ce projet, aucune restriction particulière on était libre de faire ce qu'on voulait que ce soit un jeu, une application, de la 3D, de la 2D, on était pas bloqué par des consignes strictes.

### **Enzo:**

Ce projet a été l'occasion pour moi de découvrir un nouvel aspect de la programmation : le chiffrement de données. J'ai eu

donc l'occasion de découvrir de nouveaux outils et méthodes au travers des documentations qu'il m'a été nécessaire de lire et comprendre afin d' accomplir les objectifs que nous nous étions fixés. Pouvoir étendre mes connaissances du rust a été une expérience réellement enrichissante et nouvelle dans le domaine de la cryptographie.

### **Mustapha:**

Étant le meneur du groupe, je suis satisfait du travail accompli car notre premier objectif à été rempli. Mon moment préféré est la définition des propriétés de l'extension car c'est comme si je crée un univers numérique régi selon des caractères bien spécifiques choisis au préalable. Je suis content que l'extension fonctionne

## **V - Ressentis Négatifs**

### **Evan:**

Je suis un peu déçu de la communication du groupe, je pense qu'avec plus de communication et d'entraide nous aurions pu aller plus loin dans le projet, malgré tout je suis quand même content de ce qu'on a pu faire mais un peu déçu de ne pas avoir réussi à tout faire. J'aurai voulu développer d'autres modes de chiffrement comme du chiffrement symétrique.

Les plus grosses difficultés que j'ai personnellement rencontrées lors de ce projet sont notamment les contraintes de temps et les imprévus. En effet, dès le début, j'ai éprouvé des difficultés à démarrer concrètement le travail qui m'était attribué, ce qui a entraîné à de nombreuses reprises un sentiment d'urgence et de pression temporelle. Il était essentiel de trouver un équilibre entre la qualité du travail que je devais accomplir et le respect des délais impartis.

Face à ces contraintes de temps, j'ai dû faire preuve d'une grande capacité d'adaptation et de réactivité. Il était nécessaire de prioriser à nouveau les tâches, de planifier efficacement mon emploi du temps et de m'organiser de manière stratégique.

Malgré les obstacles rencontrés, j'ai su prendre les mesures nécessaires pour accélérer la cadence de travail sans compromettre la qualité du résultat final.

En parallèle des contraintes de temps, les imprévus ont également été une source de difficulté. Il est courant dans tout projet informatique de rencontrer des problèmes techniques inattendus, des bugs ou des complications qui ralentissent la progression du travail. Ces imprévus peuvent engendrer des frustrations et perturber le flux de travail, nécessitant une résolution rapide et efficace.

### **Enzo:**

Durant cette dernière période du projet, la communication dans le groupe a laissé à désirer. Cependant cela ne m'a pas empêché de mener à bien la création de la seconde méthode de chiffrement dont j'étais chargé.

Néanmoins, ce manque de communication a entraîné un retard notoire au projet : j'ai du finalisé la fonction de chiffrement assez rapidement sans que le résultat me satisfasse complètement. Les ajustements nécessaires ont ensuite été réalisés mais cela a eu un impact sur la fluidité du travail sur le projet.

Je suis convaincu que par une meilleure cohésion et entente, nous aurions pu aller encore plus loin dans le projet, en ajoutant diverses méthodes ou optimisations intéressantes. Voir ce potentiel gâché me frustre.

## **Mustapha:**

Pour cette dernière soutenance, les problèmes majeurs ont été au niveau de la communication et de l'implication de certains membres. Le projet doit être fait à 4 mais nous sommes 3. Par conséquent, des mesures doivent être prises pour ne pas être en manque de temps et d'effort. Malgré cela, la communication n'était pas au rendez vous! En tant que chef de groupe j'ai pris la décision pour cette dernière soutenance de tout revoir depuis le début, impliquant l'architecture du programme, l'erreur commune est que nous ne nous sommes même pas posé la question de comment et dans quel ordre agencer les fonctions entre elles pour que tout fonctionne. Résultat, des problèmes entre les inputs et les output. Le fait étant que la base nouvelle vient de la partie de l'extension sous RSA, les parties suivantes devraient se calquer sur ce modèle pour pouvoir fonctionner en harmonie cependant certains membres ne se sont même pas préoccupé de la question et on simplement fait comme bon leur semble sans prendre en compte les décisions déjà prises. Le manque de considération et de présence à l'égard de ce projet nous ont malheureusement fait défaut.

