جامعة بجاية
Tasdawit n Bgayet
Université de Béjaïa

République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université Abderrahmane Mira - Béjaïa
Faculté des Sciences Exactes
Département d'Informatique

Présenté à l'

Université Abderrahmane Mira - Béjaïa

Faculté des Sciences Exactes

En vue de l'obtention du diplôme de

Master Professionnel En Informatique

Option : Administration et Sécurité des Réseaux

Intitulé du ***mémoire*** *:*

Application des Blockchains dans le contrôle d'accès distribué appliqué au Cloud Computing

---

Présenté par :

Mlle. RAMDA Lydia
Mlle. TIBOUCHE Katia

Encadré par :

Dr. DJEBARI Nabil

Soutenu le 30 Juin 2025 devant le jury composé de :

Président :	Dr. MOHAMMEDI Mohammed
Examinateur :	Dr. BELKHIRI Louiza
Examinateur :	Dr. KESSIRA Dalila
Examinateur :	Dr. EL BOUHICI Houda

Année Universitaire : 2024 - 2025

Remerciements

Au terme de ce travail, nous tenons à exprimer nos sincères remerciements à toutes les personnes qui ont contribué de près ou de loin à la réalisation de ce mémoire.

On tiens à adresser notre profonde gratitude à notre encadrant universitaire, **Dr. Djebari Nabil**, pour son encadrement de qualité, sa disponibilité, ses conseils pertinents et ses orientations précieuses tout au long de ce projet. Son accompagnement attentif et ses remarques constructives nous ont permis de progresser et de mener à bien ce travail dans les meilleures conditions. On le remercie également pour sa patience et sa confiance, qui ont été des éléments essentiels à la réussite de ce mémoire.

on souhaite également exprimer notre reconnaissance à l'ensemble des enseignants et membres du jury pour l'intérêt qu'ils portent à ce projet et pour le temps qu'ils ont consacré à son évaluation.

On voudrais adresser nos remerciements les plus affectueux à nos famille, qui nous ont toujours soutenue avec amour et encouragements tout au long de notre parcours. Leur présence et leur confiance ont été une source inestimable de motivation.

Mes remerciements chaleureux vont également à mon ami **Yassine Chahid**, pour son soutien constant, ses encouragements et sa précieuse aide durant la réalisation de ce projet. Son accompagnement et ses conseils m'ont été d'un grand réconfort.
Enfin, je remercie tous mes amis pour leur amitié sincère et leur présence bienveillante tout au long de cette aventure.

Dédicace

À ma maman et à mon père pour leur sacrifice et leur soutien,
en témoignage de mon infinie reconnaissance et de mon profond attachement.
À mes deux sœurs Lilia et Nadjet, et à mon ami Yassine Chahid,
pour leur présence précieuse et leur encouragement constant.

À tous ceux qui me sont chers...

RAMDA Lydia

Je dédie ce mémoire à ma mère et à mon père pour leur soutien quotidien tout au long de ce parcours. Leur amour, leur patience et leurs encouragements ont été une source d'inspiration constante, et je leur en suis profondément reconnaissant.
À mon frère Didine, à ma sœur Melina, ainsi qu'à toute ma famille, qui ont toujours été présents pour moi.
À tous mes amis pour leur présence.

TIBOUCHE Katia

Table des matières

Table des figures

Liste des tableaux

Listings

Introduction générale

Avec l'essor du Cloud Computing et l'adoption massive des architectures distribuées, la gestion de l'accès aux ressources devient un enjeu majeur. Les entreprises utilisent de plus en plus d'environnements multi-cloud, ce qui complexifie le suivi et le respect des politiques de sécurité. La fragmentation des données et des services entre différents fournisseurs engendre des risques importants en matière de confidentialité, d'intégrité et de disponibilité des ressources.

[1]Dans ce contexte, la technologie blockchain, grâce à ses caractéristiques d'immutabilité, de traçabilité et de décentralisation, offre une réponse prometteuse aux problèmes de gouvernance dans les systèmes distribués. Couplée à une DAPI (Decentralized API [1] Gateway) et aux smart contracts, la blockchain permet d'orchestrer de manière autonome les requêtes d'accès aux ressources cloud tout en assurant une transparence complète sur les décisions prises par le système.

Malgré ces atouts, le pilotage des accès dans un scénario multi-cloud soulève de nombreuses difficultés. La coordination entre les différents fournisseurs, la gestion des conflits entre les décisions, le respect des dépendances entre services ainsi que le traitement des cas ambigus restent des défis à relever. L'absence d'un mécanisme de négociation et de traçabilité unifié compromet la fiabilité des décisions d'accès et la capacité d'audit, ce qui constitue le cœur de notre problématique.

Ce mémoire de fin d'études propose une architecture innovante DAccess-ChainMC (Decentralized Access Chain for Multi-Cloud) reposant sur une DAPI [2] et des smart contracts pour :

automatiser le traitement des requêtes d'accès entre plusieurs fournisseurs,

garantir une prise de décision fiable, traçable et auditée par le réseau blockchain,

gérer les incohérences entre les réponses des fournisseurs grâce à un module de négociation, et simplifier la gouvernance des accès dans des environnements cloud hétérogènes.

Organisation du mémoire :

1. Le Chapitre 1 présente le contexte général ainsi que la problématique du contrôle d'accès dans un environnement multi-cloud et expose les objectifs du projet.
2. Le Chapitre 2 dresse un état de l'art sur les modèles de contrôle d'accès existants, la technologie blockchain, les smart contracts, ainsi que les approches actuelles en matière d'interopérabilité multi-cloud.
3. Le Chapitre 3 décrit en détail l'architecture proposée DAccess-ChainMC (Decentralized Access Chain for Multi-Cloud) : les différents composants (DAPI, smart

1. API : Interface de Programmation Applicative
2. DAPI : Decentralized API Gateway

contract, fournisseurs cloud).

4. Le Chapitre 4 représente un scénario fonctionnel et les mécanismes de négociation.

Chapitre 1

Concepts Fondamentaux

1.1 Introduction

Ce premier chapitre expose les concepts fondamentaux sur lesquels reposent les architectures actuelles du Cloud Computing et de la Blockchain, deux technologies au cœur des systèmes distribués modernes.

Nous commençons par détailler les différents modèles de composition des services Cloud, ainsi que les mécanismes d'orchestration et de chorégraphie qui assurent la coordination entre ces services. La discussion se prolonge par une analyse du cycle de vie d'une requête Cloud, depuis sa soumission par le client jusqu'à sa prise en charge par le fournisseur, en passant par le rôle clé du broker et des orchestrateurs.

La seconde partie du chapitre est consacrée aux technologies de la Blockchain : sa structure, ses différents types (publique, privée, hybride), ainsi que les éléments cryptographiques qui lui confèrent sécurité, intégrité et traçabilité. Sont également expliqués les algorithmes de consensus (PoW [1], PoS [2], PoA [3]), le processus de minage, le rôle des signatures numériques et des arbres de Merkle dans la validation et l'intégrité des transactions.

Ces fondamentaux constituent le socle conceptuel indispensable à la compréhension des approches plus avancées présentées dans les chapitres suivants.

1.2 Cloud Computing

1.2.1 Principes du Cloud computing

1.2.1.1 Définition

[2]Le cloud computing désigne l'utilisation de serveurs accessibles via Internet pour l'hébrgement de logiciels, de bases de données et de services informatiques. Hébérgés dans des centres de données répartis à travers le monde, ces serveurs permettent aux utilisateurs et aux entreprises de se libérer de la gestion d'infrastructures locales.
Grâce au cloud, les données et applications restent accessibles depuis n'importe quel

1. PoW : Proof of Work (Preuve de travail)
2. PoS : Proof of Stake (Preuve d'enjeu / de participation)
3. PoA : Proof of Authority (Preuve d'autorité)

appareil connecté, puisque le stockage et les traitements sont effectués à distance.
Par exemple, un utilisateur peut retrouver l'intégralité de ses fichiers sur un service comme Google Drive,indépendamment de l'appareil utilisé.
Pour les entreprises, cette technologie réduit les coûts informatiques en supprimant la nécessité d'entretenir des serveurs internes, puisque c'est le fournisseur cloud qui s'en charge. De plus, elle facilite la collaboration et l'accès aux ressources à l'international.

Cette Figure 1.1 illustre le concept du Cloud Computing, ses principaux services (stockage, applications, bases de données, serveurs, accès mobile) ainsi que ses modèles de déploiement (cloud public, privé et hybride).

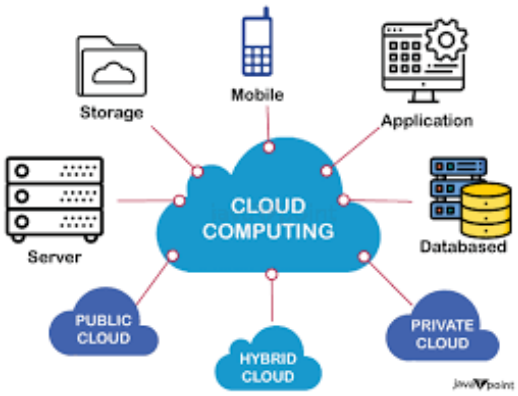

FIGURE 1.1 – Cloud Computing[2]

1.2.1.2 Services du cloud Computing

1. **Software-as-a-Service (SaaS) :**[2]Le logiciel proposé comme un service permet aux utilisateurs d'accéder au logiciel d'application et aux bases de données. Les fournisseurs de cloud gèrent l'infrastructure tandis que les utilisateurs stockent les données sur leurs serveurs.

2. **Platform-as-a-Service (PaaS) :**[2]Plate-forme en tant que service , accès aux outils et services de développement utilisés pour fournir les applications aux utilisateurs.
3. **Infrastructure-as-a-Service (IaaS) :**[2] L'infrastructure en tant que service , fournit des ressources informatiques virtualisées sur Internet. Le fournisseur héberge le matériel, les logiciels, les serveurs et les composants de stockage.

   La Figure 1.2 Services du Cloud Computing présente les trois modèles de services du Cloud Computing, à savoir l'Infrastructure en tant que Service (IaaS), la Plateforme en tant que Service (PaaS) et le Logiciel en tant que Service (SaaS).

FIGURE 1.2 – Services du Cloud Computing[2]

1.2.1.3 Types de déploiements Cloud Computing

1. **Cloud Privé :** [2]Également appelé cloud interne, cloud d'entreprise, un cloud privé comme représenté dans la Figure 1.3 est hébergé sur une plate-forme privée ,et offre à une entreprise plus de contrôle sur ses données, mais il peut être plus coûteux que d'autres services cloud en raison des coûts d'infrastructure, de maintenance et d'administration.

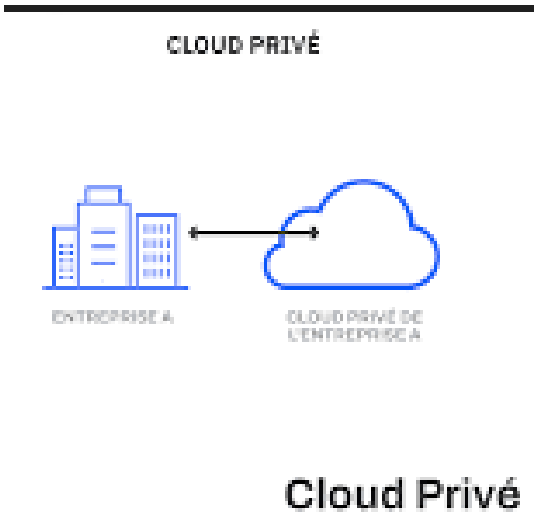

FIGURE 1.3 – Cloud Privé [3]

2. **Cloud Public :**[2] comme représenté dans la Figure 1.4 est hébergé par un fournisseur de services dans une installation hors site. Les utilisateurs payaient des frais d'utilisation mensuels ou annuels pour accéder au cloud. Cette option coûte moins cher à l'entreprise en matière d'infrastructure, de maintenance et d'administration, mais elle a moins de contrôle sur ses données.

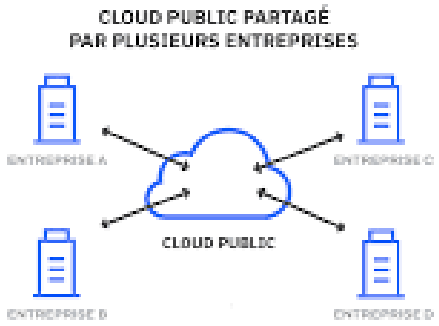

FIGURE 1.4 – Cloud Public [3]

3. **Cloud Hybride :** [2]Un cloud hybride comme représenté dans la Figure 1.5 combine le cloud privé et le cloud public en offrant le contrôle des données de l'entreprise, qui sont toujours hébergées dans un cloud public...

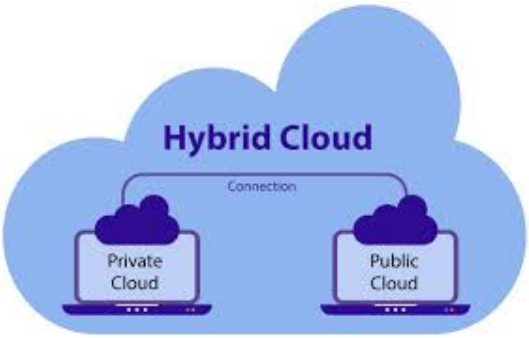

FIGURE 1.5 – Cloud Hybride [3]

4. **MultiCloud :**[2]L'approche comme représenté dans la Figure 1.6 constitue un type de déploiement cloud impliquant l'utilisation de plusieurs clouds publics. En d'autres termes, une entreprise qui s'appuie sur un déploiement multicloud loue des serveurs et des services virtuels auprès de plusieurs fournisseurs externes (pour reprendre l'analogie utilisée ci-dessus, l'opération revient à louer plusieurs terrains adjacents auprès de différents propriétaires). Les déploiements multicloud peuvent également concerner des clouds hybrides, et vice-versa.

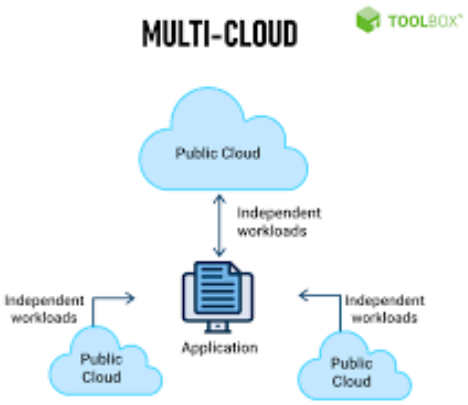

FIGURE 1.6 – Multi-Cloud [3]

1.2.2 Composition Cloud

1.2.2.1 Définition

[4] La composition de plusieurs clouds, dans le cadre d'une architecture multi-cloud ou cloud hybride, repose sur l'intégration de diverses infrastructures cloud, qu'elles soient privées, publiques ou issues de différents fournisseurs. Cette approche stratégique permet aux entreprises de tirer parti des spécificités et des avantages de chaque solution cloud, en adaptant leurs déploiements selon les besoins en termes de performance, de coûts, de conformité et de sécurité.
L'architecture multi-cloud consiste à utiliser plusieurs fournisseurs de services cloud, ce qui permet d'éviter la dépendance envers un seul prestataire, d'améliorer la résilience et d'optimiser la répartition des charges de travail en fonction des capacités et des coûts de chaque plateforme. De son côté, le cloud hybride combine une infrastructure cloud privée et un ou plusieurs clouds publics, permettant ainsi une plus grande flexibilité dans la gestion des données et des applications. Cette configuration est particulièrement utile pour les entreprises ayant des exigences strictes en matière de conformité et de sécurité, tout en souhaitant tirer parti des ressources élastiques du cloud public pour absorber des pics d'activité.
En combinant ces approches, les organisations peuvent concevoir des solutions adaptées à leurs besoins spécifiques, tout en améliorant la continuité des activités, la réactivité aux évolutions du marché et la rationalisation des coûts d'exploitation.

La Figure 1.7 illustre la composition du Cloud, mettant en évidence l'interaction entre le client, les applications et différents services tels que les plateformes, le CRM, le stockage

et le traitement de données massives.

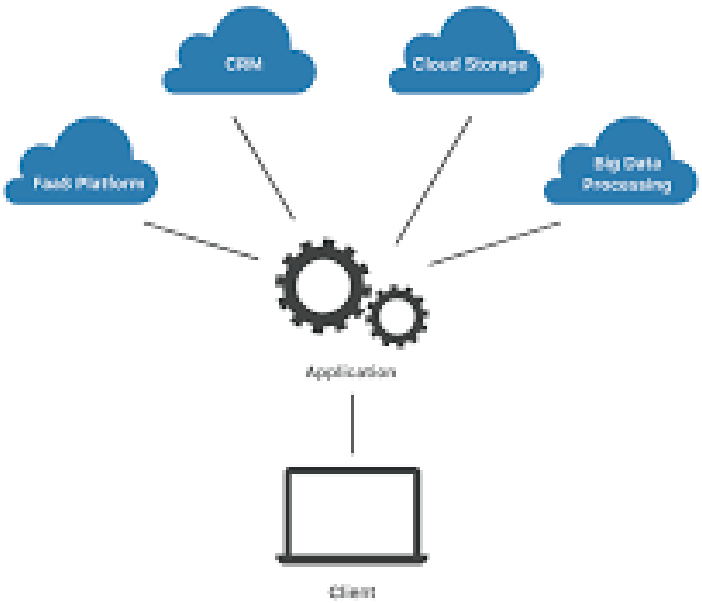

FIGURE 1.7 – Composition Cloud [5]

1.2.3 Types de composition du Cloud Computing

Dans une architecture cloud, la composition des services peut être réalisée selon deux approches principales : la composition statique et la composition dynamique.
Ces approches influencent directement la gestion et l'évolution des applications hébérgées dans le cloud.

1. **Composition statique :** repose sur la sélection et la configuration préalable des services cloud, qui restent fixes tout au long du cycle de vie de l'application. Cette approche est adaptée aux besoins constants et prévisibles, où les ressources nécessaires sont bien définies dès le départ.
   Exemple : Une plateforme e-commerce qui repose sur un serveur fixe, une base de données relationnelle et un réseau de diffusion de contenu (CDN) [4] pour optimiser la distribution des médias aux utilisateurs.
2. **Composition dynamique :** À l'inverse, la composition dynamique permet dâajuster les services et les ressources en temps réel en fonction des évolutions de la demande ou des changements d'environnement. Cette flexibilité est particulièrement utile pour les systèmes nécessitant une adaptation continue à des conditions fluctuantes.
   Exemple : Une application IoT [5] qui adapte dynamiquement l'allocation des ressources en fonction du nombre de capteurs actifs et du volume de données à traiter.

4. Content Delivery Network : Réseau de diffusion de contenu permettant de rapprocher les ressources des utilisateurs pour améliorer la vitesse d'accès.
5. Internet of Things : Réseau d'objets physiques interconnectés capables de collecter et d'échanger des données.

1.2.4 Orchestration et Chorégraphie des Services

Pour assurer la gestion efficace de ces compositions, deux mécanismes de coordination sont généralement employés : l'orchestration et la chorégraphie.La figure 1.8 illustre les différences fondamentales entre les approches d'orchestration et de chorégraphie. L'orchestration centralise la coordination via un composant maître, tandis que la chorégraphie s'appuie sur des messages d'événements échangés entre services autonomes, souvent par l'intermédiaire d'un event broker

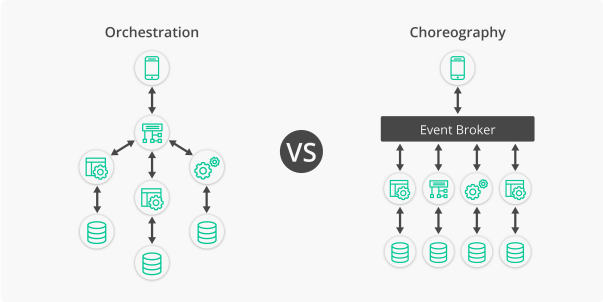

FIGURE 1.8 – Orchestration Vs Chorégraphie [6]

1. **Orchestration** [7] comme représenté dans la Figure 1.9 désigne un mode de gestion centralisé dans lequel un composant central (l'orchestrateur) supervise et coordonne les interactions entre les différents services cloud. Cela garantit une exécution contrôlée des processus et une meilleure gestion des dépendances.
   Exemple : Un orchestrateur Kubernetes [6] qui gère le déploiement et la mise à l'échelle des conteneurs dans un environnement cloud.

