Rapport projet Sécurité des applications

Sermeus Steven et Frippiat Gabriel

Patz M. et Absil R.

Master 1 en Architecture des Systèmes Informatiques
Année académique 2023-2024

Table des matières

1 Introduction

Pour ce projet, nous avions comme objectif de développer une application permettant de partager des photos avec du chiffrement de bout en bout. L'application avait comme contrainte d'utiliser l'architecture client-serveur où le serveur ne peut pas être considéré comme une entité de confiance. Ce serveur permet d'enregistrer de nouveaux utilisateurs, d'authentifier des utilisateurs et d'autoriser des utilisateurs authentifiés d'effectuer des actions sur l'application.

Les fonctionnalités attendues étaient :

- La création d'un album photos
- L'upload de photos
- Le partage de photos
- Le partage d'album
- La création d'un compte
- S'authentifier sur l'application
- Pouvoir utiliser l'application sur plusieurs devices

Pour réaliser ce projet, nous avons utilisé différentes technologies qui sont présentées et détaillées dans la suite de ce rapport.

2 Architecture physique

Pour répondre à la problématique de l'énoncé tout en respectant les contraintes, nous avons utilisé une architecture client-serveur où le serveur ne peut pas être considéré comme un acteur de confiance. De ce fait, il a été nécessaire de mettre en place un chiffrement de bout en bout des informations sensibles. De plus, la sécurité des différents composants de l'application devait être assurée. Pour cela, nous avons utilisé différentes technologies pour chaque composant de l'architecture.

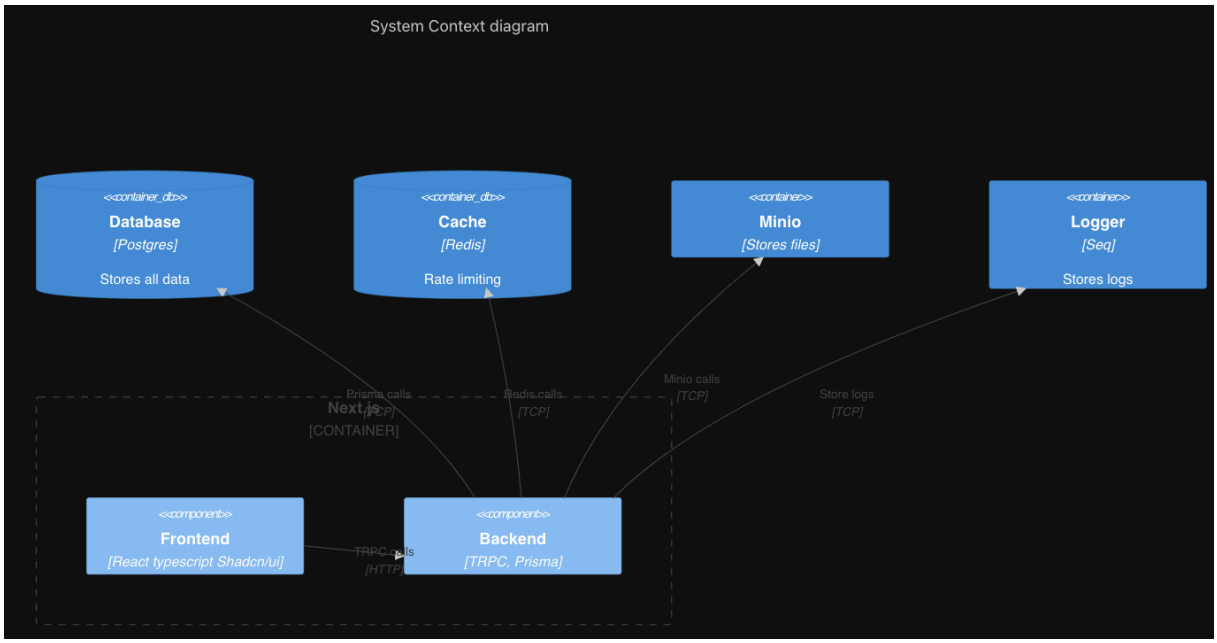

Figure 1 – Architecture physique

Comme le montre le schéma ci-dessus, l'architecture physique de notre application est composée de plusieurs éléments :