## **VI - Problèmes rencontrés**

Evan :

Durant l'ensemble du projet, j'ai eu beaucoup de problèmes de compile, surtout deux qui revenaient tout le temps et j'ai eu énormément de mal à les réparer:

***note: run with `RUST\_BACKTRACE=1` environment variable to display a backtrace error: process didn't exit successfully: `C:\Users\source\repos\S4project\target\debug\Da***

***xe` (exit code: 0xc000041d)***

***C:\Users\source\repos\S4project\target\debug\deps\libscope***

***guard-e641b1ee7a986764.rmeta: Le processus ne peut pas accéder au fichier car ce fichier est utilisé par un autre processus. (os error 32)***

Ces deux erreurs m'ont pas mal retardé sur l'avancement de l'interface utilisateur.

Je n'ai pu utiliser que la version 0.3 de iced car les dépendances des autres versions ne fonctionnaient pas sur mon pc ce qui m'a pas mal limité pour l'innovation de l'application.

J'ai également rencontré des problèmes pour associer les fonctions aux boutons présents dans l'application, beaucoup ne fonctionnaient pas comme je le voulais j'ai dû plusieurs fois contourner le problèmes mais ça nous a fait perdre pas mal de temps.

Des import qui ne fonctionnaient pas j'ai donc dû mettre toutes les fonctions dans la main au lieu de les ranger proprement et de les appeler ce qui était très désorganisé.

Implémenter les 2 modes de chiffrement a été très dur j'y ai passé plusieurs jours pour que le résultat ne soit celui convenu, la difficulté étant au niveau du .ds, certe beaucoup plus sécurisé mais beaucoup plus dur à implémenter pour un algorithme qui ne prend pas de clé privée.

Enzo :

Tout d'abord j'ai eu des problèmes concernant l'exécution du projet sur ma machine personnelle, notamment obtenant le message d'erreur "exit code: 0xc0000005, STATUS\_ACCESS\_VIOLATION". J'ai pu savoir comment cette erreur est arrivée mais j'ai eu du mal à la résoudre avant de finalement me concentrer sur la création de la fonction de chiffrement par courbe elliptique.

La création et les ajustements nécessaires à l'implémentation de cette nouvelle méthode de chiffrement a été un véritable défi : la documentation n'était pas souvent claire et j'ai dû construire plusieurs prototypes avant d'arriver à une version fonctionnelle qui satisfait les attentes.

Mustapha :

D'un point de vu technique il y a eu des erreurs à corriger : la structure du projet qui n'est pas assez lisse et organisée car la pédagogie d'organisation des dossiers a été ignorée. L'image en background, la similarité du contenu de l'extension avec les courbes elliptique qui doit être similaire avec celle du RSA mais puisque le responsable de cette partie ne s'est pas informé sur les enjeux et bien l'extension sous ECC n'a que de contenu le message chiffré et ne respecte pas les propriété de l'extension défini en amont, le site web n'a pas de responsive car considéré comme secondaire en ordre de priorité. Cependant la fonctionnalité de base de notre logiciel c'est à dire le chiffrement de fichier txt est accomplie, j'aurais vraiment aimé aller plus loin en implémentant correctement avec les autres membres du groupe la partie ECC. Nous aurions pu éviter ce genre de problème si à la base nous nous serions mis d'accord de manière plus précise la répartition des tâches. La question n'est pas qui fait quoi. La question est qui fait quoi et comment il le fait pour que tout fonctionne en harmonie et qu'il n'y ai pas de conflit. Néanmoins je prends la responsabilité et j'ai appris que la stratégie et la gestion de projet sont essentielles et c'est sur quoi il faut se pencher dès le début.