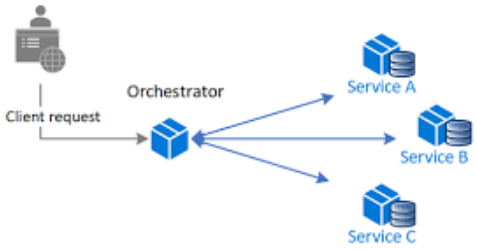

FIGURE 1.9 – Orchestration Cloud [7]

2. **Chorégraphie** [7]À l'inverse, Figure 1.10 la chorégraphie repose sur une interaction distribuée entre les services, sans supervision centralisée. Chaque service communique avec les autres de manière autonome, en respectant des règles d'interaction prédéfinies.
   Exemple : Un système de microservices où chaque service réagit aux messages qu'il reçoit et prend des décisions de manière indépendante.

6. Kubernetes : Système open-source d'orchestration de conteneurs permettant l'automatisation du déploiement, de la gestion et de la mise à l'échelle des applications.

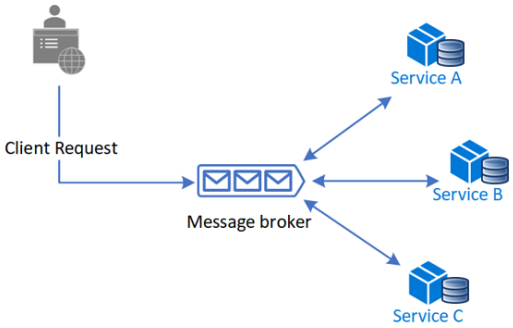

FIGURE 1.10 – Chorégraphie Cloud [7]

1.2.5 Processus de la Composition Cloud

[8]La composition cloud consiste à assembler différents services cloud afin de créer une architecture cohérente et performante. Ce processus implique plusieurs étapes essentielles pour assurer une intégration efficace des services tout en garantissant leur performance, leur sécurité et leur disponibilité.

1. **Sélection des services**
   L'identification des services nécessaires est la première étape de la composition cloud. Il s'agit de déterminer les besoins de l'application et de choisir les fournisseurs appropriés (IaaS, PaaS, SaaS) en fonction des critères de performance, de coût, de conformité réglementaire et de sécurité.
   Exemple : Une entreprise déployant une plateforme de commerce en ligne peut choisir un fournisseur de stockage cloud, un service de gestion des bases de données et une solution de mise en cache pour optimiser les temps de réponse.
2. **Orchestration des services**
   L'orchestration consiste à organiser et coordonner les interactions entre les différents composants cloud. Cela permet de garantir une exécution fluide et une automatisation efficace de processus. L'utilisation d'outils tels que Kubernetes, Terraform ou AWS CloudFormation permet de déployer et de gérer les ressources automatiquement.
   Exemple : Un système de microservices utilisant des conteneurs Docker [7] orchestrés par Kubernetes pour assurer une scalabilité dynamique.
3. **Intégration des services**
   L'intégration vise à connecter les différents services cloud entre eux à l'aide d'API [8] , de middleware ou de protocoles standardisés (REST [9], GraphQL [10], SOAP [11]). Elle assure la communication et l'interopérabilité entre les composants.
   Exemple : Une application d'analyse de données intégrant des services de stockage, d'intelligence artificielle et de visualisation de données via des API REST.
4. **Surveillance et optimisation**
   Une fois les services intégrés et fonctionnels, il est crucial de surveiller leur per-

7. Conteneurs Docker : Unité standardisée d'exécution d'une application, regroupant le code et toutes ses dépendances dans un environnement isolé et portable.
8. API : Interface de Programmation Applicative
9. REST : Representational State Transfer (Transfert d'État Représentationnel)
10. GraphQL : Graph Query Language (Langage de Requête en Graphe)
11. SOAP : Simple Object Access Protocol (Protocole Simple d'Accès aux Objets)

formance, leur consommation de ressources et leur disponibilité. Des outils comme Prometheus, Datadog ou CloudWatch permettent d'analyser les performances et d'optimiser les ressources en fonction des besoins.
Exemple : Une entreprise surveillant l'utilisation de ses ressources cloud et ajustant dynamiquement la puissance de calcul pour optimiser les coûts tout en garantissant des performances optimales.

1.2.6 Architecture du Cloud Computing

1.2.6.1 Client Cloud

L'utilisateur dans un environnement cloud, c'est la personne ou le système qui formule une demande pour utiliser des ressources ou services cloud. peur être (Une application automatisée, entreprise, client)

1.2.6.2 API -Application Programming Interface-

[8]Quand un utilisateur ou une application demande une ressource cloud (par exemple, une VM ou du stockage), cette demande passe par une interface appelée API (Application Programming Interface).
Ces APIs sont des ponts de communication entre le client (utilisateur, broker, etc.) et le fournisseur cloud.La figure 1.11 illustre le fonctionnement d'une interface de programmation d'application (API). Le client initie une requête via une API, qui est ensuite traitée par le serveur. En retour, le serveur fournit une réponse structurée, permettant une communication standardisée entre différentes entités logicielles.

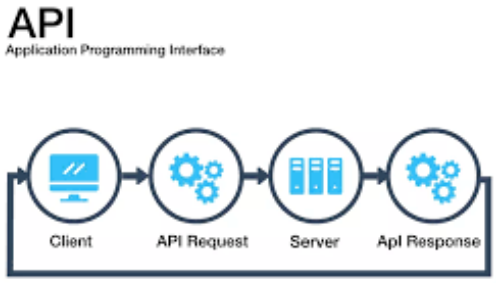

FIGURE 1.11 – Fonctionnement de l'API -Application Programming Interface- [8]

Types d'APIs utilisées dans le cloud :

1. **REST API (Representational State Transfer API) :**Basée sur le protocole HTTP (GET, POST, PUT, DELETE), très utilisée dans le webSimple, bien supportée, facile à débugger ,Lancer une VM via un simple appel POST.
2. **gRPC API (Google Remote Procedure Call API) :** rapide et efficace basée sur HTTP/2 + protocol buffers(not JSON/XML) Très rapide, compact, idéal pour microservices Gérer les communications entre services internes dans un cloud.
3. **GraphQL :**permet de demander uniquement les données nécessaires dans une requête unique Moins de surcharge, flexible ,Récupérer précisément les infos d'un utilisateur cloud ou d'un service actif.

1.2.6.3 Broker Cloud

Le broker cloud (ou courtier cloud) est un intermédiaire entre l'utilisateur et un ou plusieurs fournisseurs de services cloud.
Rôle :
- Aider l'utilisateur à choisir les meilleures offres disponibles selon ses besoins.
- Comparer, négocier ou combiner les services de différents fournisseurs.
- Gérer l'abstraction, la facturation, ou les politiques d'accès.
- Faire du matching intelligent entre la demande et les catalogues des fournisseurs.

Dans un environnement multi-cloud, on peut introduire un **broker externe (indépendant)** pour gérer les interactions entre plusieurs clouds.

1.2.6.4 Orchestrateur Cloud

un composant ou outil qui automatise et coordonne le déploiement et la gestion des ressources cloud.
Rôle :
- Déployer les applications ou services.
- Configurer la sécurité, le réseau, les pare-feux, etc.
- Superviser les performances et le cycle de vie (scaling, mise à jour...).

1.2.6.5 Fournisseur Cloud

l'entité qui possède l'infrastructure physique ou virtuelle et la met à disposition sous forme de services.
Rôle :
- Héberger les serveurs, bases de données, stockage, etc.
- Offrir des services IaaS, PaaS, SaaS.
- Gérer la disponibilité, la scalabilité, la sécurité physique.
- Appliquer les SLA [12] (garantie de service).

1.2.6.6 Structure d'une Requête Cloud Général :

– `id_utilisateur` : identifiant unique de l'utilisateur demandeur.
– `auth_token` : jeton d'authentification de l'utilisateur.
– `timestamp` : horodatage de l'envoi de la requête.
– `requested_resources` : liste des ressources demandées, avec pour chaque ressource :
  — `type` : type de service (ex. VM [13] , conteneur, base de données, etc.)
  — `os` : système d'exploitation requis (ex. Linux, Windows)
  — `cpu` : nombre de cœurs processeur requis
  — `ram` : quantité de mémoire vive demandée (en Go)
  — `storage` : capacité de stockage (en Go)
  — `region` : région géographique préférée

12. SLA : Service Level Agreements ou Accords de Niveau de Service (ANS)
13. VM : Machine Virtuelle

  - `duration` : durée d'utilisation estimée
- `constraints` : contraintes spécifiques à respecter :
  - `max_price_per_hour` : coût maximal par heure acceptable
  - `availability` : exigences en matière de disponibilité
  - `compliance` : critères de conformité réglementaire (ex. GDPR [14])
- `preferred_providers` : liste optionnelle de fournisseurs préférés
- `multi_cloud` : indicateur booléen précisant si une répartition multi-cloud est souhaitée

1.2.7 Schéma du Flux de la requête :

La figure 1.12 illustre l'ensemble des étapes impliquées dans le traitement d'une requête cloud, depuis la soumission par l'utilisateur jusqu'à l'accès effectif à la ressource. Voici le détail de chaque étape :

- **1. Soumission de la requête** : L'utilisateur cloud soumet une requête contenant ses besoins en ressources (type de VM, CPU [15], RAM [16], durée, etc.) au *Broker Cloud*.
- **2. Analyse de la requête** : Le *Broker* analyse la requête, effectue un matching avec les fournisseurs disponibles, optimise les paramètres selon les contraintes (prix, conformité, disponibilité).
- **3. Transmission optimisée** : Le *Broker* transmet ensuite la requête optimisée à l'*Orchestrateur Cloud*.
- **4–6. Provisionnement** : L'*Orchestrateur Cloud* planifie, provisionne, déploie et configure les ressources nécessaires auprès des *Fournisseurs Cloud*.
- **7. Allocation des ressources** : Le *Fournisseur Cloud* (ex. AWS, GCP, Azure) alloue les ressources physiques ou virtuelles selon les instructions de l'orchestrateur.
- **8. Retour d'état** : L'*Orchestrateur* envoie au *Broker* un état de déploiement des ressources.
- **9. Réponse à l'utilisateur** : Le *Broker* informe l'utilisateur que la ressource demandée est prête.
- **10. Accès à la ressource** : L'utilisateur peut alors accéder directement à la ressource provisionnée via le fournisseur.

14. GDPR : Règlement Général sur la Protection des Données
15. CPU : Unité Centrale de Traitement
16. RAM : (Random Access Memory) – Mémoire Vive

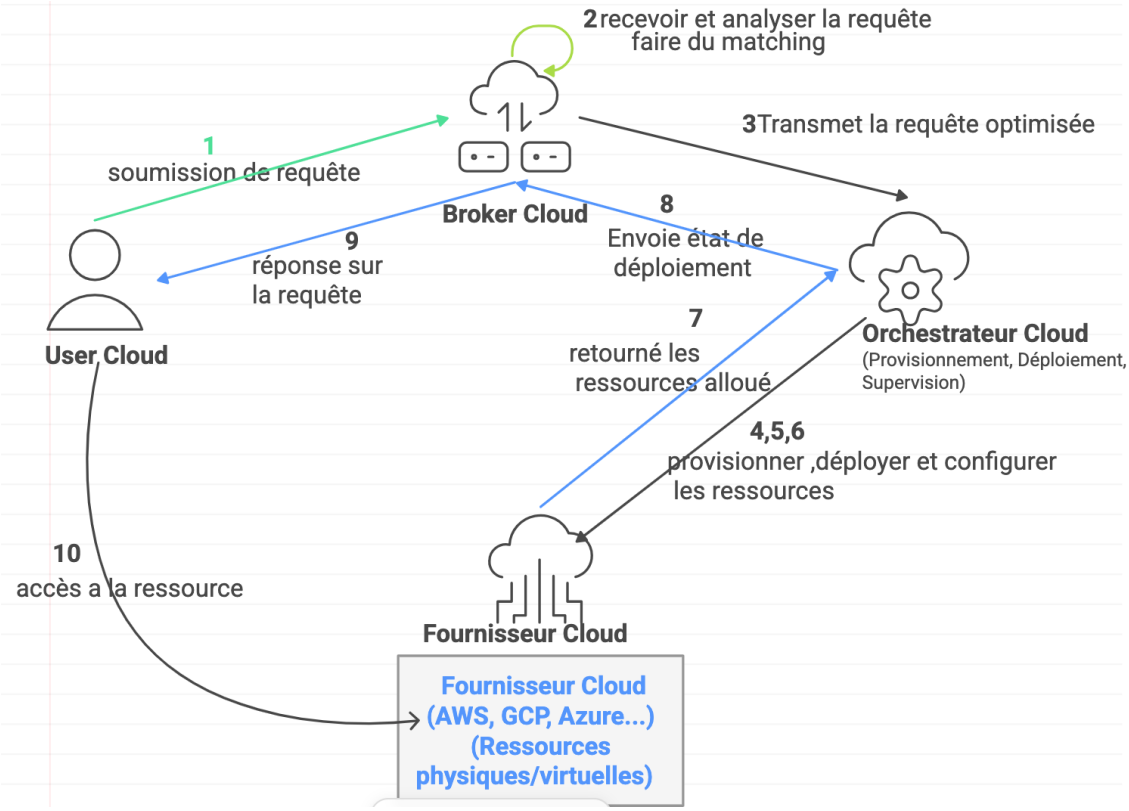

FIGURE 1.12 – Flux complet d'une requête Cloud

1.3 Technologies de la blockchain

1.3.1 Définition

[9]est une **base de données distribuée** qui enregistre des transactions de manière sécurisée et transparente. Chaque transaction est regroupée dans un bloc, qui est ensuite ajouté à une chaîne de blocs, d'où le nom "blockchain". Cette technologie repose sur des principes cryptographiques qui garantissent l'intégrité et la sécurité des données.

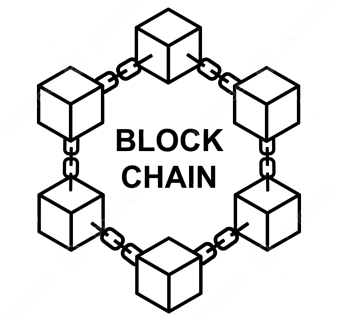

FIGURE 1.13 – Blockchain[10]

1.3.2 Types de Blockchain

1.3.2.1 Blockchain Publique

[9]sont sans permission et permettent à tout le monde de les rejoindre. Tous les membres de la Blockchain ont des droits égaux pour lire, modifier et valider la Blockchain.
Les gens utilisent principalement les Blockchains publiques pour échanger et miner des crypto-monnaies comme le Bitcoin, l'Ethereum et le Litecoin.

1.3.2.2 Blockchain Privée

[9]Une seule organisation contrôle les Blockchains privées, également appelées Blockchains gérées. L'autorité détermine qui peut être membre et quels droits ils ont dans le réseau.
Les Blockchains privées ne sont que partiellement décentralisées, car elles ont des restrictions d'accès. Ripple, un réseau d'échange de devises numériques pour les entreprises, est un exemple de Blockchain privée..

1.3.2.3 Blockchain Hybride

[9]combinent des éléments des réseaux privés et publics. Les entreprises peuvent mettre en place des systèmes privés, basés sur les autorisations, parallèlement à un système public. De cette façon, ils contrôlent l'accès à des données spécifiques stockées dans la Blockchain tout en gardant le reste des données publiques. Ils utilisent des contrats intelligents pour permettre aux membres publics de vérifier si les transactions privées ont été effectuées.
Par exemple, les Blockchains hybrides peuvent accorder un accès public à la monnaie numérique tout en gardant la monnaie appartenant à la banque privée.

1.3.2.4 Différence entre une base de données et une Blockchain

La Blockchain est un type particulier de système de gestion de base de données qui possède plus de fonctionnalités qu'une base de données ordinaire.

Liste quelques différences significatives entre une base de données traditionnelle et une Blockchain :[9]

1. Les Blockchains décentralisent le contrôle sans porter atteinte à la confiance dans les données existantes. Cela n'est pas possible dans d'autres systèmes de base de données.
2. Les entreprises impliquées dans une transaction ne peuvent pas partager l'intégralité de leur base de données. Mais dans les réseaux Blockchain, chaque entreprise possède sa copie du grand livre, et le système maintient automatiquement la cohérence entre les deux grands livres.

3. Alors que dans la plupart des systèmes de base de données, vous pouvez modifier ou supprimer des données, dans la Blockchain, vous ne pouvez qu'insérer des données.

1.3.3 Composants clés de la technologie Blockchain

1.3.3.1 Grand livre distribué :

[9]est la base de données partagée du réseau Blockchain qui stocke les transactions, comme un fichier partagé que tous les membres de l'équipe peuvent modifier.
Dans la plupart des éditeurs de texte partagés, toute personne disposant de droits d'édition peut supprimer l'ensemble du fichier. Cependant, les technologies de grands livres distribués ont des règles strictes concernant qui peut modifier et de quelle manière. Vous ne pouvez pas supprimer les entrées une fois qu'elles ont été enregistrées..

1.3.3.2 Contrats intelligents "Smart contract" :

[9] Il s'agit de programmes stockés sur le système Blockchain qui s'exécutent automatiquement lorsque des conditions prédéterminées sont remplies. Ils exécutent des vérifications « si-alors » afin que les transactions puissent être effectuées en toute confiance .

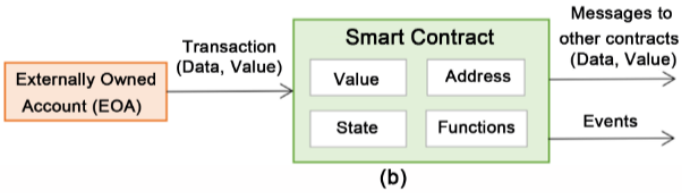

FIGURE 1.14 – Structure du smart contract [11]

1.3.3.3 cryptographie à clé publique

est un dispositif de sécurité permettant d'identifier de manière unique les participants au réseau Blockchain. Ce mécanisme génère deux jeux de clés pour les membres du réseau.
une clé publique qui est commune à tous les membres du réseau. L'autre est une clé privée qui est unique pour chaque membre. Les clés privées et publiques fonctionnent ensemble pour **déverrouiller** les données du grand livre.

1.3.3.4 Consensus :

est un système Blockchain qui établit des règles concernant le consentement des participants pour l'enregistrement des transactions.Ce sont ce que l'on appelle des nœuds distribués qui sont régis par un algorithme de consensus qui vont devoir se mettre d'accord pour valider une transaction."Vous ne pouvez enregistrer de nouvelles transactions que lorsque la majorité des participants du réseau donnent leur accord."
L'algorithme de consensus a donc pour rôle de s'assurer que les règles ont bien été respectées.

Différents types d'algorithmes de consensus :

1. **Preuve de Travail (PoW) :**[12] C'est le premier algorithme de consensus utilisé par Satoshi Nakamoto, pour la création de Bitcoin. il repose sur une fonction de hachage. Elle va calculer une empreinte unique nécessaire à la validation du bloc à partir des données fournies. Ce défi est difficile à résoudre mais facile à vérifier, garantissant ainsi la sécurité et l'intégrité du réseau.
   [13]Toute modification d'un bloc nécessiterait de recalculer la preuve de travail de tous les blocs suivants, ce qui demanderait une puissance de calcul colossale. Cependant, la PoW est **énergivore** et peut mener à une diminution du nombre de mineurs, rendant ainsi la blockchain vulnérable à une attaque des 51/100.
   Cette attaque survient lorsqu'un acteur malveillant contrôle plus de 51/100 de la puissance de calcul du réseau, lui permettant ainsi de falsifier les transactions.

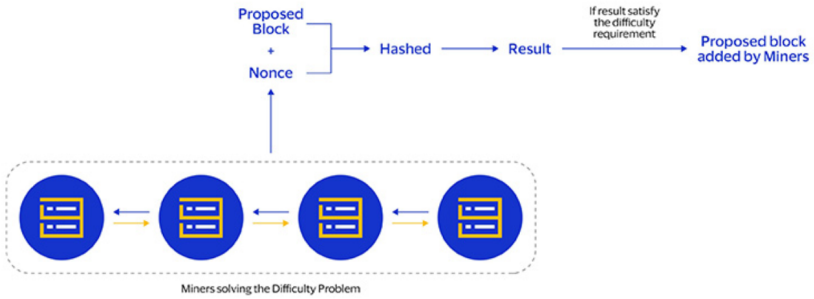

FIGURE 1.15 – Processus de Preuve de Travail (PoW) [14]

2. **Preuve d'Enjeu (PoS) :** [12]est une méthode alternative au PoW en supprimant notamment le concept de minage physique pour opter pour du minage virtuel qui est notamment moins énergivore et donc plus écologique. Dans le cas des crypto-monnaies, le PoS choisit de manière aléatoire parmi les utilisateurs éligibles celui qui aura le droit de valider le bloc.