- **Base de données** : Celle-ci permet de stocker les informations des utilisateurs à long terme.
- **Reverse proxy** : Il permet d'assurer la connexion HTTPS entre le client et l'infrastructure. Nous avons utilisé Nginx, il assure également la connexion en HTTPS entre le serveur et le client.
- **Serveur de fichiers** : L'utilisateur pouvant être malveillant, nous avons décidé de ne pas stocker les fichiers sur le serveur principal. Nous avons délégué cette tâche à un serveur de fichiers dédié, Minio.
- **Serveur de logs** : Pour avoir un stockage de log fiable et sécurisé, nous avons utilisé un serveur de logs dédié, Seq.
- **Base de données clés valeurs** : Celle-ci permet d'effectuer des opérations de lecture et d'écriture rapides et nous a été utile pour effectuer du rate limiting sur les différentes routes

de l'application. Nous avons utilisé Valkey, qui est le fork open-source de Redis.
— **Next.js** : Il s'agit du coeur de l'application, il permet de gérer les différentes routes et de générer les pages côté serveur.

3 Cryptographie

Dans le cadre de ce projet, nous avons utilisé deux types de cryptographie : la cryptographie symétrique et la cryptographie asymétrique.

3.1 Cryptographie asymétrique

Celle-ci est utilisée pour l'authentification des utilisateurs ainsi que le partage de clés de chiffrement symétrique. Nous avons utilisé la librairie standard de JavaScript, `crypto`, pour générer des clés RSA et les utiliser pour chiffrer et déchiffrer des données. La librairie standard permet uniquement d'utiliser l'algorithme RSA pour de chiffrement asymétrique. Nous avons utilisé une clé de taille 4096, même si cette taille de clé est suspectée d'être "cassable". Pour le moment elle reste sécurisée et, en cas de besoin, il est toujours possible d'augmenter la taille de la clé.

3.1.1 Authentification

Lorsqu'un utilisateur s'inscrit sur notre application, un couple de clés RSA est généré. La clé publique de l'utilisateur est stockée dans la base de données et la clé privée est stockée dans le navigateur de l'utilisateur. Si l'utilisateur souhaite se connecter, il doit déchiffrer un challenge qui lui est envoyé par le serveur. Ce challenge est chiffré avec la clé publique de l'utilisateur. Si l'utilisateur parvient à déchiffrer le challenge, il est considéré comme authentifié étant donné que seul l'utilisateur possède la clé privée correspondante à la clé publique stockée dans la base de données permettant de déchiffrer le challenge.

3.1.2 Partage de clés de chiffrement symétrique

Un mécanisme similaire est utilisé pour le partage de clé de chiffrement symétrique. Avant leur envoi vers le serveur, elles sont chiffrées avec la clé publique de l'utilisateur destinataire. Ainsi seul l'utilisateur destinataire peut déchiffrer la clé de chiffrement symétrique.

3.2 Cryptographie symétrique

Nous avons choisi d'utiliser AES-GCM, un algorithme de chiffrement symétrique qui permet de chiffrer et d'authentifier les données. Tout comme pour la cryptographie asymétrique, nous avons utilisé la librairie standard de JavaScript, `crypto`, pour chiffrer et déchiffrer les données.

4 Hmac

Pour assurer que les logs n'aient pas été modifiés, nous avons utilisé un HMAC. Celui-ci est généré à partir du contenu du log et d'une clé secrète stockée sur le serveur. Le HMAC est stocké dans le log. Si le message est modifié, le HMAC ne correspondra plus au contenu du log et nous pourrons détecter qu'il y a eu une modification.

