> 1. You are developing a software that takes the Facebook friends of a user and displays a nice friendship graph out of it. For this, you need to somehow gain access to the Facebook account of the user. Why isn't it recommended in this case to use passwords ? Give all the reasons you can think of.

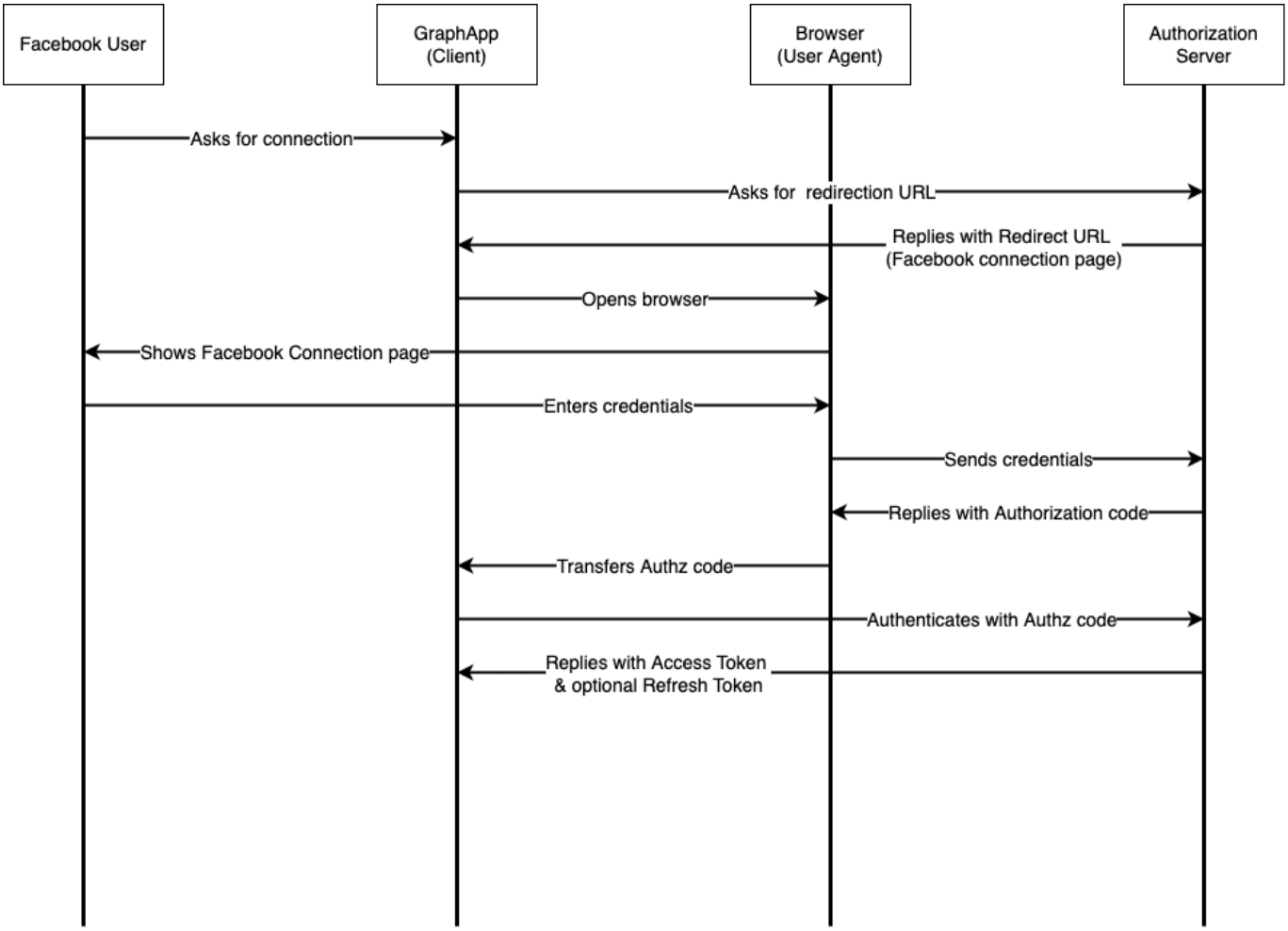
L'application doit essentiellement accéder à des données qui existent sur un service tier, en l'occurence Facebook. Son but n'est en aucun cas de fournir un moyen d'authentification. Pour fonctionner, elle a simplement besoin d'un utilisateur disposant d'compte Facebook. Il est donc logique de déléguer à Facebook l'authentification. En plus, cela évite la gestion de mots de passe avec tout ce que cela implique en terme de sécurité de stockage des données.

> 2. Let's suppose that you are using OAuth2 for managing the authorization of your Facebook friendship graph website. Draw a schema describing what are the interactions between your app, Facebook, and the user for a legitimate request (obtaining Facebook friends) and for an illegitimate request (e.g. obtaining the private messages of the user). Explain which mechanism will disallow the app to recover private messages.