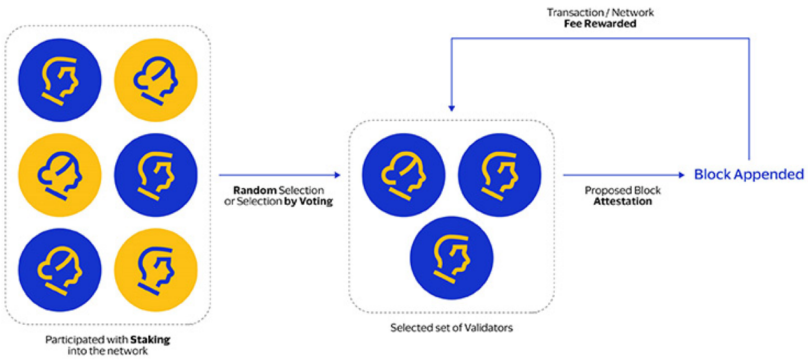

FIGURE 1.16 – Processus de Preuve d'Enjeu (PoS) [14]

3. **Preuve d'autorité (PoA) :** [15] a été proposé en 2017 par Gavin Wood, co-fondateur d'Ethereum. la preuve d'autorité ne s'effectue pas sur la puissance de

calcul, ou encore le nombre de jetons qu'un nœud possède, mais bien sur la réputation des validateurs, que nous appelons "entités de confiance".
[16] La validation des blocs est effectuée par les validateurs qui ont été autoritairement sélectionnés.ce type de consensus est souvent utilisé et apprécié sur des réseaux privés , avec PoA, seul un petit nombre de noeuds ont le pouvoir de participer à l'élection des nouveaux blocs.

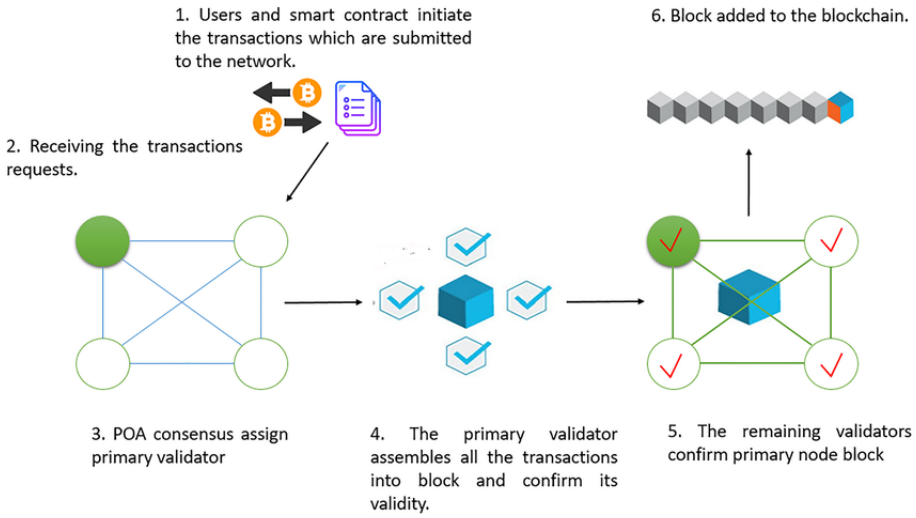

FIGURE 1.17 – Processus de Preuve d'autorité (PoA) [17]

1.3.4 Architecture de la Blockchain

1.3.4.1 En-tête du Bloc :

1. **Hash du bloc précédent :** Assure le chaînage des blocs dans la Blockchain. Par exemple,le bloc 31 fait référence au bloc 30.
   Le premier bloc d'une chaîne, appelé Genesis Block,est unique, car tous les blocs validés en découlent.
2. **Index :** Numéro d'identification du bloc dans la chaîne.
3. **Horodatage :** Mécanisme attribuant une date et une heure à chaque transaction et bloc,permettant dâenregistrer précisément le moment de chaque opération.
4. **Compte ou portefeuille (wallet) :** Logiciel permettant de stocker les clés privée et publique dâun utilisateur de Blockchain. Il peut être une application web, mobile ou un logiciel sur ordinateur, et sert à gérer les crypto-monnaies, consulter le solde et effectuer des transactions.
5. **Hash du bloc :** Valeur hexadécimale générée à partir des données du bloc, jouant le rôle dâempreinte numérique unique. Elle permet dâidentifier rapidement un bloc. La fonction de hachage la plus utilisée est SHA-256.

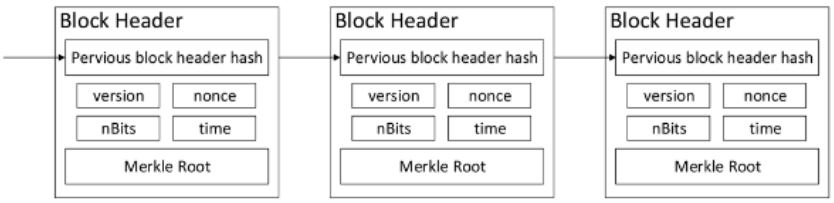

FIGURE 1.18 – Structure de donnée d'un block [18]

1.3.4.2 Transactions :

Une transaction est l'unité fondamentale d'un système Blockchain. Elle contient une adresse de destinataire, une adresse d'expéditeur et des données. Son fonctionnement repose sur la cryptographie asymétrique.
Lorsqu'un utilisateur initie une transaction, son portefeuille génère une clé publique et une clé privée. La clé privée reste strictement confidentielle, tandis que la clé publique est accessible à tous. Avant d'être envoyée, la transaction doit être signée numériquement.
Cette signature est un nombre calculé à partir de la clé privée de l'expéditeur, sans jamais être divulguée. La validité de la signature peut ensuite être vérifiée à l'aide de la clé publique correspondante.

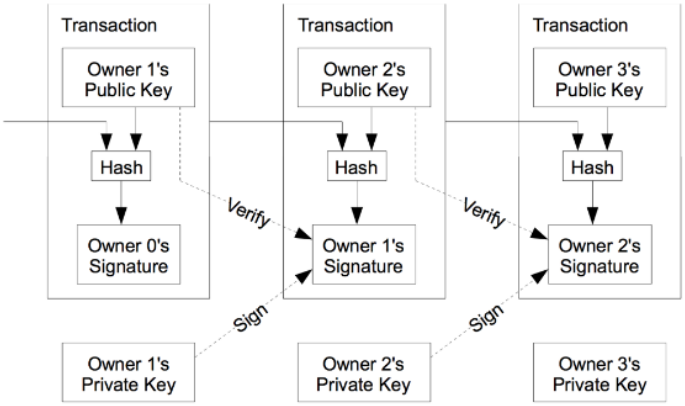

FIGURE 1.19 – Structure de donnée d'une transaction [18]

1.3.4.3 Fonctions de hachage :

[2] Les algorithmes de hash transforment n'importe quelle quantité de données en hash numérique ou empreinte de longueur fixe. Il est impossible pour un hacker d'inverser un hash numérique pour découvrir l'entrée d'origine. Si la saisie subit une quelconque modification, cela se traduit par un hash différent.
On distingue deux principales approches de hachage :
- Le hachage simple
- Le hachage arborescent, basé sur l'arbre de Merkle

La Blockchain utilise principalement le hachage arborescent, qui repose sur l'arbre de Merkle pour organiser et vérifier efficacement un grand nombre de transactions. Cette structure permet d'assurer l'intégrité des blocs et de faciliter leur validation rapide et

sécurisée.

Fonctionnement d'un Arbre de Merkle :

[19] Merkle Tree ("Arbre de Merkle") est une méthode permettant de structurer des données en vue d'y accéder et d'en vérifier la véracité plus rapidement. En effet, les transactions sont regroupées par groupe de deux, un hachage est ensuite applique à ce groupe. Les groupes sont ensuite regroupes par groupe de deux puis soumis au même procédé jusqu'au dernier hachage appelé la racine ("Merkle Root") qui lui est ajoute comme référence dans le "Header" du bloc.

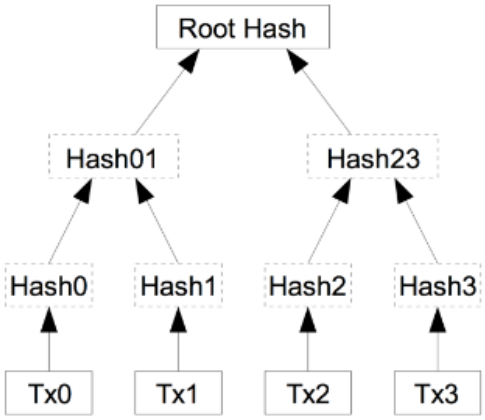

FIGURE 1.20 – Arbre de Merkle [19]

1.3.4.4 Chiffrement asymétrique - Confidentialité-

[2]Les algorithmes asymétriques assurent la confidentialité sans partager de mot de passe au préalable. La confidentialité des algorithmes asymétriques est garantie quand vous lancez le processus de chiffrement avec la clé publique.

Le processus peut être résumé à l'aide de la formule :

Clé publique (chiffrement) + Clé privée (déchiffrement) = Confidentialité

Là où la clé publique sert à chiffrer les données, la clé privée doit être utilisée pour les déchiffrer. Un seul hôte possède la clé privée , par conséquent, la confidentialité est assurée.

Si la clé privée est compromise, une autre paire de clés doit être générée pour remplacer la clé compromise.

1.3.4.5 Signatures Numériques

[2] est une technique mathématique utilisée pour assurer l'authenticité, l'intégrité et la non-répudiation. Les signatures numériques possèdent des propriétés spécifiques qui assurent l'authentification des entités et l'intégrité des données. En outre, les signatures numériques attestent de la non-répudiation de la transaction. En d'autres termes, la signature numérique constitue une preuve légale de l'échange de données. La cryptographie

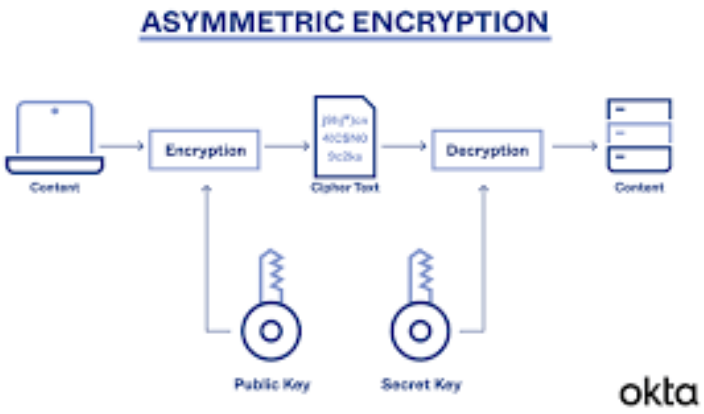

FIGURE 1.21 – Chiffrement asymétrique[19]

asymétrique est à la base des signatures numériques .

Signatures numériques pour les certificats numériques : [2]Un certificat numérique est l'équivalent d'un passeport électronique. Il permet aux utilisateurs, hôtes et entreprises d'échanger des informations sur Internet de manière sécurisée. Concrètement, un certificat numérique permet d'authentifier et de vérifier que l'expéditeur d'un message est bien celui qu'il prétend être. Les certificats numériques permettent également de garantir la confidentialité du destinataire en lui permettant de chiffrer sa réponse.

Fonctionnement des signatures numériques :[20]

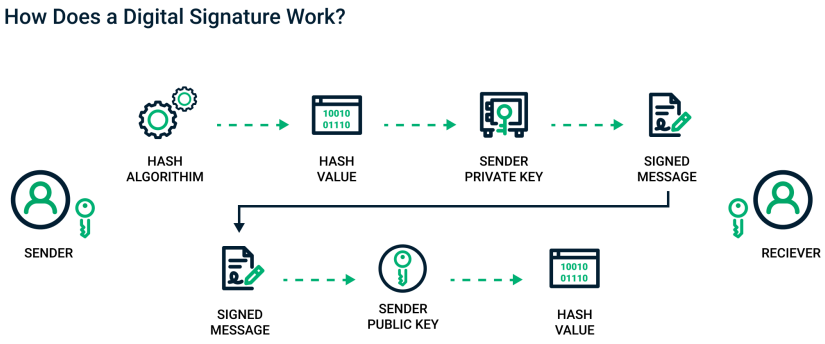

FIGURE 1.22 – Processus de signature numérique [20]

1. L'expéditeur sélectionne le fichier à signer numériquement .
2. Calculer la valeur de hachage unique du contenu du fichier .
3. Cette valeur de hachage est cryptée avec la clé privée de l'expéditeur pour créer la signature numérique.
4. Le fichier original accompagné de sa signature numérique est envoyé au destinataire.
5. Le destinataire décrypte ensuite la signature numérique à l'aide de la clé publique de l'expéditeur.

A fin de vérifier la signature : Le destinataire calcule alors le hachage du fichier original et le compare au hachage décrypté du fichier de l'expéditeur.

1.3.5 Minage

1.3.5.1 Définition :

[21] « mining » en anglais, est un terme utilisé pour décrire le processus de validation des transactions qui attendent d'être ajoutées à la base de données de la blockchain. Le minage est essentiel sur les blockchains de type Proof of Work (preuve de travail) comme celle du Bitcoin.
Les blockchains plus récentes ont tendance à utiliser la Proof of Stake (preuve d'enjeu) et d'autres mécanismes de consensus, et elles ne nécessitent ni ne permettent le minage.
Le minage est tout simplement **la création d'un bloc valide par un des membres du réseau.**

1.3.5.2 Processus du minage

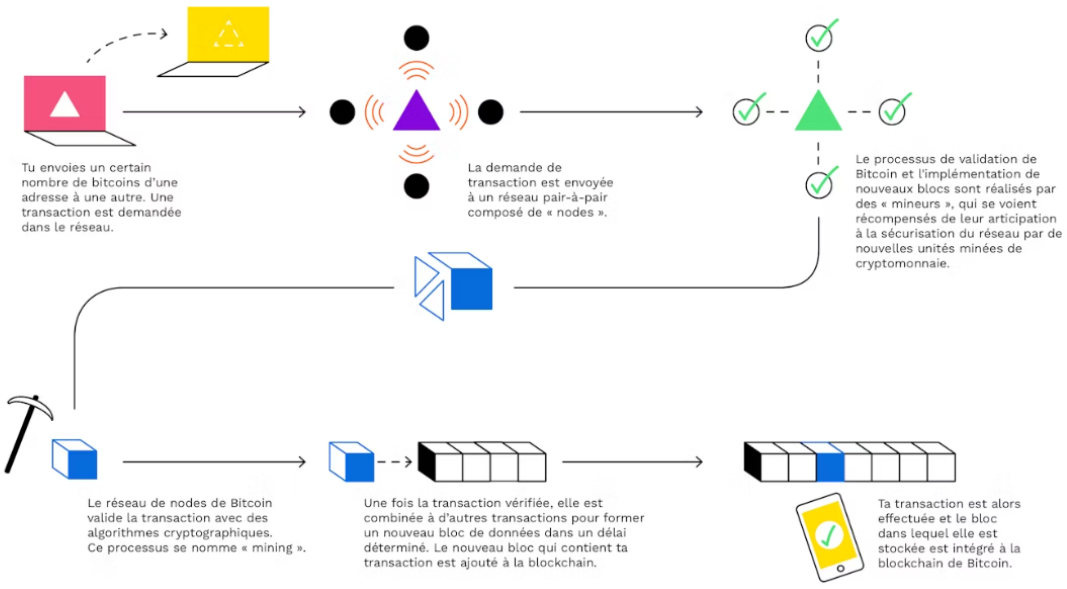

FIGURE 1.23 – Processus du minage [22]

1. **Création d'une transaction :**
   Un utilisateur déclenche une transaction à partir de son application de portefeuille, en essayant d'envoyer une crypto-monnaie ou un jeton à quelqu'un d'autre .
2. **Diffusion de la transaction sur le réseau Blockchain :**
   [21]La transaction est diffusée sur le réseau par l'application de portefeuille. Elle attend maintenant d'être récupérée par un mineur de la blockchain pour être intégrée à un bloc. Tant qu'elle n'est pas récupérée, elle est placée dans un «pool de transactions non confirmées». Ce pool est un ensemble de transactions non confirmées sur le réseau, en attente de traitement .
3. **Création du nouveau block :**
   [21]En sélectionnant des transactions et en les ajoutant à leur bloc, les mineurs

créent un bloc de transactions. Pour ajouter ce bloc de transactions à la blockchain (pour que tous les autres mineurs et nœuds enregistrent les transactions), le bloc doit tout d'abord obtenir une signature (également appelée preuve de travail). Cette signature est créée en résolvant un problème mathématique très complexe, unique à chaque bloc de transactions.
Le mineur qui trouve une signature éligible pour son bloc en premier, diffuse ce bloc et sa signature à tous les autres mineurs.

4. **Vérification et validation du block :**
[21]D'autres mineurs vérifient maintenant la légitimité de la signature. Ils prennent la chaîne de données du bloc diffusé et l'analysent pour voir si le hachage de sortie correspond bien à la signature incluse.
S'il est valide, les autres mineurs confirmeront sa validité et conviendront que le bloc peut être ajouté à la blockchain. Ils doivent parvenir à un consensus, c'est-à-dire qu'ils doivent être tous d'accord.Et la signature est la «preuve» du travail effectué. Cela correspond à la puissance de calcul dépensée.

5. **Mise a jour de Blockchain :**
[21]Après l'ajout de ce bloc à la chaîne, Les autres nœuds acceptent le bloc et l'enregistreront dans leurs données de transaction.

Chapitre 2

État de l'art : étude des cas existants

2.1 Introduction

Ce chapitre dresse un état de l'art des approches actuelles en matière de contrôle d'accès basé sur la blockchain dans les environnements cloud. Après une présentation des modèles existants (BMAC [1],CBFF [2],BC-ABAC [3] , etc.), nous examinerons les mécanismes qu'ils utilisent, leurs avantages, ainsi que leurs limites dans un contexte multi-cloud. Cette analyse critique servira de base pour identifier les lacunes actuelles et positionner la contribution proposée dans ce mémoire.

2.2 Sécurité dans le cloud

[23]Le cloud computing repose sur la mutualisation et la virtualisation des ressources. Bien qu'il offre flexibilité et réduction des coûts, il introduit aussi des défis majeurs en matière de sécurité. Les données sont hébergées hors du périmètre de l'organisation, partagées sur des infrastructures multi-tenant, et gérées par des tiers

2.2.1 Menaces principales

Dans un environnement cloud multi-tenant, où plusieurs utilisateurs ou organisations partagent les mêmes ressources physiques et logiques, les menaces pesant sur la sécurité et la confidentialité des données sont nombreuses. Voici un aperçu des principales menaces :

— **Fuites de données entre locataires**
Dans une architecture cloud partagée, une isolation insuffisante entre les locataires peut provoquer des fuites de données sensibles. Cela peut se produire lorsque :

— Les machines virtuelles (VM) [4] d'un locataire parviennent à accéder indirectement aux ressources d'un autre locataire par le biais de canaux cachés (side-channels).

1. Blockchain Multi-Authority Control (BMAC) : Modèle de contrôle d'accès basé sur la blockchain avec plusieurs autorités de décision.

2. Chain-Based Federated Framework (CBFF) : Cadre fédéré basé sur la blockchain permettant la gestion décentralisée des autorisations d'accès.

3. Blockchain-Enabled Attribute-Based Access Control (BC-ABAC) : Contrôle d'accès basé sur les attributs, renforcé par la blockchain pour garantir la transparence et la traçabilité des décisions.

4. Virtual Machine : Machine virtuelle simulant un ordinateur physique.

  - Des erreurs de programmation ou de configuration permettent à un locataire malveillant d'extraire des informations d'un espace mémoire ou d'un stockage commun.
  - Des vulnérabilités dans les hyperviseurs ou les conteneurs compromettent la séparation entre les environnements d'exécution.

- **Mauvaises configurations des services**
  La mauvaise configuration des services cloud est l'une des causes les plus fréquentes d'incidents de sécurité. Cela inclut :
  - Des permissions excessives accordées aux utilisateurs ou applications.
  - L'exposition publique de bases de données, de serveurs ou d'interfaces de gestion.
  - Le déploiement de politiques de sécurité faibles ou incomplètes.

- **Accès non autorisés (internes ou externes)**
  Les accès non autorisés peuvent provenir de :
  - D'attaquants externes qui exploitent des vulnérabilités dans les interfaces du cloud ou dans les applications déployées.
  - De personnes internes (employés, sous-traitants) qui utilisent leur accès privilégié à des fins malveillantes ou par négligence.
  - De comptes compromis (phishing, mots de passe faibles, token volé).

- **Manque de transparence et de contrôle** Le manque de transparence des fournisseurs de services cloud est un vrai sujet de préoccupation. Souvent, ils ne fournissent pas une image claire des actions qu'ils entreprennent concernant les données de leurs clients. Voici quelques points où cette opacité se manifeste :
  - Les traitements automatisés appliqués aux données, comme l'indexation et l'analyse.
  - L'emplacement réel où les données sont stockées, ce qui soulève des questions de souveraineté.
  - Les politiques relatives à la sauvegarde, à la journalisation et à la destruction des données.
  - Les accès privilégiés dont disposent leurs administrateurs internes.

