

SLH 2024-2025

Labo 3 - Biscuits

1. Introduction & Consigne

Le système Karamel™ est une architecture **distribuée en microservices**, pour le stockage **sécurisé** du dossier électronique du patient.

Karamel™ laisse à l'utilisateur le **libre choix du fournisseur** de services, et utilise des **biscuits**¹ pour permettre à un **Patient** de donner accès à son dossier au **Médecin** de son choix.

Répondez aux questions (en bleu) dans le fichier ANSWERS.md, qui sera inclus avec le rendu. Merci de ne **PAS MODIFIER** les lignes déjà présentes dans ce fichier; écrire uniquement la réponse dans la zone vide entre deux titres de questions.

Implémentez les fonctionnalités demandées (en turquoise) directement dans le code.

Pour préparer le rendu, exécutez la commande cargo package puis déposez le fichier .crate situé sous target/package/.

2. Présentation de l'Architecture (8 pts)

Les acteurs suivants sont impliqués dans le fonctionnement du système:

- **l'Autorité (Directory)** est le référent de confiance unique, en charge de la gestion des utilisateurs et de leur authentification.
- les **fournisseurs (store)** sont des prestataires qui offrent un service de stockage des données. Ils choisissent librement quels utilisateurs peuvent stocker quelles données chez eux.
- Les **patients** s'enregistrent auprès de l'**Autorité** pour obtenir un identifiant, et sont libres de choisir qui peut accéder à leurs données et sous quelles conditions, chez un **fournisseur** quelconque.
- Les **médecins** envoient des rapports aux **fournisseurs** de leur choix, et peuvent accéder aux données de n'importe quel **patient** chez n'importe quel **fournisseur**, à partir du moment où le **patient** y a consenti.

La Fig. 1 présente le processus d'autorisation de Karamel™:

¹<https://biscuitsec.org>

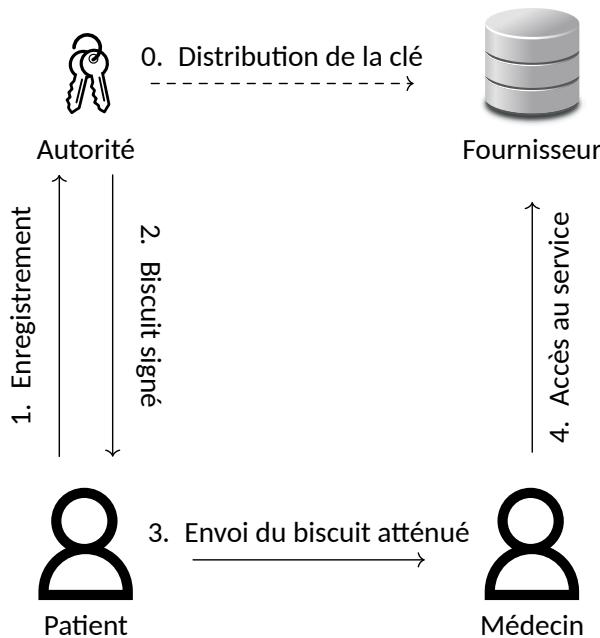


Fig. 1. – Architecture de Karamel™

0. La **clé publique** de l'Autorité a été préalablement **distribuée** à tous les acteurs du système.
1. Un utilisateur (**Patient** ou **Médecin**) s'**enregistre** auprès de l'Autorité, qui lui attribue un **identifiant** unique.
2. L'autorité délivre à l'**Utilisateur** un **biscuit** donnant accès à toutes les parties de son dossier
3. Un **Patient** qui le souhaite peut, à sa discrétion (et donc sans interaction avec l'Autorité) **atténuer son biscuit** pour donner accès uniquement à **une partie** de ses informations médicales (rapports précis, ou dans un intervalle de temps prédéterminé, ou sur un sujet précis), et **transmettre** ce biscuit à son médecin.
4. Le **Médecin** peut utiliser le **biscuit atténué** pour **accéder** directement au **service de stockage**

Q1 (8 pts)	Lesquelles des affirmations suivantes sont vraies ou fausses ? (justifier chaque réponse). <ul style="list-style-type: none"> • L'Autorité peut accéder à toutes les informations médicales si elle le désire • L'Autorité peut déterminer avec quel médecins un patient partage ses données • Un fournisseur peut déterminer tous les médecins avec lesquels un patient partage ses données • Un médecin peut partager des données auxquelles il a accès autre médecin, sans le consentement du patient.
---------------	--

3. Organisation de l'Implémentation

Le code de Karamel™ est organisé comme suit:

authorization.rs contient la **logique d'autorisation** pour le fournisseur de stockage
db.rs est un mécanisme de **persistence** des données à fin de débogage. En production, on utiliserait évidemment une vraie base de données.
model.rs contient le **modèle de données** utilisé pour les objets à stocker, et du code ancillaire pour permettre leur utilisation dans le contexte des biscuits
password.rs gère le stockage de **mots de passe**
protocol.rs définit les messages utilisés pour **communiquer** avec les **API** de l'autorité et des fournisseurs de stockage.
bin/directory.rs est l'exécutable du **serveur de l'autorité**
bin/store.rs est l'exécutable d'un **service de stockage**
bin/karamel.rs est le **client** en ligne de commande.