5 Fonctionnalités de l'application

5.1 Authentification

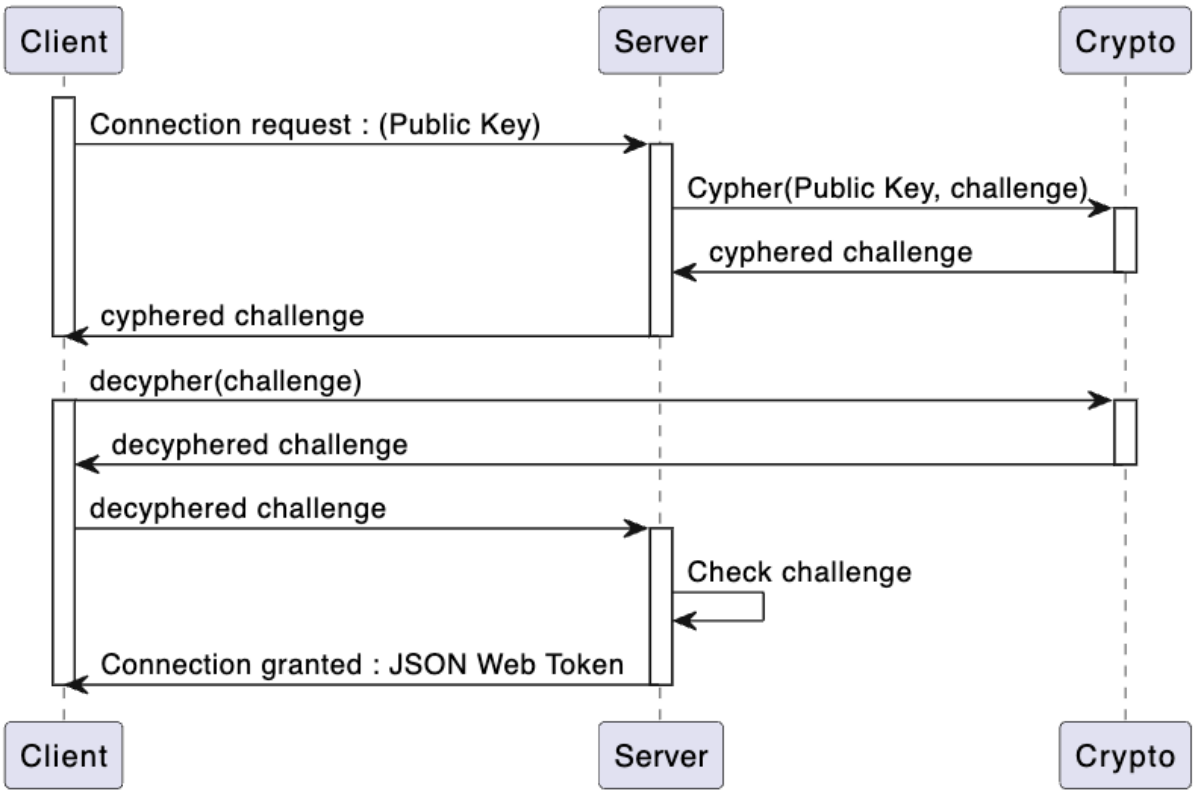

Figure 2 – Authentification

Comme le montre le diagramme ci-dessus, lorsqu'un utilisateur souhaite s'authentifier, il va envoyer au serveur la clé publique du device depuis lequel il essaye de se connecter. Lorsque le serveur reçoit cette clé publique, il vérifie d'abord si elle existe et va alors renvoyer à l'utilisateur un challenge si cette clé publique existe bien dans la base de données. Ce challenge est alors uniquement déchiffrable avec la clé privée correspondante. Une fois le challenge déchiffré au serveur, le serveur vérifie bien si le challenge est correct ou non. Si le challenge est correct, alors l'utilisateur peut se connecter et reçoit un JWT (Json Web Token). Dans le cas contraire, l'utilisateur ne peut se connecter.

5.2 Management des devices

L'application permet à chaque utilisateur d'ajouter autant de devices qu'il le souhaite. Quand un utilisateur ajoute un device, une paire de clés publique-privée est générée pour ce device. Cela se produit tout d'abord lorsque l'utilisateur va créer son compte pour la première fois, une paire de clés publique-privée est alors créé pour ce device. Ensuite, une fois connecté sur son device, l'utilisateur va avoir la possibilité d'ajouter d'autres devices. Une fois la paire de clés générée, l'utilisateur va devoir valider le device qu'il vient d'ajouter via un device de confiance. C'est-à-dire un device déjà "Trust" (de confiance) par l'utilisateur.