2.2.2 Mesures de sécurité mises en œuvre

Pour assurer la sécurité des données et des services dans les environnements cloud, les fournisseurs déploient un ensemble de mesures solides et complémentaires, parmi lesquelles :

- **Isolation des environnements via machines virtuelles ou conteneurs**
  Les fournisseurs cloud recourent à des technologies de virtualisation et de conteneurisation pour séparer les environnements utilisateurs. Chaque machine virtuelle (VM) ou conteneur fonctionne de manière indépendante, ce qui empêche qu'une faille dans un environnement puisse avoir des répercussions sur les autres. Cette séparation garantit que les ressources informatiques sont partagées sans compromettre la sécurité ou la confidentialité des données entre différents clients.

— **Chiffrement des données, en transit et au repos**

Pour protéger les données sensibles, les fournisseurs mettent en œuvre des mécanismes de chiffrement robustes à plusieurs niveaux. Les données stockées (« au repos ») sont chiffrées à l'aide d'algorithmes standards (comme AES-256), ce qui empêche tout accès non autorisé en cas de compromission du support physique. De plus, les données échangées entre les utilisateurs, les applications et les serveurs cloud sont sécurisées par des protocoles cryptographiques (TLS/SSL) qui garantissent la confidentialité et l'intégrité des informations lors de leur transfert.

— **Gestion des identités et des accès (IAM)** [5]

La gestion des identités et des accès est cruciale pour déterminer qui peut accéder à quelles ressources. Les fournisseurs mettent en place des solutions IAM avancées qui permettent d'authentifier les utilisateurs par des méthodes sécurisées (authentification multi-facteurs, SSO, etc.) et d'appliquer des politiques strictes de contrôle d'accès basées sur les rôles ou les attributs. Ces mécanismes réduisent les risques d'accès non autorisés et facilitent la gestion des droits dans des environnements complexes.

— **Journalisation et auditabilité des opérations**

Pour garantir la traçabilité des actions effectuées dans le cloud, toutes les opérations critiques sont soigneusement consignées dans des journaux d'audit (logs). Ces journaux jouent un rôle essentiel en permettant de repérer et d'analyser les comportements suspects, de vérifier la conformité des accès et des modifications, et de réagir rapidement aux incidents de sécurité. L'intégration d'outils d'analyse et de surveillance en temps réel renforce cette approche de contrôle.

— **Conformité à des normes et certifications reconnues**

Pour assurer un niveau de sécurité qui respecte les exigences réglementaires et industrielles, les fournisseurs de services cloud adoptent des référentiels standards comme ISO/IEC 27017 (meilleures pratiques pour la sécurité dans le cloud), ISO/IEC 27001 (système de gestion de la sécurité de l'information), ou encore SOC 2 (audit des contrôles liés à la sécurité, la disponibilité, l'intégrité, la confidentialité et la protection des données). Ces certifications nécessitent des audits réguliers et un engagement formel à maintenir des mesures de sécurité strictes.

2.2.3 Modèle de responsabilité partagée

La sécurité dans le cloud est répartie entre :

Le fournisseur responsable de l'infrastructure

Le client responsable de ses données, accès, et configurations

2.3 Typologie des modèles de sécurité

Le contrôle de la sécurité dans les systèmes informatiques repose généralement sur deux grandes approches : la sécurité centralisée et la sécurité décentralisée. Chacune répond à des objectifs différents, en fonction du contexte, de l'environnement réseau, et des exigences de gouvernance.

5. Identity and Access Management : Gestion des identités et des accès.

2.3.1 Sécurité centralisée

[24] Dans les architectures centralisées, une autorité unique est responsable de la gestion des identités, des politiques d'accès, de la surveillance et des décisions de sécurité. Ce modèle est simple à mettre en œuvre et adapté à des environnements homogènes.

2.3.1.1 Avantages :

— Administration centralisée simplifiée

— Contrôle direct des politiques et des utilisateurs

— Facilité d'intégration avec les systèmes traditionnels

2.3.1.2 Limites :

— Point de défaillance unique

— Moins de résilience en cas d'attaque sur l'autorité centrale

— Manque de flexibilité dans des environnements multi-organisationnels ou multi-cloud

— Risque accru de conflits d'intérêts si l'autorité est compromise

2.3.2 Sécurité décentralisée

[25]La sécurité décentralisée répartit la prise de décision entre plusieurs entités autonomes, sans dépendre d'une autorité centrale unique. Cette approche est de plus en plus utilisée dans les systèmes distribués, multi-cloud, ou inter-organisationnels.

L'émergence de la blockchain renforce ce paradigme en introduisant :

— Un registre immuable et distribué (Distributed Ledger) [6]

— Des contrats intelligents (Smart Contracts) pour exécuter automatiquement des règles de sécurité

— Une traçabilité vérifiable des accès et des décisions

2.3.2.1 Avantages :

— Élimine les points uniques de défaillance

— Favorise l'interopérabilité entre entités indépendantes

— Renforce la transparence et l'auditabilité

2.3.2.2 Limites :

— Complexité d'intégration entre plusieurs systèmes hétérogènes

— Besoin d'une normalisation sémantique (vocabulaire, formats de politiques)

— Moins de maturité dans les outils et les standards

6. Grand Livre Distribué (Distributed Ledger) : Base de données numérique partagée, synchronisée entre plusieurs sites ou participants, permettant un enregistrement sécurisé et immuable des transactions sans autorité centrale.

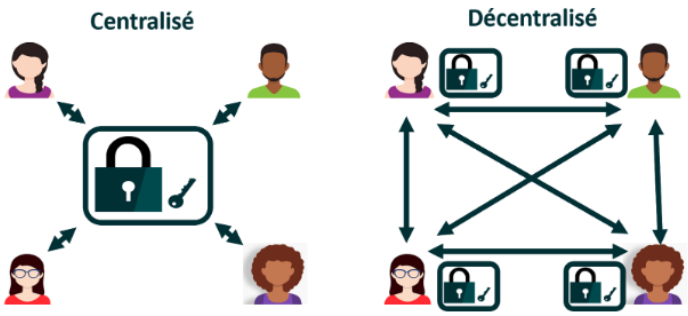

FIGURE 2.1 – Modéles de sécurité

2.4 Contrôle d'accès

2.4.1 Définition

[26]Une méthode ou un ensemble de politiques qui déterminent si un utilisateur ou une entité peut accéder à des ressources ou des services spécifiques. Il vise principalement à protéger les ressources du cloud contre l'accès non autorisé et les activités malveillantes, en restreignant les actions des utilisateurs en fonction de certains paramètres de confiance et de conformité aux politiques.

Plus précisément, le modèle proposé s'appuie sur une évaluation préalable de la confiance de l'utilisateur et des ressources cloud, où l'accès est accordé uniquement si ces scores de confiance dépassent des seuils prédéfinis. Ce contrôle d'accès dynamique et basé sur la confiance permet d'assurer une meilleure sécurité dans les environnements cloud en vérifiant à la fois l'identité et la fiabilité des utilisateurs et des ressources.

2.4.2 Composants d'un système de contrôle d'accès

Les composants d'un système de contrôle d'accès cloud interagissent pour assurer la sécurité et la gestion des accès.

2.4.2.1 Policy Enforcement Point (PEP)

Le Policy Enforcement Point (PEP) représente le composant opérationnel du système où toutes les demandes d'accès sont initialement reçues et traitées. Il a pour responsabilité principale d'appliquer les décisions d'autorisation transmises par le PDP, tout en assurant le blocage systématique de toute tentative d'accès non autorisée. Son rôle crucial en fait la première ligne de défense du système de contrôle d'accès.

2.4.2.2 Policy Decision Point (PDP)

Fonctionnant comme le cerveau décisionnel du système, le Policy Decision Point (PDP) effectue une analyse approfondie de chaque demande d'accès. Il consulte l'ensemble des politiques de sécurité en vigueur et prend la décision finale concernant l'autorisation ou le refus d'accès. Une particularité essentielle de son fonctionnement réside dans l'intégration dynamique des scores de confiance dans son processus d'évaluation, permettant une prise de décision contextuelle et adaptative.

2.4.2.3 Policy Information Point (PIP)

Le Policy Information Point (PIP) joue un rôle central dans la collecte et la distribution des données nécessaires au processus décisionnel. Il rassemble en permanence les attributs relatifs aux utilisateurs et aux ressources, tout en assurant la transmission efficace des métadonnées de confiance vers le PDP. Son fonctionnement garantit que les décisions d'accès sont prises sur la base d'informations contextuelles complètes et à jour.

2.4.2.4 Policy Administration Point (PAP)

Le PAP, ou Point d'Administration des Politiques, est un élément crucial dans les systèmes de contrôle d'accès. Il est responsable de l'élaboration, de la gestion et de la révision des politiques de sécurité qui régulent l'accès aux ressources. En pratique, le PAP offre aux administrateurs la possibilité d'établir les règles d'autorisation selon divers paramètres tels que l'identité de l'utilisateur, le contexte de la requête ou le type de ressource. Ces politiques sont par la suite mises en œuvre par d'autres éléments du système, y compris le PDP (Point de Prise de Décision de Politique), qui se sert de ces règles pour déterminer les décisions d'accès. Le PAP a donc une importance cruciale pour s'assurer que le système met en œuvre des politiques qui sont cohérentes et répondent aux exigences de sécurité de l'organisation.

2.4.3 Processus d'interaction

La Figure 2.2 montre comment les composants interagissent selon la séquence suivante :

1. Le PEP intercepte la demande d'accès
2. Le PDP consulte le PIP et PAP
3. Évaluation des politiques et scores de confiance
4. Décision finale renvoyée au PEP
5. Application de l'accès/refus

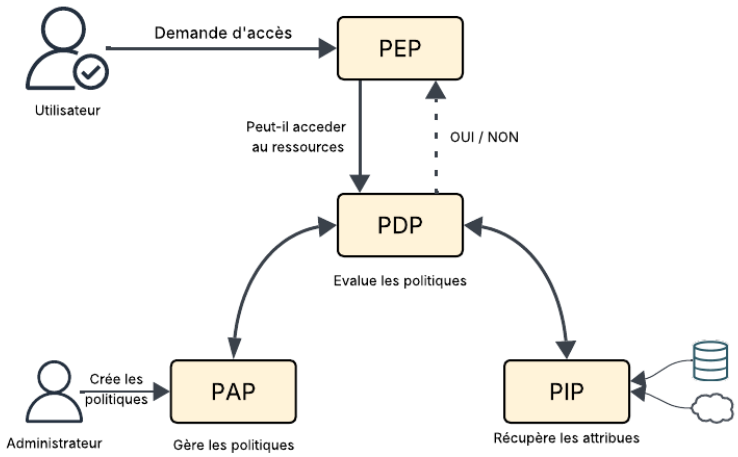

FIGURE 2.2 – Processus d'un système de contrôle d'accès cloud

2.4.4 Modèles de politiques d'accès

Différents modèles sont utilisés selon le contexte d'application :

2.4.4.1 RBAC (Role-Based Access Control)

[27]Le modèle *Role-Based Access Control* (RBAC) est un modèle de politique d'accès où les droits d'accès sont fournis aux utilisateurs en fonction des rôles et non des utilisateurs eux-mêmes. Malgré l'efficacité du contrôle hiérarchique des droits, il a été contesté en ce qui concerne la flexibilité de l'adaptation à un certain environnement . Puisque cette politique est stricte avec une majeure architecture bien construite ou une entrée autorisée, elle n'est pas adaptée aux nouvelles entreprises/startups.

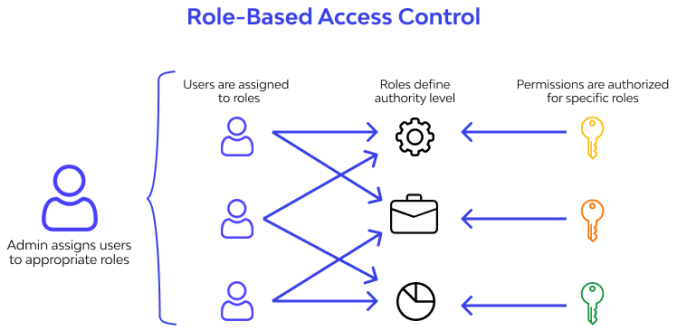

FIGURE 2.3 – RBAC (Role-Based Access Control)[28]

2.4.4.2 ABAC (Attribute-Based Access Control)

[29]L'*Attribute-Based Access Control* (ABAC) prend des décisions en fonction d'une évaluation dynamique de caractéristiques (utilisateur, ressource, environnement). Ce degré d'abstraction rend la modélisation décentralisée et dynamique plus facile . XACML en est une implémentation bien connue.

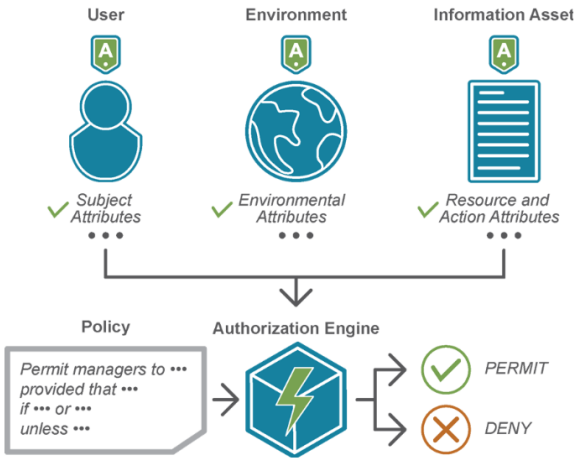

FIGURE 2.4 – ABAC (Attribute-Based Access Control)[30]

2.4.4.3 MAC (Mandatory Access Control)

Le *Mandatory Access Control* (MAC) opère sous un paradigme centralisé, c'est-à-dire que les règles sont définies par une autorité unique et immuables par les utilisateurs finaux. Ce prototype est ici très strictement utilisé pour les contextes militaires et des systèmes critiques, où la sécurité prime sur la flexibilité.

2.4.4.4 PBAC (Policy-Based Access Control)

Approche émergente, le *Policy-Based Access Control* (PBAC) met en œuvre des systèmes de règles dynamiques et adaptatives, particulièrement adaptés aux architectures cloud récentes. L'utilisation du langage XACML pour son application permet une gestion précise des politiques de sécurité.

Critères de sélection Le choix d'un modèle spécifique dépend de trois facteurs clés :

— L'environnement technique d'implémentation

— Le niveau de sécurité requis

— Le degré de flexibilité opérationnelle nécessaire

Ces modèles fondamentaux constituent la base des approches hybrides et avancées que nous détaillerons dans les sections suivantes.

2.4.5 Demande d'accès dans une composition clouds

La figure 2.5 Démontre le processus de gestion des requêtes d'accès dans une structure multi-cloud fondée sur une assemblage de services autonomes. Ce mécanisme est

central dans les environnements décentralisés où divers prestataires de services en cloud travaillent ensemble sans une instance centrale d'autorité.

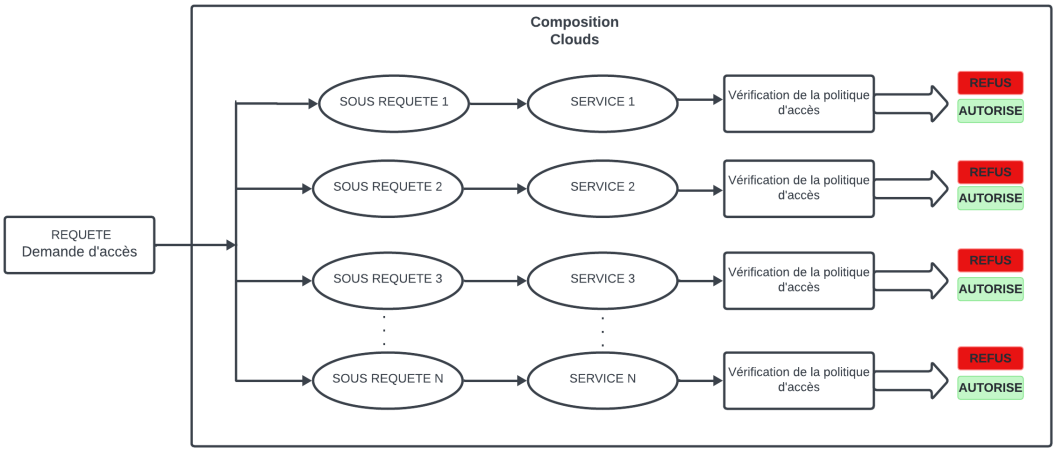

FIGURE 2.5 – Processus de gestion des demandes d'accès dans une architecture multi-cloud

2.4.5.1 Requête Initiale

L'utilisateur (ou une application) initie une demande d'accès globale. Celle-ci correspond généralement à une opération composite (ex. : affichage d'un tableau de bord, exécution d'un workflow, accès à des données réparties) nécessitant des interactions avec plusieurs services cloud.

2.4.5.2 Décomposition en Sous-Requêtes

La requête est automatiquement décomposée en sous-requêtes, chacune étant adressée à un service spécifique dans un cloud donné. Cette étape est réalisée par un composant orchestrateur, qui identifie les ressources ciblées et génère les sous-requêtes appropriées.

2.4.5.3 Traitement Local par les Services

Chaque service cloud effectue :

1. **Réception de la sous-requête**
   Analyse du contenu et vérification de l'intégrité du message
2. **Vérification par le PDP [7] local**
   Consultation des politiques de sécurité et évaluation du contexte
3. **Modèles de contrôle appliqués**
   - RBAC (Contrôle par rôles)
   - ABAC (Contrôle par attributs)
   - PBAC (Contrôle par politiques)

7. Policy Decision Point (PDP) : Composant clé d'une architecture de contrôle d'accès qui évalue les politiques et décide si une requête d'accès doit être acceptée ou refusée.

4. **Décision finale**
   AUTORISÉ : Requête conforme aux politiques
   REFUSÉ : Non-respect des règles

2.4.5.4 Production des Réponses

Les décisions locales sont ensuite retournées individuellement à l'orchestrateur central. Ce dernier peut être un composant classique (broker) ou décentralisé

2.4.5.5 Enjeux de Coordination

L'agrégation de ces décisions soulève plusieurs défis :
Que faire si certains services autorisent l'accès, mais d'autres le refusent ?
L'un des services est-il critiquement nécessaire à l'exécution globale ?
L'absence de standardisation entre les politiques nécessite-t-elle une normalisation préalable ?
Ces considérations sont traitées dans les étapes suivantes,

2.5 Contrôle d'accès & blockchains

Le lien entre le contrôle d'accès (AC) [8] et la blockchain repose sur la capacité de cette dernière à fournir une infrastructure décentralisée, sécurisée et transparente pour gérer les droits d'accès aux ressources. La blockchain permet d'enregistrer de manière immuable et auditée les politiques, les demandes et les permissions d'accès, ce qui renforce la confiance et la traçabilité dans les processus d'autorisation [31]. En utilisant les smart contracts, elle facilite l'automatisation, la vérification et l'application des politiques d'accès sans dépendre d'une autorité centrale, augmentant ainsi la sécurité, la scalabilité et la flexibilité du contrôle d'accès dans des environnements distribués comme le cloud ou l'Internet des Objets (IoT).

Avantage	**Impact**
Exécution autonome	Élimination des intermédiaires
Logique programmable	Politiques dynamiques
Auditabilité	Conformité réglementaire

Table 2.1 – Avantages des smart contracts pour le contrôle d'accès

2.6 Problématique

La diversité des services que proposent différents fournisseurs conduisent les organisations à adopter des architectures multi-cloud. Cette évolution pose de nouveaux défis

8. Access Control (AC) : Mécanisme de sécurité permettant de restreindre ou d'autoriser l'accès à des ressources en fonction de politiques prédéfinies.

en termes de contrôle d'accès, du fait de l'hétérogénéité des politiques de sécurité, du manque de standardisation entre les fournisseurs et des environnements distribués.

Les approches classiques, souvent centralisées, ne permettent pas de répondre aux enjeux contemporains de souplesse, de traçabilité et de transparence. Dans ce cadre, la blockchain apparaît comme une technologie potentiellement plus adaptée, grâce à ses garanties d'immuabilité et de décentralisation. Cependant, les solutions proposées aujourd'hui n'exploitent que partiellement ces propriétés, et la coordination sécurisée des décisions d'accès dans un système multi-cloud reste une question ouverte. Comment concevoir un mécanisme de contrôle d'accès interopérable et sécurisé dans un environnement multi-cloud, qui garantisse, en même temps, la transparence, l'auditabilité et la résilience du système ?

