CHAPITRE 4 : APPROCHE

Deux (02) options de procédé :

- **OPTION 1 :** algorithmes de hachage simple + blockchain
- **OPTION 2 :** algorithmes de hachage + signature numérique + blockchain
- **Chaque option est constituée de deux (02) phases (voir figure 5 ci-dessous) :**
  - **Phase 1 :** Enregistrement de documents administratifs dans la blockchain
  - **Phase 2 :** Vérification/authentification de documents administratifs via la blockchain

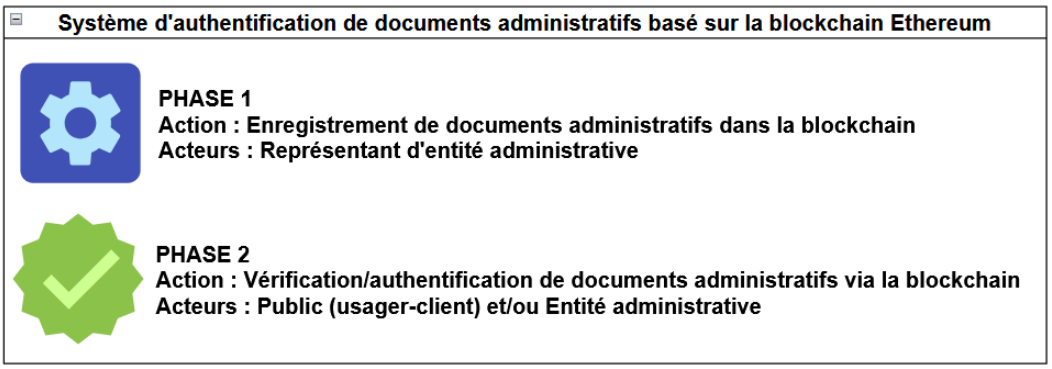

***Figure 1** : Différentes phases de l'approche d'authentification de documents administratifs à l'aide de la blockchain*

OPTION 1 (PHASE 1) : ALGORITHMES DE HACHAGE SIMPLE + BLOCKCHAIN

ALGORITHME PHASE 1 (voir figure 6 ci-dessous)

1. L'auteur upload (via le frontend) la version finale du document PDF ou Word. Le frontend transmet ces données au backend.
2. Le backend calcule un hash SHA-256 du document uploadé.
3. Le backend transfère le hash brut au contrat intelligent qui le stocke sur la blockchain. Une notification est retournée au frontend depuis la blockchain en passant par le backend.

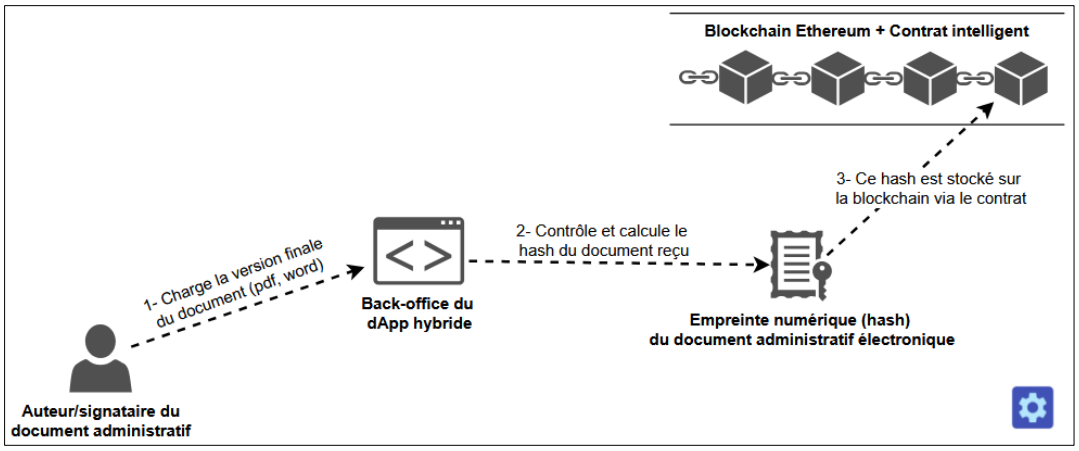

***Figure 2** : Etapes d'enregistrement de document dans la blockchain en utilisant le hachage simple + blockchain (OPTION1-PHASE1)*