Avant tout, il est nécessaire de bien compendre les concepts suivants, utilisés par OAuth2:

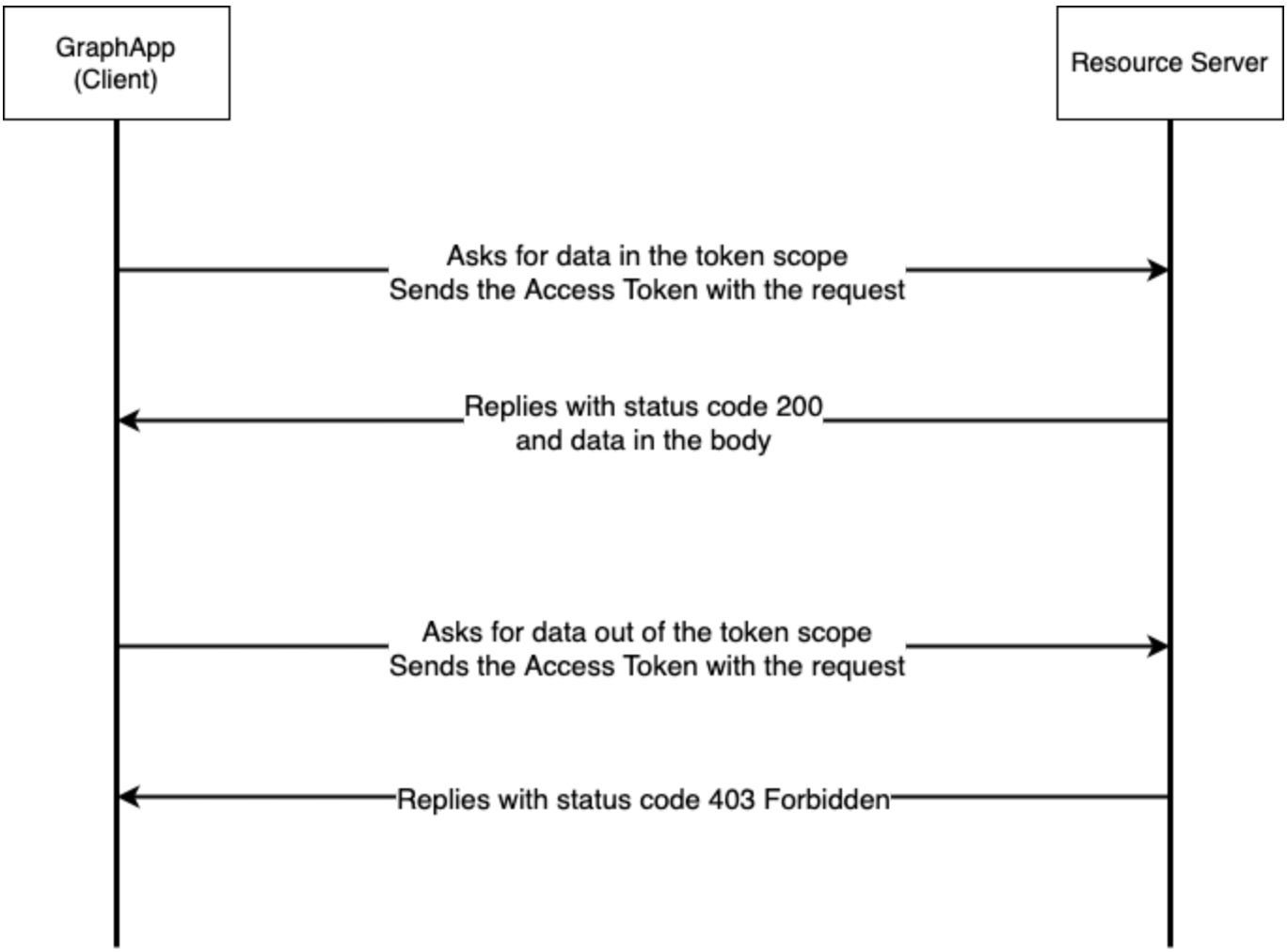
- `Resource Owner` : Il s'agit de l'utilisateur qui possède les données sur le service tier, dans notre cas l'utilisateur Facebook.
- `Client` : Il s'agit de l'application qui accède aux resources tièrces dont l'autorisation d'accès se fait au moyen de OAUTH2.
- `User Agent` : Il s'agit de l'application qui est utilisée pour communiquer avec le serveur d'autorisation du service duquel le `Client` veut accéder aux ressources. Cela est dans la plupart de cas le navigateur web en lui-même ou une application mobile. Le `User Agent` est conceptuellement distinct du `Client`, qui lui est l'application qui veut accéder aux données. Le `User Agent` peut être considéré comme un tier de confiance du point de vue du `Resource Owner`.
- `Authorization Server` : Il s'agit du serveur d'autorisation du service duquel le `Client` veut accéder aux ressources.
- `Resource Server` : Il s'agit en lui-même du service duquel le `Client` veut accéder aux ressources.
- `Athorization code` : Il s'agit du code utilisé pour authentifier le `Client`.
- `Access Token` : Il s'agit du token permettant au `Client` d'accéder aux ressources du `Resource Owner`. Ce token ne doit révéler aucune information sur l'utilisateur au `Client`. Il contient une durée de validité et un *scope*, qui sont vérifiés à chaque requête par le `Resource Server`.
- `Refresh Token` : token optionnel permettant au client de demander un nouvel `Access Token`, lorsque la validité de celui-ci échoit, sans avoir besoin d'intéragir avec l'utilisateur. Cela permet à l'`Authorization Server` de délivrer des `Access Token` token de très courte durée de validité. C'est particulièrement utile si le `Resource Server` est de type *stateless*.