2.7 Modèles d'utilisation de la blockchain dans le Contrôle d'accès

2.7.1 Modèles Basés sur la Blockchain

2.7.1.1 Modèle BACC (Blockchain-Based Access Control for Cloud Data)

Proposé par N. Sohrabi et al[1], Ce modèle BACC (*Blockchain-Based Access Control for Cloud Data*) est un système de contrôle d'accès innovant qui utilise la blockchain pour sécuriser les clés de déchiffrement dans un environnement cloud. Voici comment cela fonctionne :

1. **Chiffrement des données** : Le propriétaire des données utilise une clé symétrique (comme AES) pour chiffrer les données, puis divise cette clé en plusieurs morceaux.
2. **Distribution des morceaux de clé** : Ces morceaux de clé sont ensuite répartis et stockés sur plusieurs nœuds maîtres de la blockchain, ce qui élimine le risque d'un point de défaillance unique.
3. **Stockage des données** : Les données chiffrées sont conservées dans le cloud, tandis que les morceaux de clé restent sur la blockchain.
4. **Accès demandé** : Quand un utilisateur souhaite accéder aux données, il doit récupérer au moins k morceaux de clé (parmi les n) auprès des nœuds de la blockchain.
5. **Reconstruction de la clé** : L'utilisateur utilise les morceaux récupérés pour reconstruire la clé de déchiffrement et ainsi déchiffrer les données pour y accéder.

2.7.1.2 Modèle BMAC (Blockchain-based Multi-Authority Access Control)

Proposé par Liu, J., et al [33] et [34], Le modèle **BMAC** (*Contrôle d'Accès Multi-Autorité basé sur la Blockchain*) est un système de contrôle d'accès innovant qui s'appuie sur la blockchain et plusieurs autorités pour gérer les attributs des utilisateurs et leurs droits d'accès. Voici comment cela fonctionne :

1. **Enregistrement des clés publiques** :
   — La *Certificate Authority (CA)* et les *Attribute Authorities (AA)* enregistrent leurs clés publiques sur la blockchain. Ces clés sont essentielles pour signer et vérifier les attributs des utilisateurs.

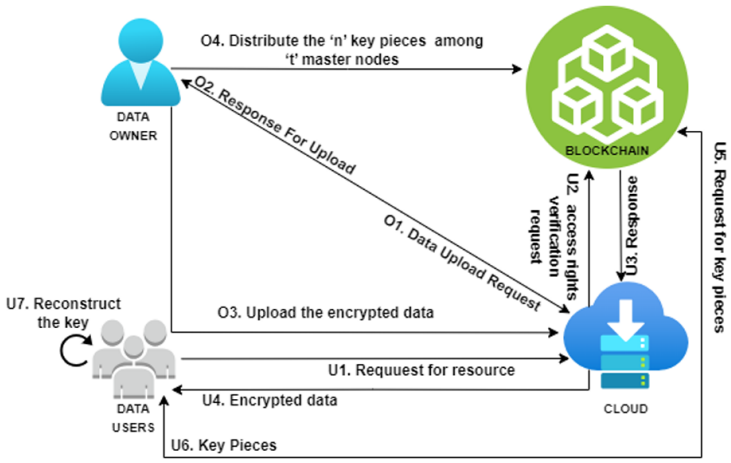

FIGURE 2.6 – Modèle BACC (Blockchain-Based Access Control for Cloud Data)[32]

2. **Chiffrement des données par le propriétaire des données (*Data Owner*)** :
   - Le propriétaire des données utilise un schéma de chiffrement basé sur les attributs (*CP-ABE*, Chiffrement basé sur les politiques d'attributs) pour sécuriser les données.
   - Il définit une politique d'accès basée sur les attributs, comme « Rôle = Manager » ou « Département = Finance », qui détermine quels utilisateurs peuvent déchiffrer les données.
3. **Demande d'accès par l'utilisateur (*Data User*)** :
   - Quand un utilisateur souhaite accéder aux données, il envoie une demande d'accès au *Cloud Service Provider (CSP)*.
   - Le CSP vérifie si l'utilisateur possède les attributs requis pour accéder aux données en consultant les *Attribute Authorities (AA)*.
4. **Génération des jetons d'attributs (*Attribute Tokens*)** :
   - Les *Attribute Authorities (AA)* créent des jetons d'attributs pour l'utilisateur en fonction de ses attributs, comme son rôle ou son département.
   - Ces jetons sont signés par les AA et envoyés à la blockchain pour validation.
5. **Validation des jetons d'attributs par la blockchain** :
   - La blockchain vérifie les jetons d'attributs signés par les AA et génère un *user_ID* pour l'utilisateur.
   - Si les jetons sont valides, la blockchain produit un jeton de déchiffrement et l'envoie à l'utilisateur.
6. **Déchiffrement des données** :
   - L'utilisateur utilise le jeton de déchiffrement pour accéder aux données les données stock ées dans le cloud.
   - Sil'utilisateur possède les attributs nécessaires (conformes à la politique d'accès définie par le propriétaire des données), il peut déchiffrer et accéder aux données.

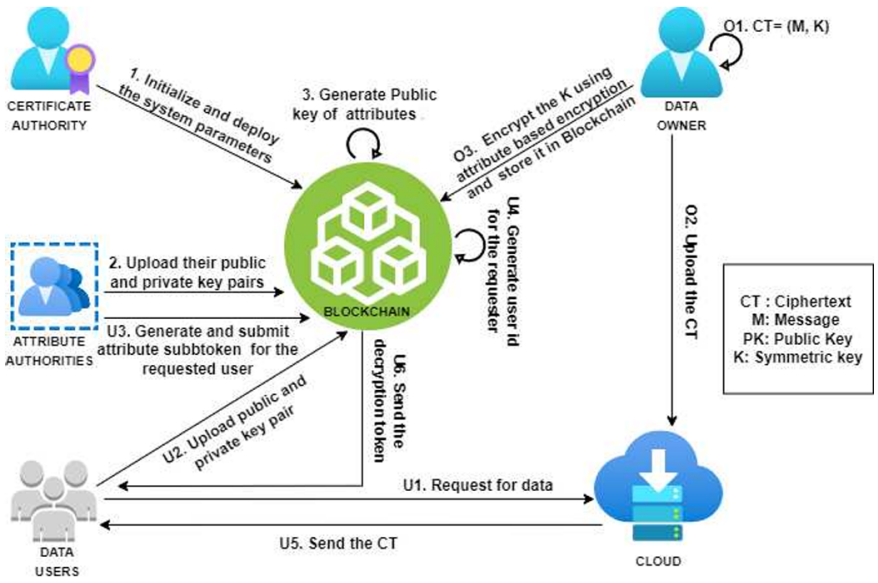

FIGURE 2.7 – Modèle BMAC (Blockchain-based Multi-Authority Access Control [32])

2.7.1.3 Modèle CBFF (Cloud-Blockchain Fusion Framework)

Le modèle CBFF, proposé dans l'article [35], est un cadre innovant qui fusionne le cloud et la blockchain pour garantir la responsabilité et la traçabilité des données dans un environnement multi-cloud.

1. **Téléchargement des données** :
   - Le propriétaire des données commence par diviser celles-ci en blocs, qu'il télécharge ensuite sur plusieurs fournisseurs de services cloud (*CSPs*).
   - Pour chaque bloc de données, le propriétaire crée deux types d'enregistrements :
     - *Short Record (SR)* : Contient des informations essentielles (ID du canal, hauteur du bloc, offset de la transaction). Ce SR est partagé avec tous les utilisateurs.
     - *Long Record (LR)* : Document détaillé incluant (type de données, nom du propriétaire, nom du cloud, chemin du fichier). Le LR est stocké dans la blockchain.
2. **Partage des données** :
   - Quand un utilisateur souhaite accéder aux données, il utilise le *Short Record (SR)* pour localiser le *Long Record (LR)* dans la blockchain.
   - Le LR contient toutes les informations nécessaires pour accéder aux données stockées sur les différents clouds.
3. **Traçabilité des opérations** :
   - Le modèle CBFF met en œuvre deux protocoles pour garantir la traçabilité :
     - *Operation Logging* : Chaque opération (téléchargement, partage, modification) est enregistrée dans la blockchain via un smart contract.
     - *Operation Tracing* : Les utilisateurs peuvent retracer l'historique via les enregistrements blockchain. Chaque entrée relie les opérations précédentes et suivantes.

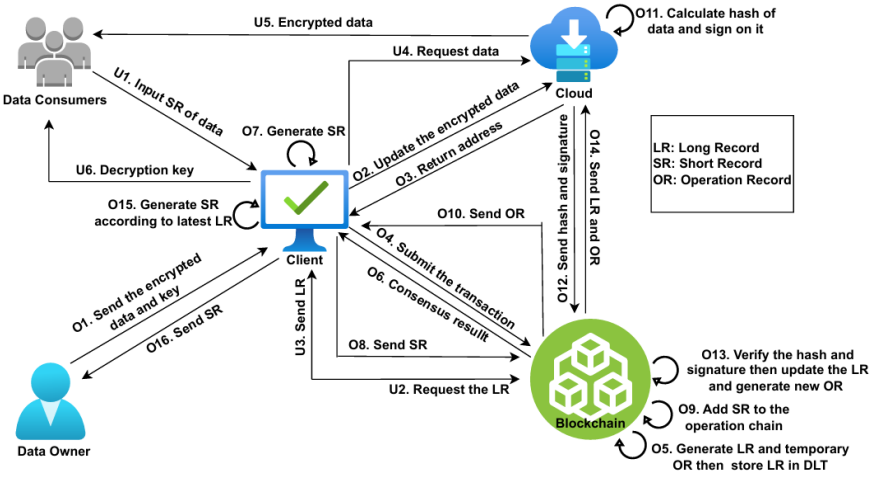

FIGURE 2.8 – Modèle CBFF (Cloud-Blockchain Fusion Framework) [32]

2.7.1.4 Modèle BC-ABAC – Contrôle d'accès par attributs avec responsabilité

Le modèle BC-ABAC (Blockchain-based Accountable Attribute-Based Access Control)[35], est un cadre innovant qui combine la technologie blockchain avec un contrôle d'accès basé sur les attributs, spécifiquement conçu pour les services cloud.

1. **Chiffrement des données par le propriétaire** :
   - Le propriétaire des données commence par chiffrer ses informations à l'aide d'un algorithme de chiffrement symétrique, comme AES.
   - Il crée également une matrice d'accès et des politiques d'utilisation qui précisent qui peut accéder aux données et dans quelles conditions.
2. **Vérification des attributs par les TAAs** :
   - Quand un utilisateur souhaite accéder aux données, il soumet ses attributs (comme son rôle ou son département) aux Autorités de Confiance des Attributs (TAAs) [9] pour validation.
   - Les TAAs s'assurent que les attributs de l'utilisateur correspondent aux politiques d'accès établies par le propriétaire.
3. **Création d'une session d'accès** :
   - Si les attributs sont jugés valides, le Fournisseur de Services Cloud (CSP) [10] met en place une session d'accès pour l'utilisateur.
   - Ensuite, le CSP publie les détails de cette session (comme l'URL des fichiers concernés, le hash des données, et l'adresse de l'utilisateur) sur la blockchain, garantissant ainsi transparence et responsabilité.
4. **Accès aux données** :
   - L'utilisateur peut alors accéder aux données chiffrées stockées dans le cloud.

9. Trusted Access Agents (TAAs) : Agents de confiance responsables de la gestion et de la sécurisation des accès dans des environnements distribués. Ils facilitent l'application des politiques d'accès en assurant l'intégrité et l'authenticité des demandes.
10. Cloud Service Provider (CSP) : Fournisseur de services cloud, entreprise qui met à disposition des ressources informatiques (stockage, calcul, réseau) via Internet.

- Il utilise la clé de déchiffrement donné pour déchiffrer et exploiter les données.

5. **Responsabilité et traçabilité** :
   - Toutes les sessions d'accès sont enregistrées sur la blockchain, permettant à tous les membres du réseau de suivre les activités des utilisateurs.
   - Cela assure que toute utilisation non autorisée ou malveillante des données peut être détectée et retracée.

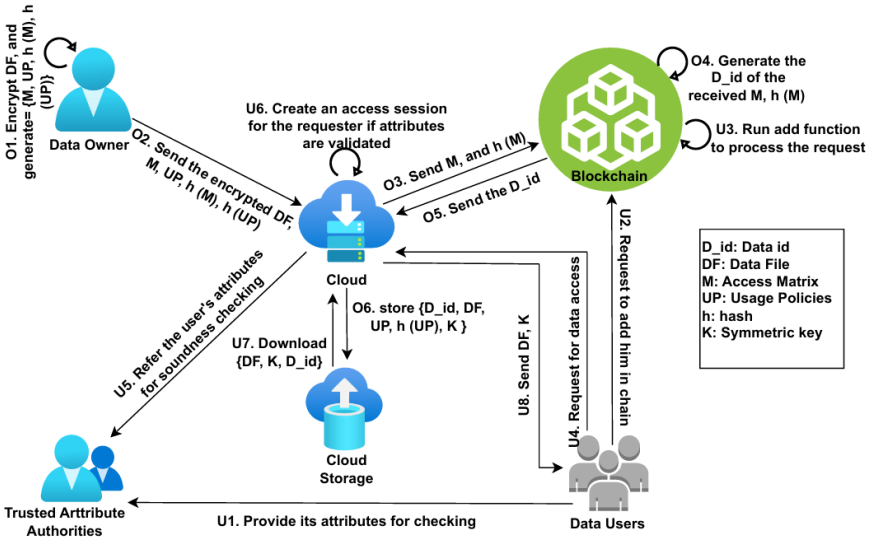

FIGURE 2.9 – Modèle BC-ABAC – Contrôle d'accès par attributs avec responsabilité [32]

2.7.2 Approches Institutionnelles

2.7.2.1 Monitoring décentralisé pour le contrôle d'accès fédéré

L'approche décrite dans « Decentralised Runtime Monitoring for Access Control Systems in Cloud Federations » [36] utilise la blockchain comme une infrastructure de surveillance pour les systèmes de contrôle d'accès dans des environnements cloud fédérés. Elle sert à enregistrer de manière immuable les journaux d'accès, les décisions de sécurité et les événements critiques. Les auteurs tirent parti des smart contracts pour automatiser la vérification des autorisations et détecter les violations d'accès en temps réel. La traçabilité est garantie par l'immutabilité de la chaîne, tandis que la confidentialité est préservée grâce au chiffrement des journaux. Cette architecture permet un monitoring distribué, sans point de défaillance central, offrant ainsi une auditabilité renforcée dans des environnements multi-organisations.

2.7.2.2 Blockchain comme infrastructure de contrôle d'accès

L'article intitulé « Blockchain for Access Control Systems » [25] présente la blockchain comme une infrastructure décentralisée qui peut vraiment renforcer le contrôle d'accès dans les systèmes informatiques d'aujourd'hui. Elle sert à enregistrer de manière immuable et vérifiable les politiques d'accès, les demandes d'autorisation et les journaux

d'activité, ce qui facilite la détection d'activités malveillantes. De plus, elle permet une gestion décentralisée des droits d'accès, sans avoir besoin d'une autorité centrale, et offre la possibilité d'exécuter des smart contracts pour adapter dynamiquement les permissions. Grâce à son mécanisme de consensus distribué, la blockchain assure une résilience solide et une scalabilité qui conviennent parfaitement aux systèmes cloud et IoT. En somme, elle joue un rôle essentiel dans l'automatisation et la transparence du contrôle d'accès dans les environnements modernes et distribués.

2.7.3 Comparaison des modèles de contrôle d'accès basés sur la blockchain

Afin de mieux situer notre approche par rapport aux travaux existants, nous avons établi une comparaison entre plusieurs modèles de contrôle d'accès reposant sur la blockchain. Cette comparaison s'appuie sur un ensemble de critères essentiels permettant d'évaluer la pertinence, l'efficacité et la robustesse de chaque solution.

Les critères retenus sont les suivants :

— **Type de contrôle d'accès** : désigne le modèle de gestion des droits d'accès (par exemple, basé sur les clés, sur les attributs ou sur des politiques hybrides).

— **Méthode de chiffrement** : fait référence aux techniques cryptographiques utilisées pour protéger les données ou les clés.

— **Stockage des données** : indique l'emplacement de stockage des données utilisateurs, qu'il soit dans un cloud unique ou réparti sur plusieurs fournisseurs.

— **Stockage blockchain** : désigne les types d'informations effectivement inscrites dans la blockchain (clés, jetons, logs, etc.).

— **Rôle de la blockchain** : met en lumière la fonction principale de la blockchain dans l'architecture (audit, sécurité, traçabilité, etc.).

— **Acteurs principaux** : énumère les entités impliquées dans le contrôle d'accès (utilisateurs, fournisseurs, autorités, etc.).

— **Traçabilité/Auditabilité** : évalue la capacité du système à garder une trace vérifiable des accès et des décisions prises.

— **Décentralisation** : mesure le niveau de distribution des responsabilités et l'absence de point de contrôle centralisé.

— **Smart contracts** : indique si des contrats intelligents sont utilisés et à quelles fins (vérification, audit, gestion des accès).

— **Objectif principal** : résume le but fondamental poursuivi par chaque modèle.

Ces critères permettent d'avoir une vue d'ensemble claire et objective des caractéristiques fonctionnelles et techniques des approches étudiées.

TABLE 2.2 – Comparaison des modèles de contrôle d'accès basés sur la blockchain

Critère / Modèle	**BACC**	**BMAC**	**CBFF**	**BC-ABAC**	**Monitoring décentralisé**	**Infra. Blockchain (CA)**
Type de contrôle d'accès	Basé sur clés (Seuil k parmi n)	Basé sur attributs multi-autorité (ABAC)	Fusion cloud + blockchain	ABAC avec responsabilité	Contrôle d'accès fédéré	Modèle générique d'enregistrement des droits
Méthode de chiffrement	Symétrique (AES), seuil de reconstruction	CP-ABE (Chiffrement basé sur la politique d'attributs)	Pas précisé, séparé de la logique de partage	Symétrique (AES) avec politique	N/A (surveillance et vérification)	N/A
Stockage des données	Cloud (chiffrées)	Cloud	Multi-cloud	Cloud	Cloud (multi-fédération)	N/A
Stockage blockchain	Fragments de clé	Jetons d'attributs et user_ID	Long Record (LR), traçabilité	Journaux de session et métadonnées	Journaux d'accès chiffrés, décisions, événements	Politiques, demandes, logs
Rôle de la blockchain	Sécurisation des clés	Vérification attributs, génération de jetons	Référencement, audit, transparence	Auditabilité et traçabilité	Surveillance, détection d'anomalies	Résilience, immutabilité, transparence
Acteurs principaux	Propriétaire des données, utilisateur, nœuds	Autorités d'attributs (AA), CA, CSP, utilisateur	Propriétaire, utilisateurs, smart contracts	Propriétaire, TAA, utilisateur, CSP	Agents de sondage, PDP, blockchain, smart contracts	Utilisateurs, fournisseurs, smart contracts
Traçabilité / Auditabilité	Faible (seulement sur les clés)	Moyenne (centrée sur attributs)	Forte (traçabilité d'opérations)	Forte (logs détaillés)	Très forte (monitoring distribué, événements temps réel)	Forte (audit et journalisation distribuée)
Décentralisation	Partielle (pour les clés)	Moyenne (multi-autorité)	Élevée (multi-cloud)	Moyenne à élevée	Très élevée (pas de point de défaillance)	Élevée (pas d'autorité centrale)

Critère / Modèle	BACC	BMAC	CBFF	BC-ABAC	DRAMS (Monitoring)	Infra. Blockchain (CA)
Smart contracts	Non	Oui (vérification jetons)	Oui (logging et tracing)	Oui (audit et session)	Oui (vérification automatique d'accès)	Oui (règles et permissions dynamiques)
Objectif principal	Sécuriser la clé de déchiffrement	Gérer des accès multi-attributs via blockchain	Fusion cloud-blockchain pour auditabilité	Garantir un accès ABAC traçable	Superviser l'accès en temps réel	Journalisation et contrôle d'accès distribué

2.8 Synthèse critique

L'analyse transversale des modèles de contrôle d'accès basés sur la blockchain met en évidence une diversité d'approches selon les priorités de sécurité, de traçabilité, de décentralisation ou d'efficacité. Chaque modèle se distingue par des choix techniques particuliers, influençant directement ses performances selon les critères étudiés.

Sur le plan du type de contrôle d'accès, le modèle BMAC s'impose comme le plus performant grâce à son architecture ABAC distribuée. En reposant sur plusieurs autorités d'attributs (AA) indépendantes, il permet une gestion fine, souple et évolutive des droits d'accès, tout en assurant une séparation des responsabilités et une meilleure tolérance aux fautes.

Concernant la méthode de chiffrement, BMAC se démarque également par son recours au chiffrement basé sur les attributs (CP-ABE). Cette approche permet de lier directement l'accès aux données aux attributs certifiés d'un utilisateur, sans besoin de clés symétriques partagées, comme c'est le cas dans le modèle BACC. Cela assure une meilleure évolutivité dans un environnement distribué.