## **VII - Conclusion**

Ce projet, Data Shielder, a été une véritable aventure, un périple à travers les méandres de la cryptographie et du développement logiciel. Depuis nos premiers jours, l'objectif était clair : concevoir un logiciel capable de protéger les données sensibles des utilisateurs, offrant une solution de chiffrement fiable et intuitive. De janvier à juin, nous avons plongé tête baissée dans ce défi, naviguant entre les lignes de code, les concepts mathématiques et les défis humains.

Chaque membre de l'équipe a apporté sa contribution unique à ce projet. Mustapha, en tant que chef de projet, a su guider l'équipe avec détermination, fixant des objectifs clairs et veillant à ce que chacun trouve sa place dans le processus. Enzo s'est plongé dans le développement de l'interface utilisateur, créant une expérience fluide et ergonomique pour les utilisateurs. Evan, quant à lui, s'est attaqué à la complexité de la cryptographie, jonglant avec les algorithmes et les concepts mathématiques.

L'un des moments forts de ce projet a été l'apprentissage de la cryptographie. Comprendre les bases du chiffrement RSA et plonger dans les courbes elliptiques a été à la fois stimulant et gratifiant. Chaque concept maîtrisé nous rapprochait un peu plus de notre objectif, chaque algorithme implémenté nous rapprochait un peu plus de la sécurité des données.

Pourtant, le plus grand défi que nous ayons rencontré n'était pas technique, mais humain. La gestion de projet, la communication au sein de l'équipe, les différences de rythme et d'approche : autant de défis qui ont mis à l'épreuve notre patience et notre résilience. Mais c'est précisément dans ces moments de tension que nous avons appris le plus, en apprenant à écouter, à compromettre et à avancer ensemble malgré les obstacles.

Malgré les difficultés, nous sommes fiers du résultat que nous avons accompli. Data Shielder est bien plus qu'un simple logiciel : c'est le fruit de mois de travail acharné, de collaboration et de détermination. Son interface intuitive et ses fonctionnalités avancées en font un outil précieux pour quiconque cherche à protéger ses données.

Ce projet nous a non seulement permis d'acquérir des compétences techniques précieuses, mais il nous a également enseigné des leçons précieuses sur le travail d'équipe, la communication et la résolution de problèmes. En regardant en arrière, nous réalisons à quel point nous avons grandi en tant que développeurs et en tant que personnes.

## Annexe

lien du site web:

<https://demoklyr.github.io/datashielder.github.com/>

### fonctions:

```
pub fn cyclone(value: i128, password: &str) -> i128 {
    // Convertir le mot de passe en un entier en sommant les valeurs ASCII des caractères
    let shift_amount: u32 = password.bytes().map(|b: u8| b as u32).sum::<u32>() % 128;

    // Effectuer une rotation circulaire à droite
    (value >> shift_amount) | (value << (128 - shift_amount))
}

pub fn anticyclone(value: i128, password: &str) -> i128 {
    // Convertir le mot de passe en un entier en sommant les valeurs ASCII des caractères
    let shift_amount: u32 = password.bytes().map(|b: u8| b as u32).sum::<u32>() % 128;

    // Effectuer une rotation circulaire à gauche
    (value << shift_amount) | ((value as u128) >> (128 - shift_amount)) as i128
}
```

```

impl RSAKeys {
    fn new(max: i128) -> Self {
        let (p: i128, q: i128) = generate_two_distinct_primes(max);
        let phi: i128 = (p - 1) * (q - 1);
        let n: i128 = p * q;
        let e: i128 = find_e(phi);
        let d: i128 = mod_inverse(e, phi);
        RSAKeys {
            public_key: (e, n),
            private_key: (d, n),
        }
    }

    pub fn process() -> RSAKeys {
        let max: i128 = 1_000_000_000/*_000_000_000*/;
        let rsa_keys: RSAKeys = RSAKeys::new(max);
        return rsa_keys;
    }
}