OPTION 1 (PHASE 2) : ALGORITHMES DE HACHAGE SIMPLE + BLOCKCHAIN

ALGORITHME PHASE 2 (voir figure 7 ci-dessous)

1. L'utilisateur (public ou agent public) upload le document PDF ou Word (via le frontend). Le frontend transmet ces données au backend.
2. Le backend calcule le hash du document soumis à l'authentification.
3. Le backend utilise cet hash et interroge la blockchain via le contrat intelligent pour récupérer une occurrence du hash (empreinte numérique) auparavant stocké dans la blockchain.
4. La blockchain retourne une réponse au backend via le contrat intelligent.
5. Le backend compare le hash recalculé avec celui stocké sur la blockchain. Si les hashs ne sont pas identiques, alors le document soumis à authentification a été modifié et n'est donc pas authentique (vérification de l'intégrité du document). S'ils correspondent, on conclut que le document est authentique.
6. Après comparaisons, une notification est faite par le backend à l'utilisateur via le frontend.

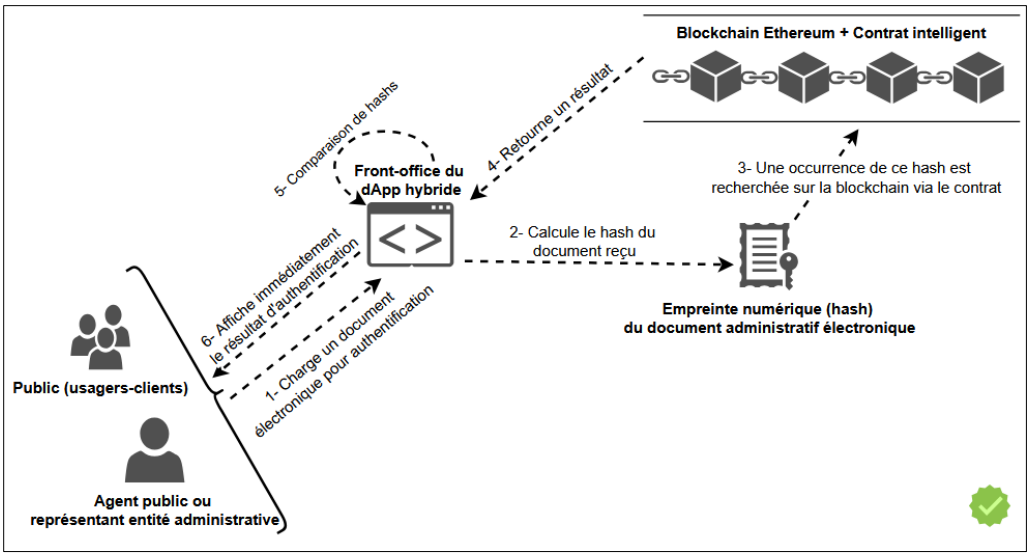

Figure 3 *: Etapes de vérification/authentification de documents via la blockchain (OPTION1-PHASE2)*

OPTION 2 (PHASE 1) : ALGORITHMES DE HACHAGE + SIGNATURE NUMERIQUE + BLOCKCHAIN

ALGORITHME PHASE 1 – SIGNATURE NUMERIQUE BASEE SUR SHA256WITHECDSA (P-256) (voir figure 8 ci-dessous)

1. L'auteur génère une paire de clés (via le backend en passant par le frontend). Ces clés ne doivent pas être perdues et la clé privée doit être soigneusement conservée.
2. L'auteur intègre la clé publique dans le document de sorte à ce qu'elle soit visible.
3. L'auteur upload (via le frontend) la version finale PDF ou Word du document contenant sa clé publique. Lors de l'upload, il renseigne la paire de clés. Le frontend transmet ces données au backend.
4. Le backend calcule un hash SHA-256 du document uploadé. Le backend chiffre le hash calculé avec la clé privée de l'auteur (on obtient donc un hash signé – c'est la signature numérique).
5. Le backend transfère le hash brut, le hash signé et la clé publique (le tout encodé) au contrat intelligent qui les stocke sur la blockchain. Ainsi, lorsqu'un utilisateur veut vérifier l'authenticité, la clé publique correspondante est utilisée pour déchiffrer et vérifier la signature.