Ce diagramme montre les interations ayant lieu lors de l'authentification :



Dans l'`Access Token` délivré dans le cadre de notre application, le *scope* donne l'autorisation d'accéder uniquement aux amis de l'utilisateur (`Resource Owner`). Quant à la présence d'un `Refresh Token`, elle dépend de l'API Facebook et de son implémentation de OAuth2.

Ce diagramme montre les interations entre le `Client` et le `Resource Server` :



Lorsque le `Resource Server` reçoit une requête, il vérifie d'abord l'athenticité et l'intégrité du token. Il vérifie aussi sa date de validité. Ensuite, il vérifie que la ressource demandée par le `Client` est comprise dans le *scope* du token, c'est-à-dire que le `Client` a le droit d'accéder à la ressource demandée. Ce mécanisme permettrait de rejeter une requête du `Client` pour les messages privés du `Resource Owner`.

> 3. Your Facebook friendship graph website runs only over HTTP. Although this is bad, you cannot change this. You decide to use the OAuth2 protocol for authorization. However, by default, the requests are not signed (unlike in OAuth1). What could an adversary do ? Be precise and provide an example. Provide also a detailed technical solution that would fix the problem. Don't forget to propose algorithms. Explain also how the keys are managed.

> 4. In our scenario, how would a user revoke access to his account ? Explain technically what it implies. What prevents a malicious application to reuse some previously generated tokens ?

Pour révoquer un client OAuth2, il faut signaler à l'`Authorization Server` que l'`Access Token` lié à cette application doit être invalidé. Il se peut que l'`Authorization Server` utilise des `Refresh Token`. Dans ce cas, se sont eux que le serveur doit invalider. Typiquement l'`Authorization Server` maintient une base de donnée des token valides et les supprime lorsque l'utilisateur demande leur révocation, ou que le serveur détecte que le client a été compomis.
Dans le cas de notre application, une possibilité est d'offrir une fonction de *logout* qui permette de révoquer l'accès aux données utilisateurs directement. Une seconde possibilité est que l'utilisateur utilise

directement l'interface proposée par Facebook qui liste les applications autorisées et qui permet de supprimer les clients.
Au niveau de ce que cela implique, cela dépend des choix d'implémentation côté Facebook. Si un système de base de donnée de type whitiste est utilisé et que le `Resource Server` vérifie à chaque requête si l'`Access Token` est dans cette liste, la révocation est immédiate. A noter que dans ce cas, les optionnels `Refresh Token` ne sont pas utilisés.
Une seconde possibilité implique l'utilisation de `Refresh Token` et `Resource Server` de type *stateless*. Dans ce cas, l'`Authorization Server` maintient une liste de ces token valides. Révoquer un client implique alors dans les faits de révoquer les `Refresh Token` qui lui sont liés. Ce dernier ne pourra alors plus obtenir des `Access Token`. L'accès de ce client aux données utilisateurs sera alors effectif lors de l'échéance du dernier `Access Token` délivré par l'`Authorization Server`. Autrement dit, la révocation de l'accès n'est dans ce cas pas immédiate. C'est pourquoi dans le cas d'un `Resource Server` de type *stateless*, il est recommandé que les `Access Token` est une durée de validité très courte.

> 5. In your Facebook friendship graph software, what information would typically contain all the JWTs used by OAuth2 ? Would you need to encrypt the token ? Justify. What expiration time would you typically set ?
>
>   OAuth2 a besoin d'au moins un token, l'`Access Token`. Il y a aussi l'`Authorization Code`, mais comme son nom l'indique, c'est un simple code. Il est définit au chapitre 1.3.1 de la RFC 6749 (OAuth2).
>
>   L'`Access Token` est défini au chapitre 1.4 de la RFC 6749. C'est une chaîne de caractère qui représente des identifiants permettant d'accéder à une ressource délévrée par un `Resource Server`. La RFC 6749 précise que ce token peut avoir différent formats, structures et méthodes, et que cela dépend des besoins.
>
>   On va donc supposer que Facebook a choisit JWT comme format pour ses `Access Token`.
>
>   La RFC 6749 ne précise pas quelles informations doivent être contenues dans ce token et renvoie à des spécifications annexes.
>
>   L'une d'entre elle, la RFC 6750, définit un type de token appelé "Bearer". La documentation officielle de OAuth2 indique que c'est le type de token prédominant utilisé pour les `Access Token`.

Nous allons nous intéresser à 2 token en particulier, l'`Access Token` et le `Refresh Token`. La documentation officielle

La RFC 9068 étend la spécification de OAuth2 en définissant le format de