4. Analyse du code (12 pts)

Démarrez votre propre serveur d'autorité avec cargo run --bin directory, et démarrez un serveur de stockage avec cargo run --bin store.

Vous pouvez lancer le client, par exemple pour consulter l'aide, avec cargo run -- help.

Le code utilise le mécanisme de tracing pour les logs; vous pouvez affiner le détail des logs de la manière standard avec la variable d'environnement RUST_LOG.

Q2 (2 pts)	En principe, les utilisateurs doivent pouvoir s'inscrire librement auprès de l'autorité, mais les médecins doivent être validés manuellement. Le développeur a toutefois laissé une backdoor pour l'enregistrement des médecins . Quelle URL pouvez-vous utiliser pour enregistrer un médecin ? Sous quelle CWE pourrait-t-on classer cette vulnérabilité ?
Q3 (2 pts)	Examinez le fonctionnement du processus de login entre l'exécutable client (src/bin/karamel.rs) et le serveur d'autorité; que contient le message envoyé par le serveur au client, que contient la réponse en cas de succès, et que fait le client avec la réponse ?
Q4 (2 pts)	Pour assurer la confidentialité des mots de passe pendant le processus de login, quelle fonctionnalité essentielle devrait absolument être ajoutée avant de déployer ce code en production ?
Q5 (2 pts)	Examinez le fonctionnement du serveur de stockage. Comment la clé publique de l'autorité est-elle passée au processus du serveur de stockage ?
Q6 (2 pts)	read_report dans bin/store.rs renvoie une erreur 404 si l'ID du rapport demandé n'existe pas, avant de procéder à la décision d'autorisation. Est-ce un problème ? Justifier.
Q7 (2 pts)	Le serveur d'autorité implémente un mécanisme de défense contre un canal auxiliaire pour empêcher l'énumération des utilisateurs. Est-ce pertinent ? Justifier.

5. Contrôle d'accès (20 pts)

Q8 (6 pts)	<p>Listez les noms des faits datalog disponibles pour effectuer l'autorisation, en indiquant ceux provenant du biscuit et ceux provenant du contexte de la requête</p>
Q9 (2 pts)	<p>Par rapport à un système d'ABAC tel que vu dans Karak, y a-t-il des limitations sur la manière dont les règles d'accès peuvent dépendre du contenu d'un rapport ?</p>
C10 (10 pts)	<p>La politique d'autorisation actuelle permet à chacun d'accéder aux données le concernant, quelle que soit l'opération. Modifiez la politique d'autorisation pour obtenir les règles suivantes:</p> <ul style="list-style-type: none">• Les patients peuvent lire et modifier leurs informations personnelles, et lire les rapports les concernant• L'auteur d'un rapport peut lire ce rapport• Les médecins peuvent lire les informations personnelles de tout le monde.• Seuls les médecins peuvent créer de nouveaux rapports, et ils doivent être l'auteur du nouveau rapport.
Q11 (2 pts)	<p>Dans le système tel qu'il est implémenté, que se passe-t-il si un utilisateur perd ses droits de médecin ? Que proposez-vous pour limiter le problème, sans modifier la structure des communications entre les différentes parties ?</p>

6. Atténuation (8 pts)

Le client implémente la commande `lock`, qui écrase le token en cours d'utilisation avec un token atténué.

Q12 (8 pts)	<p>Vous désirez fournir à votre médecin l'autorisation de lire certains rapports médicaux. Proposez une commande pour créer un token atténué permettant l'accès à chacun de ces cas:</p> <ul style="list-style-type: none">• un seul rapport en particulier• tous les rapports concernant les problèmes de cœur (keyword « heart »), et postérieurs au 1er janvier 2010• tous les rapports par un médecin particulier
----------------	--

7. Validation d'entrées (8 pts)

Il n'existe pour l'instant aucune validation des entrées, ni pour les comptes utilisateurs, ni pour les mots de passe. Implémentez la validation manquante, **en utilisant les libs déjà à disposition dans le projet.**

C13
(4 pts) **Modifiez** le serveur d'**autorité** pour assurer que les **noms d'utilisateurs** correspondent syntaxiquement uniquement à des **adresses email** valides.

C14
(4 pts) **Modifiez** le serveur d'**autorité** pour rejeter les mots de passe trop faibles (critère: score Zxcvbn < 3)