6. Une notification est retournée au frontend depuis la blockchain en passant par le backend.

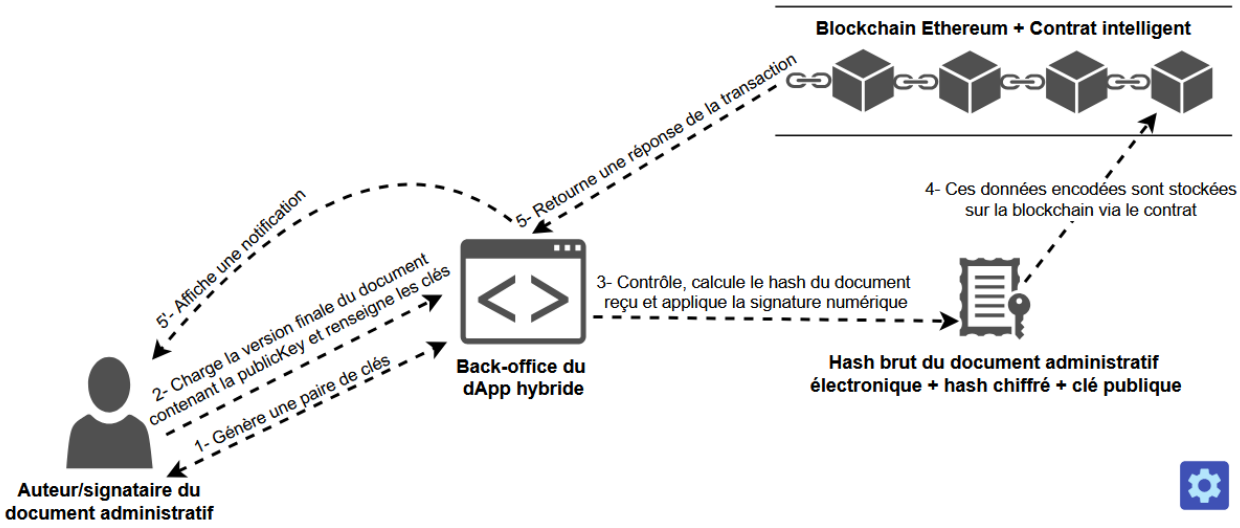

***Figure 4** : Etapes d'enregistrement de document dans la blockchain en utilisant le hachage + signature numérique + blockchain (OPTION2-PHASE1)*

OPTION 2 (PHASE 2) : ALGORITHMES DE HACHAGE + SIGNATURE NUMERIQUE + BLOCKCHAIN

ALGORITHME PHASE 2 (voir figure 9 ci-dessous)

1. L'utilisateur (public ou agent public) upload le document PDF ou Word (via le frontend). Le frontend transmet ces données au backend.
2. Le backend calcule le hash du document soumis à l'authentification.
3. Le backend utilise cet hash et interroge la blockchain via le contrat intelligent pour récupérer l'empreinte numérique brute, la signature numérique et la clé publique auparavant stockées dans la blockchain.
4. La blockchain retourne une réponse au backend via le contrat intelligent.
5. Le backend décode les données reçues de la blockchain et effectue 2 vérifications :
    a. Il compare le hash recalculé avec celui stocké. Si les hashs ne sont pas identiques, alors le document a été modifié et n'est donc pas authentique (c'est la vérification de l'intégrité du document). S'ils correspondent, on passe à la seconde vérification (b).
    b. Il vérifie la signature numérique avec la clé publique stockée. Si cette vérification échoue, alors la signature ne provient pas du bon signataire, c'est-à-dire de l'auteur

ayant validé la version finale du document (c'est la vérification de l'authenticité du document). Si la signature est valide, alors on déclare que le document est authentifié et authentique.