En matière de stockage des données, le modèle CBFF (Cloud Blockchain Fusion Framework) montre une nette supériorité. Il propose une architecture multi-cloud où les données sont réparties et indexées via des structures comme les short records (SR) et long records (LR), assurant ainsi une redondance, une résilience et une efficacité d'accès accrues.

Le stockage dans la blockchain varie fortement d'un modèle à l'autre. Le CBFF se distingue encore ici par sa capacité à journaliser de manière détaillée les opérations via des smart contracts dédiés, offrant ainsi un mécanisme robuste de suivi et de preuve. Ce choix fait de lui le modèle le plus performant en matière de traçabilité et d'auditabilité.

En ce qui concerne le rôle de la blockchain, le modèle BC-ABAC se démarque par son utilisation complète des propriétés immuables de la blockchain. Il permet non seulement la vérification des politiques d'accès mais aussi la journalisation des sessions, assurant une transparence maximale et une responsabilisation des acteurs.

Le critère de décentralisation est fondamental dans le contexte des systèmes distribués. Ici encore, BMAC excelle grâce à l'absence de point central de contrôle : les décisions d'accès sont prises de manière autonome par les autorités d'attributs, ce qui renforce la robustesse et la confiance dans le système.

L'usage des smart contracts est un autre facteur différenciateur. Le modèle CBFF en tire pleinement parti en les utilisant pour l'automatisation des journaux d'accès, la gestion des preuves, et l'interaction avec plusieurs fournisseurs cloud, assurant ainsi une cohérence globale dans un environnement hétérogène.

Enfin, les objectifs principaux de chaque modèle influencent directement leurs choix d'architecture. BMAC est centré sur la délégation et la sécurité des décisions d'accès, CBFF sur la traçabilité et la compatibilité inter-cloud, tandis que BC-ABAC met l'accent sur la conformité et la transparence.

En somme, si l'on considère une évaluation globale sur la base des critères définis, le modèle BMAC ressort comme le plus complet pour un système de contrôle d'accès décentralisé, sécurisé et dynamique. Il est particulièrement adapté aux environnements cloud distribués et collaboratifs. Toutefois, pour des cas d'usage où la traçabilité, la preuve d'audit et la résilience inter-cloud sont prioritaires, le modèle CBFF constitue un choix supérieur, notamment grâce à sa gestion fine des enregistrements d'accès et à son intégration transparente entre la couche blockchain et les fournisseurs cloud.

2.9 Conclusion

La recherche sur les méthodes actuelles de contrôle d'accès utilisant la blockchain montre des avancées notables en matière de traçabilité, de sécurité et de gestion fine des autorisations, grâce à l'exploitation des attributs et des identifiants décentralisés. Cependant, ces cadres restent encore limités dans des environnements multi-clouds, en raison de la diversité des politiques d'accès, du manque de collaboration entre les fournisseurs, et d'une automatisation insuffisante dans la prise de décision.

Face à ces contraintes, notre contribution propose une nouvelle architecture combinant une passerelle d'interface distribuée (DAPI), un système d'organisation standardisé par fournisseur, et des contrats intelligents participant activement à l'évaluation des règles d'accès. Chaque fournisseur cloud est ainsi représenté de manière adaptée au sein du système, tandis que la blockchain est exploitée non seulement pour la journalisation, mais aussi comme socle de gouvernance partagée.

Cette approche favorise des décisions transparentes, autonomes et interopérables, ouvrant la voie à un modèle de contrôle d'accès véritablement décentralisé, flexible et adapté aux environnements cloud modernes.

Chapitre 3

DAccess-ChainMC (Decentralized Access Chain for Multi-Cloud)

3.1 Introduction

Dans un contexte où les environnements informatiques adoptent de plus en plus des stratégies multi-cloud, la gestion des accès aux ressources réparties entre plusieurs fournisseurs pose des défis majeurs en matière de sécurité, d'interopérabilité et de gouvernance. Les modèles classiques, centralisés ou spécifiques à chaque fournisseur, ne permettent plus de répondre efficacement à la complexité croissante de ces environnements distribués.

Ce chapitre présente l'architecture proposée pour un système de contrôle d'accès décentralisé, conçu pour opérer de manière sécurisée, transparente et automatisée dans un écosystème multi-cloud. L'objectif est de permettre à un utilisateur d'accéder à des services cloud diversifiés via une orchestration neutre, tout en garantissant la traçabilité des décisions et la résilience du processus. À travers une analyse détaillée des composants, du fonctionnement du système et des mécanismes d'intégration blockchain, ce chapitre pose les bases techniques et conceptuelles de la solution envisagée.

3.2 Objectifs et contraintes

La solution proposée vise à offrir un mécanisme de contrôle d'accès décentralisé, sécurisé et interopérable entre plusieurs fournisseurs de services cloud, permettant à un utilisateur de formuler des requêtes d'accès à des ressources réparties sur différents clouds tout en assurant une prise de décision cohérente et automatisée. Elle repose sur des contrats intelligents pour une gouvernance décentralisée des politiques d'accès et utilise une passerelle décentralisée (DAPI) pour une orchestration neutre des demandes, tout en gérant les scénarios complexes multi-cloud. Les défis incluent l'hétérogénéité des politiques d'accès, la diversité des formats de données et la variabilité des délais de réponse entre fournisseurs. La solution doit également garantir la confidentialité et l'intégrité des données, éviter les points de défaillance uniques, enregistrer de manière immuable les décisions sur la blockchain et assurer une gestion transparente et traçable des autorisations.

3.3 Vue d'ensemble de l'architecture

3.3.1 Composants principaux

L'architecture proposée repose sur plusieurs composants interconnectés, conçus pour opérer de manière coordonnée dans un environnement multi-cloud. Chacun joue un rôle spécifique dans le traitement d'une requête d'accès :

— **DAPI (Decentralized API Gateway)** : Il s'agit de la passerelle décentralisée qui reçoit les requêtes d'accès de l'utilisateur. Elle agit comme un orchestrateur neutre, découpe les requêtes en sous-requêtes spécifiques à chaque fournisseur cloud, et les envois ,(Exemple : API3 et Airnode).

— **Fournisseurs Cloud** : Ce sont les prestataires de services (par exemple AWS, Azure, GCP), chacun avec ses propres politiques d'accès et formats de données. Ils reçoivent les sous-requêtes de la DAPI et y répondent selon leurs règles locales.

— **Smart Contract** : Il constitue le cœur décisionnel. Ce contrat intelligent reçoit les réponses des fournisseurs, les traite (normalisation, agrégation, évaluation de politiques) et produit une décision d'accès globale (autorisée, refusée ou partielle).

— **Utilisateur Final** : L'acteur qui initie une demande d'accès à une ou plusieurs ressources. En cas de besoin (par exemple si certains attributs manquent), l'utilisateur peut être sollicité pour compléter sa requête.

— **Registre Blockchain** : Tous les événements critiques du processus (demandes, décisions, négociations, conflits résolus) sont enregistrés de manière immuable dans la blockchain sous forme de transactions pour garantir la traçabilité et l'auditabilité.(Exemple :Ethereum et Bitcoin)

3.3.2 Structure de la requête utilisateur

La requête utilisateur représente la demande initiale envoyée par l'utilisateur à la passerelle décentralisée (DAPI). Elle contient les informations nécessaires à l'analyse, au découpage, et à l'acheminement vers les fournisseurs cloud appropriés. Sa structure est généralement standardisée sous un format JSON ou équivalent.

```
{
  "user_id": "",
  "auth_token": "",
  "timestamp": "",
  "requested_resources": [
    { "provider_id": "",// le client va choisir sur son interface un
    fourniseur a son choix
      "service_type": "",
      "service_parameters": {
        "parameter_1": "",
        "parameter_2": ""
        // ...
      },

    }
    // D'autres services peuvent suivre
  ],
  "constraints": {
    "cost_limit": "",
```

```
    "region": "",
    "compliance": ""
    // ...
  },
  "preferred_providers": [
    ""
    // Liste optionnelle de fournisseurs souhaiter si le founisseur
    demander n'est  pas disponible
  ]
}
```

Listing 3.1 – Structure de requête DAPI

Champ	Description
`user_id`	Identifiant unique de l'utilisateur final .
`auth_token`	Jeton d'authentification signé (JWT, OAuth, etc.).
`timestamp`	Horodatage de la requête.
`requested_resources`	Liste des services demandés avec précision des fournisseurs souhaité , avec leurs paramètres techniques, niveaux de priorité.
`constraints`	Contraintes fonctionnelles et réglementaires (coût, localisation, conformité).
`preferred_providers`	Fournisseurs cloud explicitement préférés par l'utilisateur (optionnel).

Table 3.1 – Champs composant la structure d'une requête utilisateur envoyée à la DAPI

3.3.3 Diagramme général de l'architecture

Le diagramme d'architecture, présenté dans la Figure 1.1, synthétise les relations fonctionnelles entre les composants. Il met en évidence :

— Le rôle central de la dAPI comme point de coordination décentralisé ;

— Les interactions entre la dAPI et les fournisseurs cloud dans la gestion des sous-requêtes ;

— Le traitement global des réponses par les smart contracts ;

— L'enregistrement final des décisions sur la blockchain.

Ce diagramme illustre l'approche distribuée et la séparation claire des responsabilités dans le traitement des requêtes d'accès.

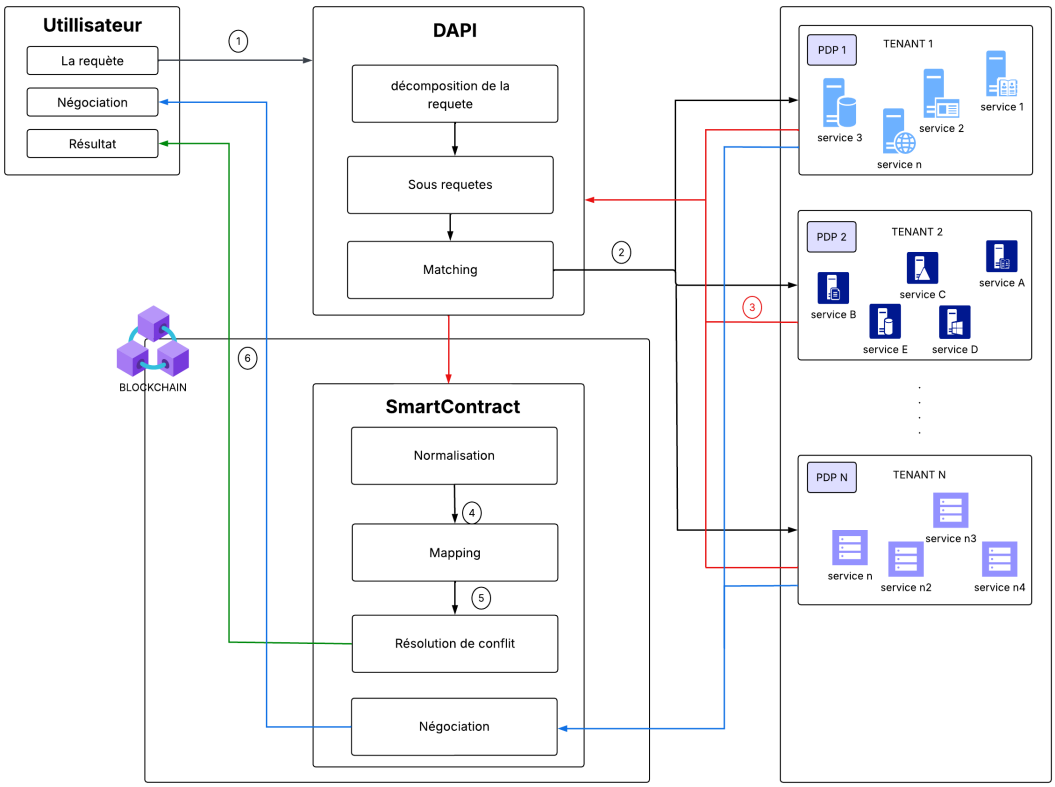

FIGURE 3.1 – Diagramme général de l'architecture

3.4 Intégration des fournisseurs cloud au réseau blockchain

Intégration des fournisseurs cloud via la blockchain Pour permettre aux fournisseurs cloud (AWS, Azure, Google Cloud, etc.) d'interagir de manière autonome et sécurisée avec la blockchain, deux approches sont envisageables : l'utilisation d'une API JSON-RPC ou l'adoption d'Airnode, un nœud oracle first-party. Notre choix s'est porté sur Airnode pour son interopérabilité native avec les smart contracts, sa simplicité d'intégration, sa sécurité cryptographique et son alignement avec les principes de décentralisation.

Airnode = passerelle entre une API Web classique et un smart contract sur la blockchain

Airnode agit comme une passerelle automatique entre les API REST des fournisseurs et la blockchain, convertissant les requêtes on-chain en appels HTTP et publiant les réponses sous forme de transactions signées. Son déploiement est simple : une fois configuré, il écoute les appels des smart contracts, interroge l'API du fournisseur et renvoie les données on-chain sans intermédiaire. Cette approche garantit une intégration fluide, sécurisée et décentralisée des fournisseurs cloud dans notre écosystème blockchain.

3.5 Soumission de la requête par l'utilisateur

Le client initie le processus en soumettant une requête structurée via une interface connectée à un dAPI, exposé à travers un Airnode. Cette RequêteClient contient plusieurs champs essentiels permettant de définir précisément le besoin. Voici la requête du client vers le dAPI :

```
RequêteClient = { identifiant_utilisateur, jeton_authentification,
                          horodatage,
services_demandés = [ { fournisseur, type_service, paramètres = {
            paramètre_1, paramètre_2, ... } }, ... ],
      contraintes = { coût_max, région, conformité, ... },
 fournisseurs_préférés = [ fournisseur_1, fournisseur_2, ... ] }
```

Champ	Description
`identifiant_utilisateur`	Identifiant unique du client soumettant la requête, utilisé pour le suivi et la traçabilité.
`jeton_authentification`	Jeton d'accès ou clé API permettant de vérifier l'identité du client.
`horodatage`	Date et heure de la soumission de la requête, utile pour le suivi temporel.
`services_demandés[]`	Liste des services que le client souhaite consommer, chacun comprenant :
`fournisseur`	Le nom ou l'identifiant du fournisseur cloud ciblé (ex : AWS, Azure).
`type_service`	Type de service demandé (ex : stockage, calcul, machine learning).
`paramètres`	Ensemble de paramètres spécifiques au service (ex : taille, durée, capacité).
`contraintes`	Contraintes techniques ou commerciales imposées par le client :
`coût_max`	Budget maximum autorisé.
`région`	Région géographique souhaitée pour le traitement ou le stockage.
`conformité`	Exigences réglementaires ou normatives (ex : RGPD, HIPAA).
`fournisseurs_préférés[]`	Liste optionnelle de fournisseurs favoris à privilégier en cas de redondance ou d'indisponibilité du fournisseur principal.

Table 3.2 – Description des champs de la requête client transmise au dAPI

Cette requête, une fois transmise au dAPI, servira de base à la génération de sous-requêtes destinées à chaque fournisseur concerné.

3.6 Passerelle décentralisée (dAPI)

La passerelle décentralisée (dAPI) [1] est un réseau de flux de données agrégées sur la blockchain, fonctionnant sans intermédiaire centralisé. Contrairement aux API traditionnelles, elle permet aux dApps d'accéder directement à des sources de données via un agrégateur qui interroge, harmonise et transmet les réponses de manière unifiée, sous supervision décentralisée. La DAPI joue un rôle d'orchestrateur neutre : elle reçoit les requêtes utilisateur, les transforme en sous-requêtes adaptées aux fournisseurs cloud, sans prendre de décision. Ce choix s'impose face aux limites des API classiques (REST/SOA), inadaptées à la blockchain (absence de signature des données, de transactions on-chain ou d'appels directs aux smart contracts). Le dAPI garantit ainsi une interaction sécurisée, automatisée et vérifiable, avec traçabilité et immutabilité, grâce à son intégration native avec les smart contracts et l'utilisation d'un Airnode pour la transmission des données. Le schéma détaillé de l'architecture d'un dAPI est présenté en **Annexe .1** (Figure 3).

3.6.1 Découpage intelligent des requêtes par la dAPI

Une des fonctions clés de la passerelle décentralisée dAPI

consiste à effectuer un découpage intelligent de la requête d'origine formulée par l'utilisateur. Ce mécanisme permet d'adapter dynamiquement la requête globale en plusieurs sous-requêtes spécifiques, chacune étant destinée à un fournisseur cloud particulier.

Ce découpage repose sur une série d'étapes automatisées et coordonnées :

3.6.1.1 Analyse sémantique de la requête

Dès réception de la requête, la dAPI procède à une analyse de son contenu afin d'identifier les services demandés, les paramètres associés (configuration technique, région, ressources), ainsi que les niveaux d'accès souhaités (lecture, écriture, exécution, etc.). Cette analyse repose sur un parser sémantique qui lit les éléments de la requête à l'aide de modèles prédéfinis. Un schéma standard est requis pour faciliter l'analyse automatique.

3.6.1.2 Association des services aux fournisseurs

Une fois les services identifiés, la dAPI se réfère à une table de correspondance dynamique ou à un moteur de règles pour déterminer le fournisseur compétent pour chaque type de service.

Table de correspondance dynamique

La table de correspondance dynamique est une base de données évolutive qui associe, en temps réel, chaque service aux fournisseurs cloud capables de le fournir. Elle prend en compte les mises à jour fréquentes comme l'ajout de nouveaux fournisseurs, la modification des capacités techniques, ou l'évolution des politiques de service.

Moteur de règles

Le moteur de règles, quant à lui, applique un ensemble de règles logiques permettant d'automatiser le processus de sélection. Ces règles s'appuient sur plusieurs critères, notamment :

1. Decentralized API : Interface de Programmation Applicative Décentralisée

— les préférences explicites de l'utilisateur (par exemple un fournisseur préféré),

— la compatibilité fonctionnelle entre les services demandés et les offres disponibles chez chaque fournisseur,

— les contraintes contractuelles ou géographiques, telles que les exigences réglementaires (ex. : respect du RGPD), la localisation des données, ou les considérations économiques (comme les coûts d'usage).

Résultat de la sélection

Grâce à cette combinaison entre correspondance dynamique et raisonnement basé sur des règles, la DAPI oriente chaque service vers le fournisseur cloud le plus approprié, en assurant une répartition optimale et conforme aux attentes de l'utilisateur.

3.6.1.3 Déduction automatique du niveau de dépendance

La Figure 1.2 illustre l'architecture fonctionnelle du système de classification du niveau de dépendance dans la DAPI. Le processus repose sur un moteur de règles centralisé , qui orchestre l'analyse de plusieurs sources d'information pour produire une sortie annotée (`dependency_level`).

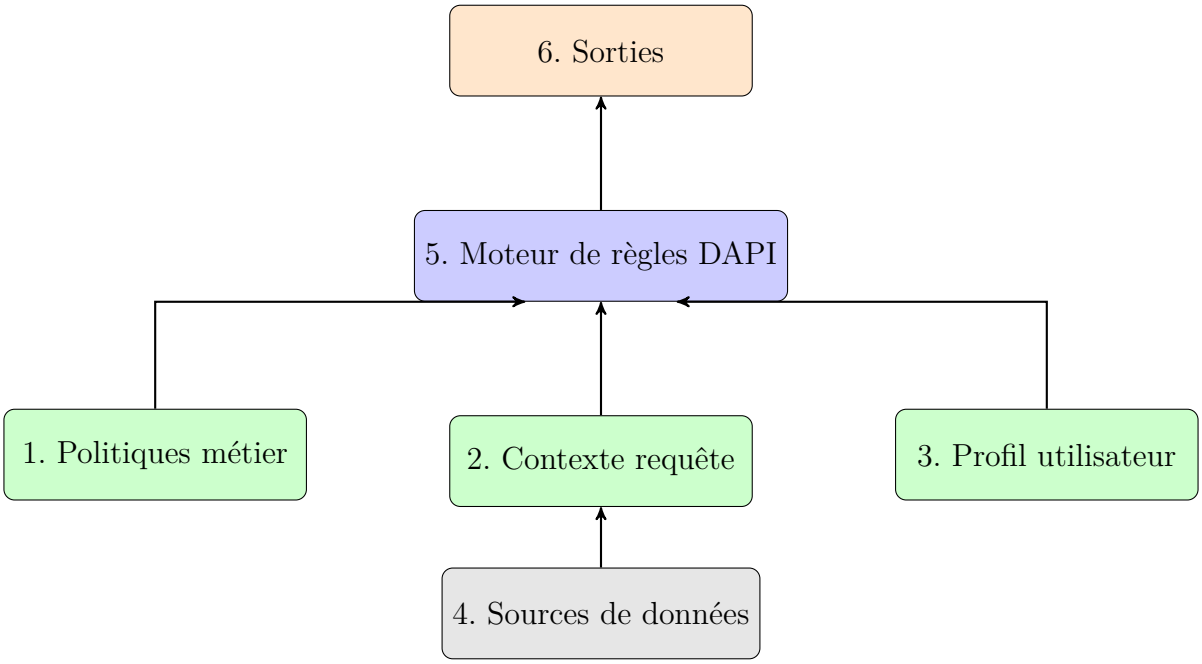

FIGURE 3.2 – Architecture fonctionnelle du système de classification

Politiques métier : Ce bloc contient les règles internes définies par l'organisation. Elles spécifient, par exemple, quels services doivent être considérés comme critiques selon le métier, le secteur ou le niveau de sécurité attendu. Ces règles influencent fortement la décision finale du moteur.