```

```

pub fn builder(password: &str, input_path: &Path, moder: &str, output_path: &Path) {
    // Lire le contenu du fichier
    let message: String = read_file_to_string(file_path: input_path).expect(msg: "Failed to read file");
    println!("Message original: {:?}", message);
    // Générer les clés RSA
    let variables: RSAKeys = process();
    let mut private_key: (i128, i128) = variables.private_key;
    println!("Clé privée avant cyclone: {:?}", private_key);
    // Appliquer le cyclone sur la clé privée
    private_key.1 = cyclone(value: private_key.1, password);
    println!("Clé privée après cyclone: {:?}", private_key);
    // Hacher le mot de passe
    let hashed_password: String = hash_password(message: password);
    // Récupérer le nom de fichier et l'extension
    let (file_name_option: String, extension_option: Option<String>) = get_file_name_and_extension(file_path: input_path).unwrap();
    let file_name: String = file_name_option;
    let extension: String = extension_option.unwrap_or_else(|| "unknown".to_string());
    // Chiffrer le message
    let encrypted_message: Vec<i128> = encrypt(&message, variables.public_key);
    //println!("Message chiffré: {:?}", encrypted_message);
    // Créer le fichier .ds avec le contenu chiffré
    create_ds_file(
        &file_name,
        &extension,
        constant: moder,
        private_key,
        &hashed_password,
        encrypted_message,
        output_path,
    ).expect(msg: "Failed to create .ds file");
}
fn builder

```

```

pub fn create_ds_file(file_name: &str, extension: &str, constant: &str, private_key: (i128, i128), hashed_password: &str, encrypted_message: Vec<i128>,
let output_path: PathBuf = output_path.join(path: format!("{}.ds", file_name));
let mut output_file: File = File::create(&output_path)?;

let encrypted_message_str: Vec<String> = encrypted_message.iter().map(|&num: i128| num.to_string()).collect();
let encrypted_message_joined: String = encrypted_message_str.join(sep: "/");

let content: String = format!(
    "DATA_SHIELDER///{}///{}///{}///{}///{}///{}///{}",
    file_name, hashed_password, extension, constant, private_key.0, private_key.1, encrypted_message_joined
);

output_file.write_all(buf: content.as_bytes())?;
Ok(())
}

```

```

pub fn collector(path: &Path, password: &str) -> io::Result<(String, String, String, String, RSAKey, Vec<i128>)> {
let mut file: File = File::open(path)?;
let mut content: String = String::new();
file.read_to_string(buf: &mut content)?;
// Vérifier si le fichier commence par "DATA_SHIELDER"
if !content.starts_with("DATA_SHIELDER") {
    return Err(io::Error::new(kind: io::ErrorKind::InvalidData, error: "Invalid file format"));
}
// Séparer le contenu par "///"
let parts: Vec<&str> = content.split("///").collect();
println!("Parts from .ds file: {:?}", parts);
if parts.len() < 7 {
    return Err(io::Error::new(kind: io::ErrorKind::InvalidData, error: "Invalid file format"));
}
// Extraire les différentes parties
let original_file_name: String = parts[1].to_string();
let hashed_password: String = parts[2].to_string();
if !valid_password(password, hash: &hashed_password) {
    return Err(io::Error::new(kind: io::ErrorKind::InvalidData, error: "wrong password"));
}
let original_file_extension: String = parts[3].to_string();
let mode: String = parts[4].to_string();
let private_key_0: i128 = parts[5].parse().map_err(op: |_| io::Error::new(kind: io::ErrorKind::InvalidData, error: "Invalid private key part 0"))?;
let private_key_1: i128 = parts[6].parse().map_err(op: |_| io::Error::new(kind: io::ErrorKind::InvalidData, error: "Invalid private key part 1"))?;

let rsa_key: RSAKey = RSAKey {
    private_key: (private_key_0, anticyclone(value: private_key_1, password)),
};
// Extraire le message crypté
let encrypted_message: Vec<i128> = parts[7..] [&str]
    .iter()
    .map(|s: &str| s.parse().map_err(op: |_| io::Error::new(kind: io::ErrorKind::InvalidData, error: "Invalid encrypted message part")))
    .collect::<Result<Vec<i128>, io::Error>()?;
//println!("Message chiffré extrait : {:?}", encrypted_message);
Ok((original_file_name, hashed_password, original_file_extension, mode, rsa_key, encrypted_message))
} fn collector