6. Après vérifications, une notification est faite par le backend à l'utilisateur via le frontend.

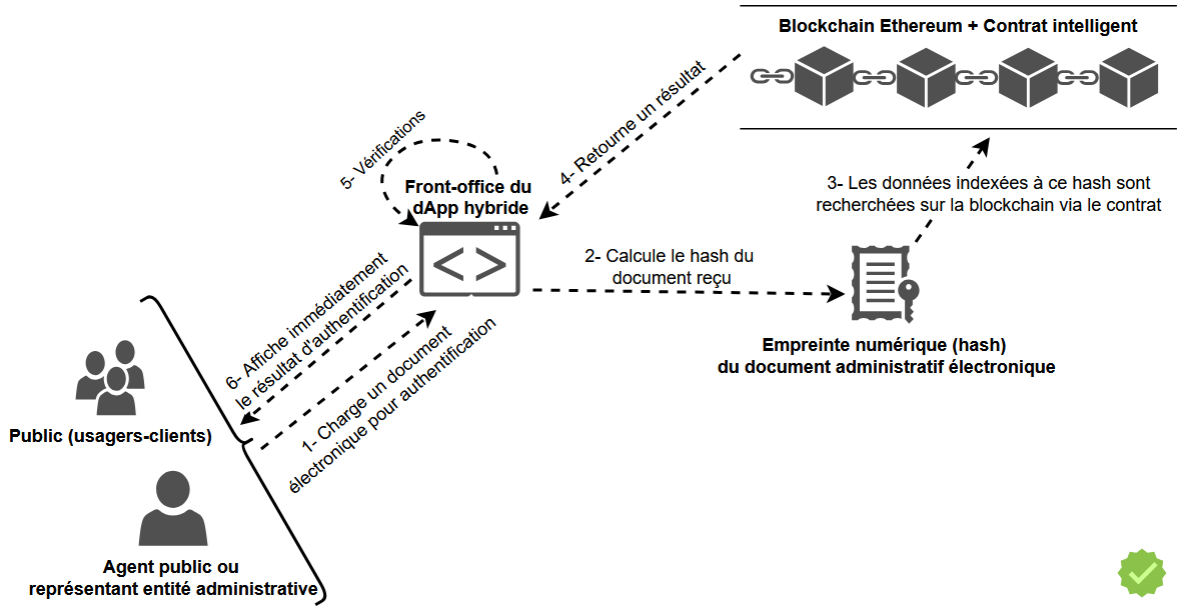

***Figure 5** : Etapes de vérification/authentification de documents via la blockchain (OPTION2-PHASE2)*

ARCHITECTURE EN COUCHE DE L'APPROCHE

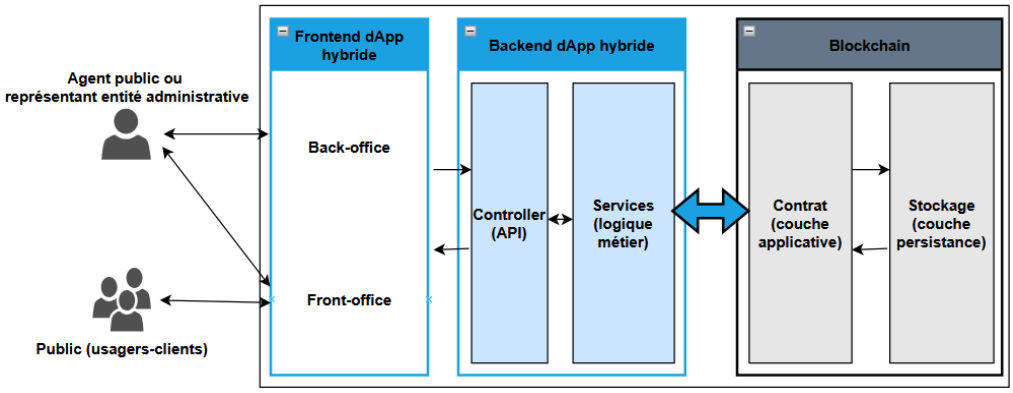

***Figure 6** : Architecture en couches de l'approche*

Technologies cibles à utiliser :