Contexte requête : Ce module regroupe les informations relatives à la requête utilisateur actuelle : le type de service demandé (stockage, base de données, VM. . .), la criticité déclarée, ou les attentes de performance (SLA). Ces données sont nécessaires pour estimer si le service est fondamental dans le contexte d'usage.

Profil utilisateur : Cette composante décrit le demandeur du service. Elle comprend son rôle (administrateur, développeur, analyste. . .), son niveau de privilège, et parfois son

unité organisationnelle. Cela permet de pondérer la priorité du service en fonction de son importance pour l'utilisateur.

Sources de données : Ce bloc représente les bases d'information utilisées pour alimenter les modules précédents. Il peut s'agir d'un cache local, de fichiers JSON/YAML, ou d'une base de données interne synchronisée avec d'autres systèmes (IAM, monitoring, etc.).

Moteur de règles DAPI : C'est le cœur du système. Il combine les informations issues des blocs précédents à l'aide de règles logiques formelles. Son objectif est de produire une évaluation automatisée du niveau de dépendance de chaque service demandé : `strong` (essentiel) ou `weak` (secondaire).

Sorties : La décision du moteur est transmise sous forme d'annotation au reste du système. Elle est injectée dans les sous-requêtes générées par la DAPI, permettant ainsi une gestion fine et contextuelle de leur exécution, leur priorité ou leur traitement dans la suite du pipeline (négociation, autorisation, etc.).

3.6.2 Génération des sous-requêtes

À partir des services associés aux fournisseurs, la DAPI génère des sous-requêtes indépendantes et personnalisées pour chaque API cible. Cette génération comprend :

— L'extraction des paramètres pertinents

— La traduction du format vers celui attendu par chaque fournisseur (AWS, Azure, GCP ayant chacun leurs spécificités)

— L'ajout des informations de sécurité nécessaires, telles que le jeton d'accès, la signature cryptographique, le timestamp, ou encore la clé d'API temporaire

Chaque sous-requête est ainsi autonome, complète, et conforme au format technique du fournisseur cible.

3.6.2.1 Structure des sous-requêtes

Une fois le découpage effectué et les fournisseurs identifiés, la DAPI génère des sous-requêtes indépendantes, adaptées à chaque API de fournisseur cloud. Chaque sous-requête est construite selon une structure standardisée, enrichie d'éléments contextuels et de métadonnées de contrôle. Cette standardisation facilite l'interopérabilité et garantit la traçabilité et la sécurité des interactions. Le listing 1.2 représente la structure des sous-requêtes.

```
{ "id_Srequete
  "id_requette": "",
  "timestamp": "",


      "service_spec": {
        "service_type": "",
        "service_parameters": {
          "parameter_1": "",
          "parameter_2": ""
        },

        "dependency_level": ""
```

```
      },

      "security": {
        "signature": ""
      }
    }
  ]
}
```

Listing 3.2 – Structure des sous-requêtes DAPI

Voici Table 1.3 qui explique chaque champ des sous-requêtes

Champ	**Description**	**Utilité / Rôle**
`id_Srequete`	Identifiant unique de la sous requête	Permet d'identifier la sous requete.
`id_requete`	Identifiant unique de la requête principale (parent).	Permet d'identifier et de tracer la requête globale.
`timestamp`	Date et heure d'émission de la requête principale.	Sert à la gestion temporelle et au suivi des requêtes.
`service_spec`	Spécifications du service demandé.	Contient la nature et les paramètres du service.
`service_type`	Type de service demandé (ex : VM, base de données).	Définit la catégorie de service souhaitée.
`service_parameters`	Paramètres spécifiques du service (ex : taille VM, version DB).	Permet une configuration fine du service.
`dependency_level`	Niveau de dépendance vis-à-vis d'autres services (ex : 0 = indépendant).	Aide à gérer les priorités et dépendances entre services.
`security`	Données liées à la sécurité et authentification.	Contient les informations nécessaires à la validation de la requête.
`auth_token`	Jeton d'authentification pour le fournisseur/service.	Permet l'authentification indépendante auprès du fournisseur.
`signature`	Signature numérique ou HMAC de la requête.	Garantit l'intégrité et l'authenticité des données envoyées.

Table 3.3 – Champs composant la structure des sous-requête multi-cloud

3.6.2.2 Envoi parallèle des sous-requêtes

Une fois que la requête utilisateur est décomposée en plusieurs sous-requêtes correspondant aux différents services cloud (par exemple : une base de données sur AWS, un service d'authentification sur GCP, un microservice web sur Azure, etc.), la DAPI entre en action :

- Génération d'un identifiant unique pour chaque sous-requête (exemple : UUID-REQ-1234-WEB, UUID-REQ-1234-DB), ce qui facilitera le suivi ultérieur.
- Envoi en parallèle :
  - La DAPI n'attend pas qu'un fournisseur ait fini avant d'envoyer à l'autre.
  - Grâce à la programmation asynchrone (multi-threading, appels non bloquants, file d'événements), la DAPI envoie simultanément les sous-requêtes aux différents points d'entrée API de chaque fournisseur.
  - Cela réduit le temps global d'exécution car les opérations se déroulent en parallèle, ce qui est crucial dans un contexte multi-cloud.
- Aucune logique d'agrégation dans la DAPI :
  - La DAPI joue le rôle d'un simple routeur parallèle.
  - Elle transfère les sous-requêtes dans les meilleurs délais, sans attendre leur réponse.
- Smart contract en réception :
  - Chaque fournisseur répond directement au smart contract (ou la DAPI transfère simplement la réponse au smart contract), ce dernier se charge ensuite du suivi, de la traçabilité, de la consolidation et de la prise de décision.
  - Ainsi, le smart contract garde une vision complète de l'état des sous-requêtes sans dépendre d'un orchestrateur centralisé.

  la figure 1.4 représente le flux complet de la DAPI

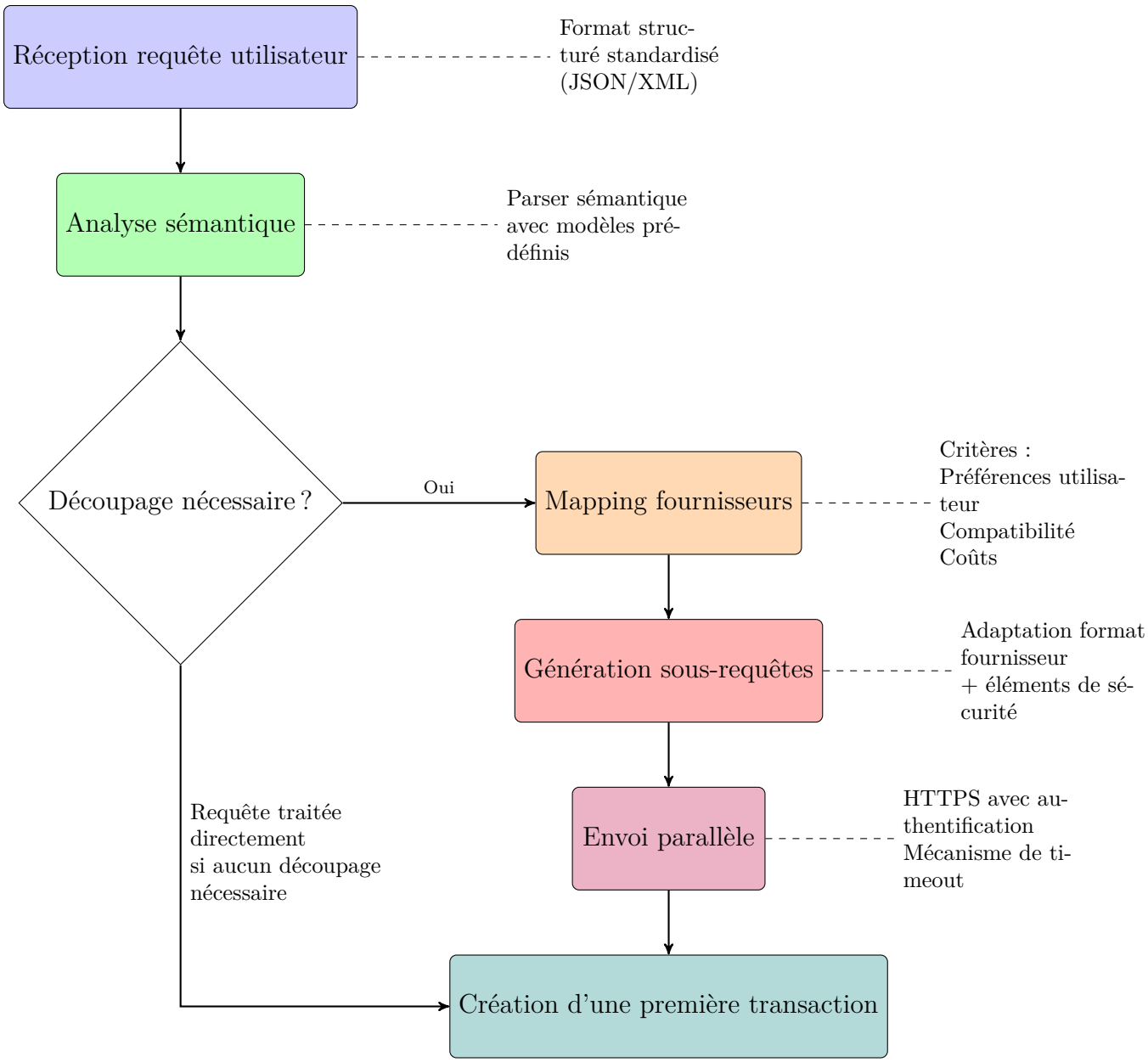

FIGURE 3.3 – Workflow complet de la DAPI

3.7 Traitements locales des sous-requetes par les fournisseur cloud

Lorsqu'une sous-requête est émise par la DAPI (Decentralized API Gateway) et transmise à un fournisseur cloud via une API sécurisée, le fournisseur procède à plusieurs étapes essentielles pour assurer le traitement correct et sécurisé de la demande. Ces étapes permettent de garantir l'intégrité, la conformité et la bonne exécution des services demandés dans un environnement multi-cloud distribué.

3.7.1 Réception et sécurisation de la sous-requête

La première étape consiste en la réception de la sous-requête via l'API sécurisée mise en place par le fournisseur. Cette communication repose sur des protocoles de sécurité

tels que TLS/SSL, assurant la confidentialité et l'intégrité des données échangées. La sous-requête contient notamment :

Un jeton d'authentification

Une signature numérique permettant de vérifier son authenticité

Des mécanismes de protection contre les altérations malveillantes en transit

3.7.2 Authentification et validation

Vérification du jeton d'authentification pour confirmer l'origine autorisée

Analyse de la signature numérique pour garantir l'intégrité du contenu

Application de mécanismes de contrôle d'accès :

—

Basés sur les rôles (RBAC)

Basés sur les attributs (ABAC)

Validation de la conformité aux politiques internes du fournisseur

3.7.3 Analyse et préparation du service

Extraction du type de service demandé et des paramètres associés

Vérification de la validité syntaxique et logique des paramètres

Aucune analyse des dépendances inter-services n'est effectuée à ce stade

3.7.4 Allocation des ressources

Réservation des ressources nécessaires

Gestion indépendante par chaque fournisseur

Non-prise en compte des interactions avec d'autres clouds

3.7.5 Exécution et configuration du service

Le service est alors instancié et configuré selon les paramètres fournis. Cela inclut l'exécution de scripts d'initialisation, la mise en place d'environnements virtuels, et la configuration réseau ou applicative nécessaire au bon fonctionnement du service demandé. Les fournisseurs aprés traitement des sous-requetes **initie une nouvelle transaction** a la blockchain contenant leurs réponses par rapport aux ressources solliciter dans la requête.

3.8 Transactions

3.8.1 Structure des transactions sur la blockchain

Une transaction valide sur la blockchain doit contenir les champs suivants :

- **Identifiant unique (transaction ID)** : permet d'identifier de manière immuable chaque transaction dans le registre.
- **Adresse de l'expéditeur (`from`)** : l'adresse blockchain du compte qui envoie la transaction.
- **Adresse du destinataire (`to`)** : l'adresse blockchain du compte qui reçoit la transaction ou le smart contract cible.
- **Données associées (`data` ou `payload`)** : contient les informations utiles comme les paramètres d'un appel au smart contract (ex. : identifiant de requête, réponse fournisseur,id block, etc.).
- **Montant (`value`)** : la quantité de cryptoactifs transférés dans la transaction (souvent 0 dans les appels aux contrats intelligents).
- **Nonce** : un nombre séquentiel propre à chaque compte, utilisé pour éviter les doublons et garantir l'ordre des transactions.
- **Signature cryptographique** : prouve que la transaction a été signée par le détenteur de la clé privée de l'expéditeur.
- **Frais de transaction (`gas` et `gasPrice`)** : même dans un réseau PoA, ces champs permettent de mesurer la complexité d'exécution et éviter les abus.

Champ	Valeur Exemple
`Transaction ID`	0x9a45d3abef
`From (expéditeur)`	0xF1e2B8c9F765a
`To (destinataire)`	0xDeAdBeEf1
`Value`	0 ETH (appel de smart contract, pas de transfert d'argent)
`Data (payload)`	{requestID : "REQ-456", response : "OK", providerID : "CLOUD-A"}
`Nonce`	12
`Signature`	0xabc123... (signature elliptique ECDSA générée avec la clé privée)
`Gas`	21000
`Gas Price`	1 Gwei

Dans ce qui suit, on va s'intéresser au **champ `data`** de la transaction. Ce champ constitue la partie utile de la transaction, contenant les informations spécifiques à l'appel d'une fonction dans un smart contract ou à l'encapsulation de données pertinentes. Dans le contexte de notre architecture, le `data` embarque l'ensemble des paramètres métiers nécessaires au traitement de la requête.

3.8.2 Création d'une première transaction par dAPI

Après la génération des sous-requêtes par le dAPI, ce dernier initie une transaction sur la blockchain via un Airnode, servant d'intermédiaire sécurisé entre le dAPI et le smart contract. Le champ data pour cette transaction est structurée comme suit :

`DATA_DapiTransaction = { requestIdParent, nbSubrequests, subRequestIds, providerIds, timestamp, client, metadata, functionSelector, contractAddress, signature }`

Champ	Description
`request_id_parent`	Identifiant unique de la requête principale
`nb_subrequests_expected`	Nombre total de sous-requêtes générées (et donc de réponses attendues)
`sub_request_ids[]`	Tableau contenant les identifiants des sous-requêtes associées
`provider_ids[]`	(Optionnel) Liste des fournisseurs ciblés (ex : AWS, Azure, Google)
`timestamp`	Horodatage de la requête, pour la traçabilité
`client_address`	Adresse blockchain du client à l'origine de la requête
`metadata`	(Optionnel) Informations supplémentaires comme les paramètres de traitement
`function_selector`	Nom ou signature de la fonction du smart contract à appeler (ex : `registerRequest(...)`)
`contract_address`	Adresse du smart contract cible à appeler sur la blockchain
`signature`	Signature cryptographique du dAPI (ou du client) pour garantir l'intégrité des données

Table 3.5 – Champs contenus dans la DATA d'une transaction initiale envoyée par le dAPI au smart contract

— `request_id_parent` : permet d'associer toutes les futures transactions (réponses des fournisseurs) à une même requête logique.

— `nb_subrequests_expected` : permet au smart contract de savoir quand déclencher le traitement (quand toutes les réponses sont là).

— `sub_request_ids[]` : optionnel mais utile si le smart contract veut suivre précisément quelle sous-réponse est reçue ou manquante.

Cette transaction déclenche un appel vers le smart contract. Toutefois, le contrat intelligent ne s'exécute pas immédiatement : il entre dans un état d'attente, se mettant à l'écoute des réponses des différents fournisseurs. Il surveille la blockchain jusqu'à la réception de la dernière réponse attendue, en se basant sur la valeur `nbSubrequests` indiquée dans la transaction initiale. Ce n'est qu'à la réception de l'ensemble des réponses associées aux sous-requêtes que le smart contract s'active, agrège les données reçues, puis exécute sa logique métier.

3.8.3 Création des transactions par les fournisseurs cloud

Lorsqu'un fournisseur cloud répond à une sous-requête, il soumet une transaction sur la blockchain via un Airnode. Cette transaction contient, dans son champ `data`, les informations nécessaires à l'identification de la requête associée, le contenu de la réponse ainsi qu'une signature garantissant son authenticité. La table suivante décrit la structure de ce champ `data` :

Champ	Description
`requestIdParent`	Identifiant unique de la requête principale permettant d'associer la réponse à une demande globale
`subRequestId`	Identifiant de la sous-requête à laquelle cette réponse correspond
`responseData`	Données retournées par le fournisseur cloud (ex. : lien vers une ressource, résultat d'un calcul, accusé de réception, etc.)
`signature`	Signature cryptographique de la réponse, générée avec la clé privée du fournisseur, garantissant l'intégrité et l'authenticité des données

TABLE 3.6 – Champs du payload `data` dans la transaction de réponse fournisseur

Afin d'enregistrer sa réponse dans la blockchain, le fournisseur cloud soumet une transaction contenant un appel à la fonction `submitResponse` du smart contract, avec les paramètres suivants :

```
data: submitResponse(requestIdParent, subRequestId, responseData, signature)
```

3.9 Intégration des smart contracts

Dans l'architecture décentralisée proposée, les smart contracts constituent le cœur logique du processus de contrôle d'accès. Déployés sur une blockchain publique ou privée, ils assurent un traitement automatisé, transparent et immuable des réponses émises par les différents fournisseurs cloud à la suite des requêtes utilisateur.

3.9.1 Fonctionnalités couvertes

Les smart contracts remplissent plusieurs fonctions essentielles :

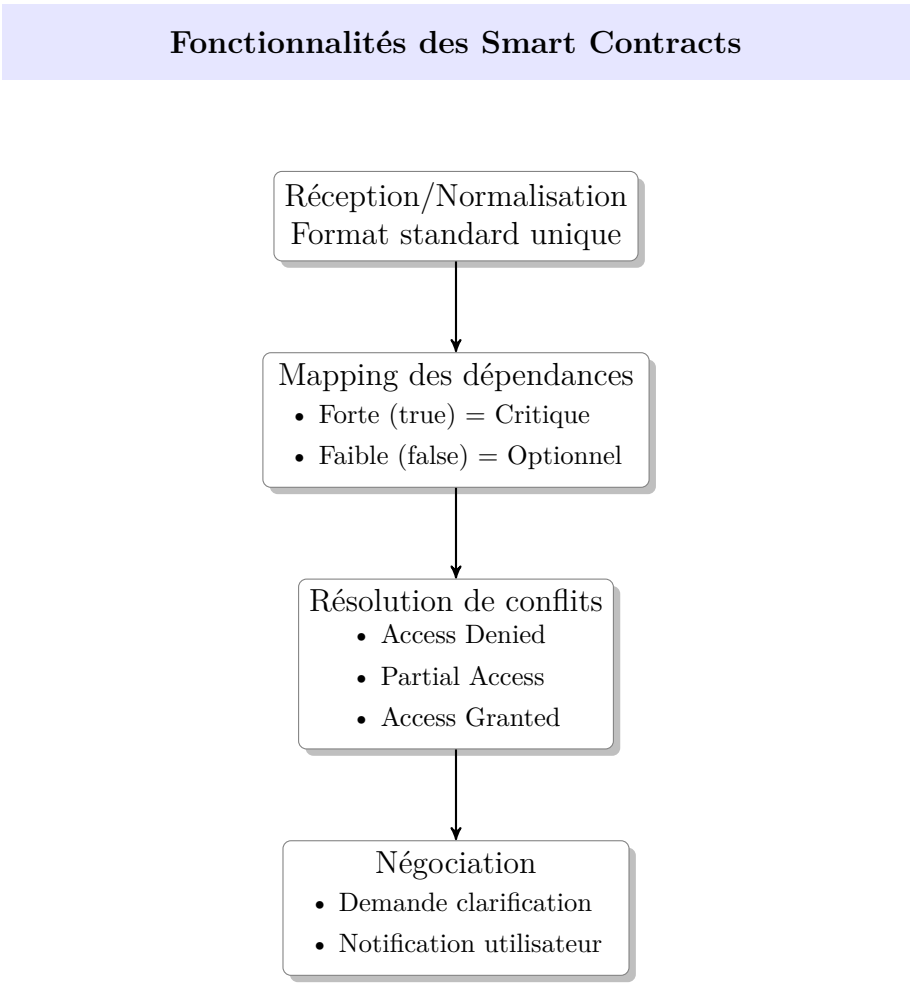

FIGURE 3.4 – Architecture fonctionnelle des smart contracts (ordre séquentiel)

3.9.2 Réception Vérification et Normalisation

3.9.2.1 Réception

Le smart contract, contient une logique de compteur d'attente quand le compteur **Listing 1.3** aurait atteint le nombre de reponse attendue il declenche automatiquement son exécution. *Pour plus de détails, voir l'.3.*

```
mapping(string => uint8) public compteur_reponses;
mapping(string => uint8) public total_attendu;

function enregistrerReponse(string memory id_requete) public {
    require(msg.sender == fournisseurAutorise);
    compteur_reponses[id_requete]++;
    if (compteur_reponses[id_requete] == total_attendu[id_requete]) {
        executerDecisionFinale(id_requete);
    }
}
```

Listing 3.3 – Smart Contract de suivi des réponses

3.9.2.2 Vérification

Aprés que toutes les transactions ont atteint le nombre attendue, le smart contract est déclenché automatiquement

Le smart contract commence par vérifier l'authenticité des réponses. Chaque réponse transmise est accompagnée d'une signature cryptographique. Le smart contract vérifie cette signature comme montré dans **Listing 1.4** afin de s'assurer que les données n'ont pas été altérées et qu'elles proviennent bien d'une source autorisée. Cela garantit l'intégrité et la non-répudiation des informations reçues.