Par exemple, si un utilisateur souhaite ajouter un 2e device, il va devoir, sur son 1er device, cliquer sur le bouton "Trust" afin d'affirmer qu'il a confiance en ce device. Une fois cela fait, il va pouvoir se connecter à ce 2e device.

Une fois qu'un device est "Trust", il faut alors que l'utilisateur puisse avoir accès à ses photos sur ce nouveau device. Pour cela, chaque clé symétrique pour chaque photo va être chiffrée avec la clé publique de ce nouveau device. Pour les noms d'album, ils seront directement chiffrés avec la clé publique du device.

Dans le cas où un utilisateur décide de ne plus "Trust" un device car il l'a perdu où il n'y a plus accès, alors on ne peut ni avoir accès à ce device, ni aux photos et albums accessibles précédemment via ce device.

Lors de chaque requête vers le serveur, on vérifie d'abord si le device en question est "Trust" (de confiance). Dans notre code, chaque procédure est une "protectedProcedure" qui s'assure que le device qui est en train d'effectuer une requête est "Trust". Il existe des exceptions concernant le login et la connexion, sinon il serait impossible d'accéder à l'application.

5.3 Upload de photos et création d'un album

5.3.1 Upload de photos

Tout d'abord, il est possible d'upload une photo sans que cette dernière fasse partie d'un album. Lorsqu'une photo est upload par un utilisateur, une entrée est ajoutée pour la table "Picture" et "SharedPicture". Même si la photo n'est pas encore partagée avec quelqu'un, une entrée existe dans "SharedPicture", qui est la table permettant de voir qui a accès à la photo. Cette table va alors permettre de récupérer les photos de l'utilisateur qui ne sont pas présentes dans un album.

Lorsqu'une photo est upload, une clé symétrique est générée. Cette clé symétrique va alors être utilisée pour chiffrer la photo. Il n'existe qu'une seule clé symétrique par photo même si la photo se trouve dans différents albums. Il n'y a donc pas de clé symétrique par album, mais bien par photo. Le fait d'avoir une clé symétrique par photo permet, en cas de réussite d'un attaquant à récupérer cette clé, d'avoir accès uniquement à la photo en question et non à l'entièreté d'un album.

Il existe un cas particulier, lorsqu'une photo est ajoutée à un album avec une certaine personne et que cette dernière est déjà présente dans un autre album avec la même photo qui est, du coup, déjà partagée avec cette personne. Prenons 3 personnes : A, B et C. A et B ont un album en commun avec une photo x ajoutée par A. Si A, B et C crée un album commun, et que A décide d'y ajouter cette photo x, alors l'accès à la photo n'est donné qu'à C étant donné que B et A possèdent déjà l'accès à cette photo.

5.3.2 Création d'un album

Comme expliqué précédemment, la clé symétrique permettant de chiffrer les photos n'est pas basée sur un album, mais bien par photo. Lorsqu'un utilisateur crée un album, l'application va alors faire une requête au serveur afin de récupérer la clé publique de l'utilisateur afin de chiffrer le nom de l'album. Le nom de l'album est quant à lui, chiffré avec la clé publique du device de l'utilisateur. L'application envoie alors au serveur le nom de l'album chiffré ainsi que l'id du device sur lequel l'album a été créé. Le nom de l'album est alors déchiffré via la clé privée qui est stockée localement sur le device de l'utilisateur. Il s'agit du scénario classique lorsqu'un utilisateur possède un seul device.

Le choix d'utiliser la clé publique du device de l'utilisateur pour chiffrer le nom de l'album plutôt qu'une clé symétrique est que le chiffrement du nom reste quelque chose d'assez rapide contrairement au chiffrement des photos qui requière le chiffrement symétrique afin d'être performant.