- **Ethereum Mainnet** (coûteux en frais de gaz, mais sécurisé) comme réseau public
- **Truffle** et **Ganache pour déployer et tester la solution localement**
- **Contrat intelligent en Solidity**
- **Frameworks Spring Boot et Angular**
- **Algorithme de hachage SHA-256, chiffrement/déchiffrement avec ECDSA (courbe P-256)**
- **Librairie Web3j, API JSON RPC**
- **IDEs IntelliJ IDEA 2024.1.4, Visual Studio Code 1.96.2, Remix-Ethereum IDE**

Mémo d'implémentation de l'approche :

1. Créer le hash du document via un backend Spring Boot. Ce code reçoit en entrée, le document PDF ou WORD et calcule son hash en utilisant **SHA-256**. Le hachage de PDF textuel et de WORD peut se faire sans inquiétude. Par contre, pour les scans images dans un fichier PDF, on peut premièrement extraire le texte du fichier via un système **OCR** (Reconnaissance Optique de Caractères) et calculer le hash de ce texte uniquement. Deuxièmement, on peut hacher directement l'image (le fichier binaire complet). Cependant pour ce deuxième cas, si par exemple, je scanne deux fois le même document papier en PDF avec des imprimantes différentes, les fichiers auront-ils le même hash ? **NON** : car les images scannées ont des variations en terme de résolution, d'alignement, de luminosité, de compressions, etc. et contiennent des métadonnées (date de création, logiciel utilisé, etc.) différentes. Dans tous les cas, nous sommes certains qu'un hash demeure identique tant que le texte extrait est strictement le même.

   S'il y avait la possibilité d'avoir une paire de clés, l'administration pourrait joindre la clé publique afin que celle-ci plus le document soient hachés. Ainsi, l'administration marquerait cette clé publique visiblement sur le document à l'endroit des usagers.
2. Créer un smart contract en **Solidity** qui permet d'enregistrer un hash sur la blockchain et de vérifier son existence et sa validité.
3. Utiliser **Web3j** pour faire interagir le backend Spring Boot avec la blockchain. Ainsi, le hash du document calculé par le backend sera directement enregistré sur la blockchain en renseignant l'adresse du compte contrat, l'url **RPC** pour Ganache et la clé privée de test (par exemple). De même, pour la vérification, il suffira que le backend recalcule le hash du

document qu'il va passer en paramètre à l'appel de la méthode de vérification du contrat qui est sur la blockchain.

4. Créer un frontend Angular qui se chargera d'envoyer le document au backend. Il aura aussi la charge de recevoir la réponse de vérification venant du backend qu'il va afficher.
5. On peut dire que **notre solution est une dApp hybride** (de rigueur) car le backend sert uniquement de passerelle vers la blockchain et ne stocke aucune donnée dans une base de données classique. Si on veut une dApp pure, il faut rendre le backend plus léger et utiliser Metamask / WalletConnect dans le frontend Angular pour interagir directement avec Ethereum. *NB : Une application est dite dApp, si elle repose uniquement (sans backend classique) sur Ethereum (contrat intelligent) pour la logique métier et la persistance*.
6. Déploiement :

   Déploiement du Smart Contract Ethereum : le faire sur un réseau public pour être conforme au choix de Ethereum Mainnet. Cela garantira que la solution soit totalement décentralisée. Pour les besoins de tests en réseau, on pourra utilise Testnets (Goerli, Sepolia, ou autre) : idéal pour tester avant d'aller sur mainnet.

   Déploiement du Backend (Spring Boot) : Le backend doit être accessible au frontend et pouvoir interagir avec la blockchain. Option Cloud (AWS) ou Serveur dédié virtuel (VPS – vitual private server) Linux avec Docker.

   Déploiement du Frontend (Angular) : L'application frontend doit être accessible aux utilisateurs finaux. Option Cloud AWS ou Serveur Nginx.