```

```

pub fn recreate_file(
    file_name: &str,
    extension: &str,
    rsa_key: RSAKey,
    encrypted_message: Vec<i128>,
    _password: &str,
    output_dir: &Path,
) -> io::Result<()> [
    // Modifier la clé privée avec anticyclone
    println!("Clé privée avant anticyclone: {:?}", (rsa_key.private_key.0, rsa_key.private_key.1-1000));
    //anticyclone(rsa_key.private_key.1, password);
    //println!("Clé privée après anticyclone: {:?}", (rsa_key.private_key.0, private_key_1));

    // Décrypter le message
    let decrypted_message: String = decrypt(encrypted_message, private_key: (rsa_key.private_key.0, rsa_key.private_key.1));
    println!("Message décrypté: {:?}", decrypted_message);

    // Créer le chemin complet du fichier de sortie
    let output_path: PathBuf = output_dir.join(format!("{}.{}", file_name, extension));
    // Écrire le contenu décrypté dans le fichier
    let mut file: File = File::create(output_path)?;
    file.write_all(buf: decrypted_message.as_bytes())?;

    Ok(())
]

```

```

fn generate_private_key() -> Scalar {
    Scalar::from_bytes_mod_order([
        0x1d, 0x07, 0x56, 0x1a, 0x1f, 0xee, 0x62, 0x5f,
        0x4b, 0x00, 0xf9, 0x6d, 0x24, 0xe5, 0x08, 0x33,
        0x6b, 0xf6, 0xd5, 0x69, 0x46, 0x81, 0x2b, 0x30,
        0x74, 0x42, 0xf6, 0xb5, 0xa7, 0x9a, 0x1d, 0xf5
    ])
}

```

```
fn encrypt_file(input_path: &Path, private_key: &Scalar) -> Result<String, Box<dyn std::error::Error>> {
    // Génère une clé publique
    let public_key = private_key * &X25519_BASEPOINT;

    // Convertit la clé publique en array de 32 bytes
    let shared_secret = public_key.to_bytes();

    // Init AES Key
    let aes_key = Key::from_slice(&shared_secret);

    // Init AESGCM à partir de la AES key
    let cipher = Aes256Gcm::new(aes_key);

    // Lit le fichier en entrée
    let mut input_file = File::open(input_path)?;
    let mut buffer = Vec::new();
    input_file.read_to_end(&mut buffer)?;

    // Génère un unique nonce pour chaque chiffrement
    let mut nonce_bytes = [0u8; 12];
    OsRng.fill_bytes(&mut nonce_bytes);
    let nonce = Nonce::from_slice(&nonce_bytes);

    // Chiffrement
    let ciphertext = cipher.encrypt(nonce, buffer.as_ref())
        .expect("encryption failure!");

    // Combine nonce et ciphertext
    let mut result = nonce_bytes.to_vec();
    result.extend_from_slice(&ciphertext);

    // Encode en Base64 et retourne un string
    Ok(encode(&result))
}
```

```
fn decrypt_file(encrypted_data: &str, private_key: &Scalar) -> Result<String, Box<dyn std::error::Error>> {
    // Decode Base64 string to binary data
    let encrypted_bytes = decode(encrypted_data)?;

    // Split nonce and ciphertext
    let (nonce_bytes, ciphertext) = encrypted_bytes.split_at(12);

    // Génère une clé publique
    let public_key = private_key * &X25519_BASEPOINT;

    // Convertit la clé publique en array de 32 bytes
    let shared_secret = public_key.to_bytes();

    // Init AES Key
    let aes_key = Key::from_slice(&shared_secret);

    // Init AESGCM à partir de la AES key
    let cipher = Aes256Gcm::new(aes_key);
    let nonce = Nonce::from_slice(nonce_bytes);

    // Déchiffrement
    let plaintext = cipher.decrypt(nonce, ciphertext)
        .expect("decryption failure!");

    // Convert decrypted bytes to a string
    Ok(String::from_utf8(plaintext)?)
}
```