Dans la réalité, quand on un utilisateur crée un album, l'application va faire une requête au serveur afin d'avoir toutes les clés publiques pour chaque device que possède l'utilisateur. Avec ces clés publiques, le nom de l'album est chiffré autant de fois qu'il y a de clés publiques. Un dictionnaire est alors renvoyé au serveur. Ce dictionnaire est composé d'un device id et d'un chiffrement lié à ce device via la clé publique de ce dernier.

Exemple : deviceID1 = chiffrementPublicKeyDeviceID1 deviceID2 = chiffrementPublicKeyDeviceID2 …

Selon le device utilisé par l'utilisateur, le nom de l'album est alors déchiffré via la clé privée stockée localement.

5.3.3 Stockage avec Minio

Pour envoyer la photo chiffrée au serveur, il y a d'abord besoin de la formater. Étant donné qu'en HTTP il n'est pas possible d'envoyer la photo chiffrée sous forme d'un array buffer, il faut formater la photo chiffrée en hexadécimal. Une fois les données relatives à la photo stockées dans la base de données, la photo est stockée dans un serveur à part (le serveur de fichier Minio).

Concernant l'intégrité des photos, au moment d'upload le chiffrement de la photo, une transaction SQL est crée. Cette transaction va alors permettre de s'assurer que l'upload de la photo sur le serveur de fichier s'est bien déroulé. Avant de terminer la transaction, on vérifie, via une méthode de Minio, si l'upload s'est bien déroulé et que les données relatives à la photo sont bien ajoutées à la base de données. S'il n'y a pas d'erreur, alors la transaction est validée et le chiffrement de la photo se trouve bien sur le serveur de fichiers Minio. En cas de problème, la transaction est annulée et les données relatives à la photo ne sont ni ajoutées à la base de données ni au serveur de fichiers. Cela permet de s'assurer que la photo soit bien ajoutée au serveur de fichier de manière intègre.

5.4 Partage des photos et albums

5.4.1 Partage d'une photo

Concernant le partage d'une photo, il faut d'abord récupérer les clés publiques de l'utilisateur avec qui on souhaite partager la photo. Pour cela, il y a une requête faite au serveur afin de récupérer toutes les clés publiques de chaque device de l'utilisateur à qui on souhaite partager notre photo. Le serveur va alors nous répondre avec un dictionnaire contenant chaque id de chaque device avec sa clé publique correspondante. Une fois les clés publiques récupérées, la clé symétrique utilisée pour chiffrer la photo va alors être chiffrée autant de fois qu'il existe de clé publique. Il y a alors un dictionnaire renvoyé au serveur contenant l'id de chaque device avec le chiffrement de la clé symétrique par device.

```
deviceID1 = chiffrementKeySymétricDeviceID1
deviceID2 = chiffrementKeySymétricDeviceID2
```

De plus, il est tout à fait possible de partager une photo qui n'est présente dans aucun album.

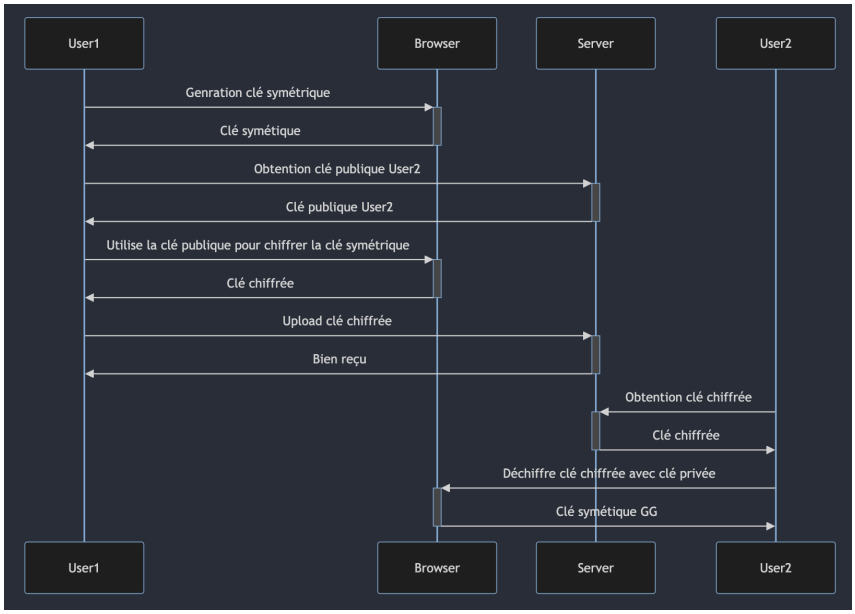

Figure 3 – Partage de photo

Comme le montre le diagramme ci-dessus, l'utilisateur avec qui la photo est partagée peut tout simplement déchiffrer la clé symétrique chiffrée avec sa clé privée stockée localement afin d'avoir accès à la valeur de la clé symétrique. Une fois cette valeur récupérée, il peut alors déchiffrer la photo et y avoir accès.

5.4.2 Partage d'un album

Concernant le partage d'un album, il s'agit du même mécanisme que celui décrit précédemment concernant le partage d'une photo. Il y a cependant le nom de l'album qui lui aussi est chiffré via la clé publique du device de l'utilisateur. Pour cela, il y a d'abord une requête au serveur pour récupérer les clés publiques de l'utilisateur afin de chiffrer le nom de l'album et ensuite une requête afin de récupérer les clés publiques de l'utilisateur avec qui on souhaite partager l'album. Comme expliqué précédemment, chacune des clés symétriques va alors être chiffrée par les clés publiques existantes.

Comme expliqué précédemment, l'utilisateur avec qui l'album est partagé va pouvoir déchiffrer le nom de l'album via sa clé privée.

6 Rate limiting

Comme cité plus haut dans le rapport, nous avons utilisé un système de rate limiting pour éviter les attaques de type brute force et DOS(Denial of Service). Son implémentation est assez simple, aucun package n'a été utilisé car aucun ne correspondait à nos besoins. Nous avons donc implémenté notre propre système de rate limiting. Celui-ci est un middleware qui est appelé avant les requêtes. Il incrémente un compteur stocké dans la base de données Valkey pour chaque adresse IP et chaque endpoint. Si le compteur dépasse une certaine valeur, une erreur est renvoyée. Voici le code de ce middleware :

```
export const rateLimitedMiddleware = t.middleware(
  async ({ path, ctx, next }) => {
    const res = await ctx.cache.incr(`${path}:${ctx.ip}`);
    if (res === 1) {
      await ctx.cache.expire(`${path}:${ctx.ip}`, env.RATE_LIMIT_WINDOW);
    }
    if (res > env.RATE_LIMIT_MAX) {
      logger.error(`Rate limit exceeded for ${ctx.ip} on ${path}`);
      throw new TRPCError({ code: "TOO_MANY_REQUESTS" });
    }
    return next();
  },
);
```

7 Protection contre les injections SQL

Pour nous protéger contre les injections SQL, nous avons utilisé un ORM (Object-Relational Mapping) nommé Prisma. Prisma est un ORM qui permet de manipuler la base de données sans écrire de requêtes SQL. Celui-ci est sécurisé par défaut et empêche les injections SQL.

8 Protection contre les attaques XSS

Pour nous protéger contre les attaques XSS, nous utilisons la librairie react-dom qui permet de manipuler le DOM de manière sécurisée. React DOM échappe automatiquement les caractères spéciaux et empêche les attaques XSS.

9 Analyse de code Snyk

Pour analyser notre code et détecter les vulnérabilités, nous avons utilisé Snyk. Snyk est un outil qui permet de détecter les vulnérabilités dans les dépendances de notre projet. Il scanne les dépendances et les compare à une base de données de vulnérabilités. Si une vulnérabilité est détectée, Snyk nous avertit et nous propose des solutions pour la corriger. De plus, Snyk peut également être utilisé pour détecter les vulnérabilités dans le code source.

10 Signature des commits git

Pour éviter une attaque de type “supply chain attack” ou d’usurpation d’identité, nous avons mis en place une signature des commits git. Cela permet de vérifier l’identité de l’auteur via sa clé GPG. Ainsi, si un attaquant modifie le code source et ne signe pas le commit, il sera facilement détecté.