— Chaque sous-réponse est signée numériquement par le fournisseur avec sa clé privée (signature cryptographique).

— Le smart contract vérifie ces signatures avec la clé publique de ce meme fournisseur en appelant une fonction comme :

```
require(verifySignature(response.signature, ...), "Invalid signature
    ");
```

Listing 3.4 – Vérification de signature

- Cela garantit que les réponses proviennent bien d'un canal sécurisé et autorisé.
- Si la signature est invalide la réponse est ignorée ou rejetée

3.9.2.3 Normalisation

Le module de normalisation a pour objectif d'uniformiser les réponses variées et hétérogènes reçues des différents fournisseurs cloud afin de faciliter leur traitement automatique dans le smart contract. En effet, chaque fournisseur peut exprimer une même décision d'accès de manière différente, Le module de normalisation convertit donc ces différentes réponses en une valeur standard interne unique. Comme exprimé dans **L'Algorithme 1**

Algorithm 1 Normalisation des réponses d'accès

```
function normalizeResponse(response)
if response == "Oui" ∨ response == "Ok" ∨ response == "Allow" ∨ response == "Access Autorisé" ∨ response == "Granted" then
    ▷ Cas d'accès autorisé standard
    normalizedResponse ← "ACCESS_GRANTED"
else if response == "Non" ∨ response == "Refusé" ∨ response == "Denied" then
    ▷ Cas de refus explicite
    normalizedResponse ← "ACCESS_DENIED"
else if response == "Autorisé avec conditions" ∨ response == "Conditional Access" then
    ▷ Cas d'accès conditionnel
    normalizedResponse ← "ACCESS_GRANTED_WITH_CONDITIONS"
else
    ▷ Cas par défaut pour réponse non reconnue
    normalizedResponse ← "UNKNOWN"
end if
return normalizedResponse                    ▷ Retourne la réponse normalisée
```

3.9.3 Mapping des dépendances

Dans un environnement multi-cloud, certains services sont interdépendants : par exemple, un service d'application peut dépendre d'une base de données ou d'un service d'authentification déployé ailleurs. La gestion de ces dépendances entre sous-requêtes est donc cruciale pour assurer la cohérence, la sécurité et l'efficacité du système.

Chaque sous-requête envoyée à un fournisseur cloud représente un composant individuel du service global demandé. Or, toutes les sous-requêtes n'ont pas le même poids fonctionnel dans la réussite de l'opération. Par conséquent, il est indispensable de classifier les dépendances en deux grandes catégories :

Dépendances fortes (ou critiques)
: Ces services sont indispensables. Si l'un d'eux est refusé ou indisponible, l'accès global au service principal ne peut être garanti.

  Exemple : une base de données nécessaire à l'application.

Dépendances faibles (ou optionnelles)
: Ces services sont secondaires. Leur absence peut affecter le confort ou la performance, mais n'empêche pas l'exécution principale.

  Exemple : un service de journalisation, une API de localisation.

Ces informations de dépendance sont extraites dynamiquement à partir de la transaction cloud stockée sur la blockchain. Lorsqu'un utilisateur fait une demande, une transaction contenant les sous-services requis est créée, avec pour chacun un champ indiquant son niveau de dépendance (true ou false).

Le smart contract, lors du traitement, lit cette transaction (ou bloc contenant l'ID de la requête) pour récupérer les métadonnées associées à chaque service, y compris son niveau de dépendance. Cela permet une prise de décision basée sur des données contextualisées, spécifiques à chaque requête.

3.9.4 Résolution de conflits

Dans une architecture décentralisée multi-cloud, chaque service demandé par un utilisateur peut être hébergé sur un fournisseur différent (AWS, Azure, GCP, etc.). Chaque sous-service est donc évalué individuellement par le fournisseur concerné. Le smart contract, après avoir reçu, vérifié et normalisé les réponses, applique une logique d'agrégation de décision globale basée sur :*Pour plus de détails, voir l'**??**.*

— Réponse normalisée (ACCÈS_AUTORISÉ, REFUSÉ, AUTORISÉ_AVEC_CONDITIONS)

— Niveau de dépendance (forte/faible)

Comme exprimé dans **L'Algorithme 2**

3.9.5 Négociation

3.9.5.1 Contexte et justification

Dans une architecture de contrôle d'accès décentralisée multi-cloud, les décisions d'accès sont basées sur l'agrégation des réponses des différents fournisseurs cloud. Cependant, certains cas particuliers peuvent rendre la décision finale ambiguë ou incertaine. Ces situations nécessitent une intervention externe sous forme de négociation.

Scénarios déclencheurs de la négociation

Le module de résolution du Smart Contract déclenche une négociation dans les cas suivants :

- Une ou plusieurs sous-réponses sont égales à `ACCESS_GRANTED_WITH_CONDITIONS`, mais sans détails exploitables sur les conditions.
- Des services critiques sont marqués comme `DENIED`, mais sans informations sur leur niveau de dépendance dans la transaction blockchain.
- Plusieurs fournisseurs donnent des réponses incohérentes ou contradictoires pour un même service (ex : AWS = `GRANTED`, Azure = `DENIED`).
- Un ou plusieurs services sont absents de la table de dépendance, rendant impossible la résolution logique.

3.9.5.2 Mécanisme déclenché par le Smart Contract

Lorsque l'un des scénarios ci-dessus est détecté, le Smart Contract :

1. Interrompt l'évaluation automatique.
2. Classe l'état comme « En attente de clarification ».
3. Génère une réponse structurée vers la DAPI contenant :
   - le service concerné,
   - la réponse ambiguë,
   - le type de blocage identifié.
4. Émet une notification à l'utilisateur via le la dapi, invitant à :
   - reformuler la requête,
   - accepter une réponse partielle,
   - fournir des politiques de dépendances complémentaires.

Algorithm 2 Algorithme de décision d'accès multi-cloud (avec négociation)

Require: **responses**[] : tableau des réponses normalisées
dependencies[] : tableau des dépendances (true = forte, false = faible)
Ensure: decision ∈ {ACCESS_GRANTED, ACCESS_DENIED, PARTIAL_ACCESS, ACCESS_WITH_CONDITIONS}

Variables :
hasStrongDenied ← $false$
hasWeakDenied ← $false$
hasConditional ← $false$
hasContradictions ← $false$
missingDependencies ← $false$
for $i \leftarrow 0$ **to** longueur(responses)-1 **do**
  if responses[i] == ACCESS_DENIED **then**
    if dependencies[i] == true **then**
      hasStrongDenied ← $true$
    else
      hasWeakDenied ← $true$
    end if
  else if responses[i] == ACCESS_GRANTED_WITH_CONDITIONS **then**
    hasConditional ← $true$
  end if
end for
hasContradictions ← detectContradictions(responses)
missingDependencies ← detectMissingDependencies(responses, dependencies)
{Scénarios nécessitant une négociation}
if (hasConditional **and not** hasDetailedConditions(responses)) **or** hasContradictions **or** missingDependencies **or** (hasStrongDenied **and** criticalServiceUnknown()) **then**
  return ACCESS_WITH_CONDITIONS
end if
{Logique finale si pas de négociation}
if hasStrongDenied **then**
  return ACCESS_DENIED
else if hasConditional **then**
  return ACCESS_WITH_CONDITIONS
else if hasWeakDenied **then**
  return PARTIAL_ACCESS
else
  return ACCESS_GRANTED
end if

Exemple de réponse JSON renvoyée

```
{
  "status": "NEGOTIATION_REQUIRED",
  "ambiguous_service": "S3",
  "reason": "ACCESS_GRANTED_WITH_CONDITIONS sans clarification",
  "suggested_action": "besoin de plus d'informations."
```

```
}
```

3.9.5.3 Retour à la blockchain

Une fois la négociation terminée (côté utilisateur ou fournisseur), une nouvelle transaction est soumise à la blockchain contenant les données corrigées ou clarifiées. Le Smart Contract peut alors réévaluer la situation et produire une décision finale.

3.9.6 Désicion final du smart contract

Lorsque le smart contract a terminé l'analyse des sous-réponses reçues via une transaction envoyée par la DAPI, il ne peut pas créer une nouvelle transaction pour transmettre sa décision. À la place, il utilise un mécanisme interne appelé **événement (event)**, qui permet de publier une information directement dans le bloc en cours. Par exemple, le contrat peut émettre :

```
event DecisionFinale(string idRequete, string decision);
```

Listing 3.5 – Émission d'un événement Solidity

Cette instruction ne crée pas une nouvelle transaction, mais inscrit l'événement comme un log immuable dans le bloc blockchain contenant l'exécution du smart contract. Ce log peut ensuite être consulté ou écouté par des entités externes (comme la DAPI ou l'application cliente), assurant ainsi une communication fiable, traçable et sécurisée de la décision finale.

3.10 Préparation du bloc sur la blockchain

3.10.1 Création du bloc par le validateur

Chaque nœud conserve un *mempool* local contenant les transactions non encore validées. Le validateur du prochain bloc surveille ce *mempool*, filtre celles ayant un `request_id_parent` commun, et vérifie via le *smart contract* si toutes les réponses attendues sont présentes. Lorsque c'est le cas, il regroupe les transactions pertinentes, élimine les doublons, puis assemble un bloc conforme au protocole (en-tête, corps, signatures, métadonnées). Ce processus assure que le contrat intelligent ne s'exécute que lorsque toutes les sous-requêtes sont traitées, garantissant ainsi cohérence et intégrité.

3.10.2 Structure d'un bloc

Après sélection, le validateur procède à l'assemblage du nouveau bloc, selon la structure standard définie par le protocole :

TABLE 3.7 – Structure d'un bloc dans la blockchain

Structure d'un bloc		
Section	**Élément**	**Description**
5*En-tête du bloc (Block Header)	Identifiant unique	Numéro de séquence du bloc dans la chaîne
	Hachage du bloc précédent	Lien cryptographique avec le bloc précédent (chaînage)
	Horodatage (Timestamp)	Date et heure de création du bloc
	Identifiant du validateur	Clé publique du nœud validateur qui a validé ce bloc
	Racine Merkle	Résumé cryptographique de toutes les transactions incluses
4*Corps du bloc (Block Body)	Transactions sélectionnées	Liste des transactions exécutées dans ce bloc
	Logs / Événements	Informations générées par les smart contracts (ex. `DecisionFinale`)
	Signatures	Signatures cryptographiques prouvant l'origine de chaque transaction
	Métadonnées	Informations d'audit facultatives : RequestID, ID utilisateur, timestamp
3*Contrôle de cohérence	Unicité des RequestID	Vérifie qu'aucun identifiant de requête n'est dupliqué
	Gestion des doublons	Ne conserve que la première transaction valide par requête
	Transactions redondantes	Les doublons sont rejetés ou renvoyés dans le mempool

3.11 Vérification et validation du bloc

3.11.1 Identifié des nœuds validateur des blocs

Dans cette architecture, un *smart contract de gouvernance* gère dynamiquement la liste des validateurs. Il permet l'ajout ou la révocation de validateurs sur la base d'un

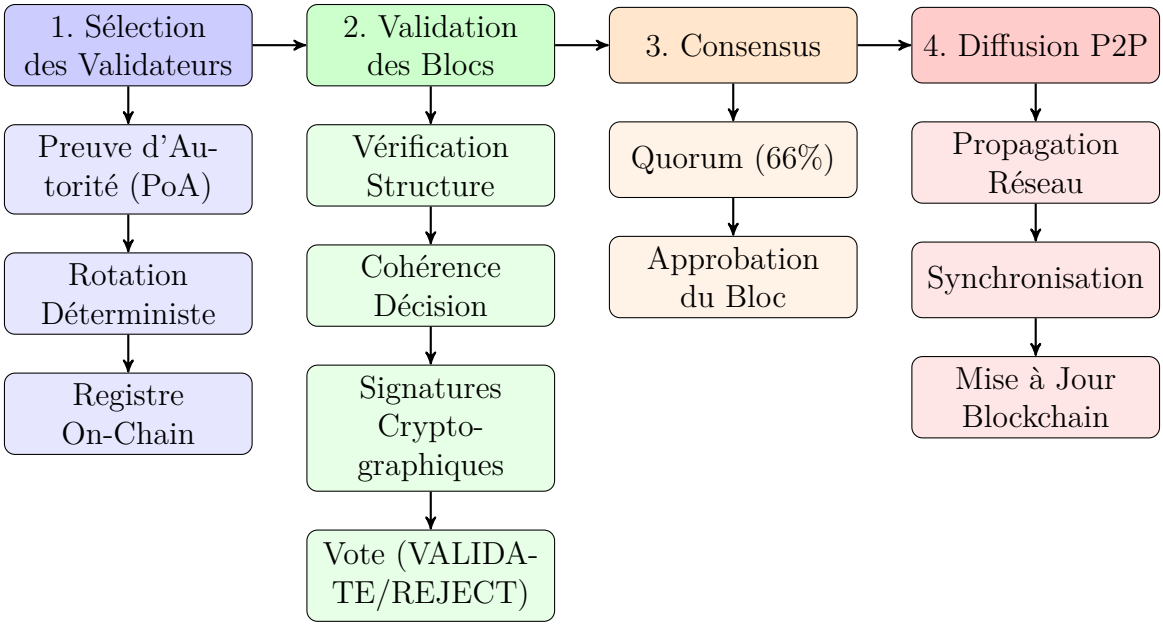

FIGURE 3.5 – Schéma du processus complet de validation et diffusion des blocs

vote ou d'un consensus, avec possibilité d'automatisation partielle. Les validateurs sont responsables de la validation des transactions, de la création des blocs, de l'exécution des smart contracts et du maintien du consensus.

La sélection du producteur de blocs suit un mécanisme par *slots* temporels, chaque validateur étant désigné pour un créneau spécifique. En cas d'indisponibilité, un mécanisme de secours permet à un autre validateur de prendre le relais.

Un système de sélection dynamique désigne un sous-groupe temporaire de validateurs pour chaque bloc, selon des règles définies par un smart contract. Cela permet une validation efficace, résiliente et distribuée.

Pour plus de détails sur les mécanismes internes de gouvernance, de sélection et de fonctionnement des validateurs dans l'architecture PoA, voir l'.2.

3.11.2 Consensus et Quorum

Avant qu'un bloc soit ajouté à la blockchain, il doit être validé collectivement par les nœuds validateurs. Ce processus repose sur deux notions clés :

— **Consensus** : mécanisme d'accord entre les validateurs sur la validité du bloc.

— **Quorum** : seuil minimal (ex. : 66%) de votes favorables requis pour l'acceptation.

Voir ca dans la *Figure 1.6*

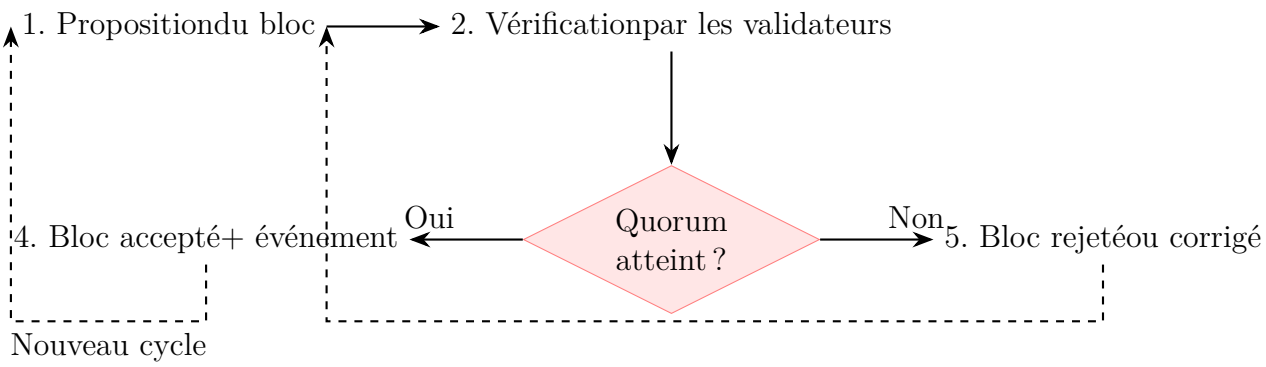

FIGURE 3.6 – Processus simplifié de validation des blocs en PoA

3.11.3 Diffusion des blocs sur le réseaux blockchain

Une fois qu'un bloc a été validé par le quorum de validateurs désignés (clouds), il est diffusé sur l'ensemble du réseau blockchain afin d'être ajouté à la chaîne commune. Ce processus garantit que toutes les parties prenantes (fournisseurs, DAPI, clients, etc.) disposent d'une copie synchronisée et à jour de l'état du registre.

— Le bloc signé est diffusé en mode pair-à-pair (P2P) vers tous les nœuds du réseau

— Chaque nœud reçoit, vérifie les signatures, et ajoute le bloc à sa copie locale de la blockchain

— Si un nœud est désynchronisé, il peut récupérer les blocs manquants grâce à un mécanisme de *replay* ou rattrapage de chaîne (*block replay/sync*)

3.11.4 Mise a jour de la blockchain

Une fois qu'un bloc validé a été diffusé à l'ensemble des nœuds via le réseau pair-à-pair (P2P), chaque nœud, qu'il soit validateur ou non, procède à sa propre mise à jour locale de la blockchain.

Cette étape comprend :

1. **Vérification finale du bloc reçu** : chaque nœud vérifie les signatures collectives, le hachage du bloc précédent, et l'intégrité des métadonnées (timestamp, ID, etc.).
2. **Ajout du bloc à la chaîne locale** : si toutes les vérifications sont concluantes, le bloc est intégré dans la copie locale de la blockchain, maintenant ainsi la cohérence de l'historique global.
3. **Mise à jour de l'état du système** : les données contenues dans le bloc (décisions d'accès, logs, événements) sont appliquées. Par exemple, les statuts d'accès des utilisateurs ou la traçabilité des requêtes sont mis à jour selon les décisions rendues.
4. **Gestion de la synchronisation** : si un nœud constate un décalage (bloc manquant ou retard), il peut initier un processus de rattrapage (block replay/sync) pour restaurer une version conforme et complète de la chaîne.

Cette mise à jour garantit que tous les nœuds participants partagent un état cohérent et vérifiable de la blockchain, assurant ainsi la fiabilité des décisions futures, la transparence des accès et l'auditabilité complète du système.

Table 3.8 – Rôles dans le processus de validation

Acteur	**Rôle**	**Étapes Clés**
Smart Contract	Générer la décision	Crée événement (event) Diffusion P2P
Mempool	Stockage temporaire	Recevoir les TX Prévenir les doublons
Validateur	Construction du bloc	Sélection TX Vérifier signatures
Réseau P2P	Consensus distribué	Propagation des blocs Collecte des votes (66%)
Blockchain	Registre immuable	Ajout des blocs validés Synchronisation globale

Dans la *Figure 1.7* suivante nous pouvant voir les étapes a passer pour la validation du bloc blockchain crée.

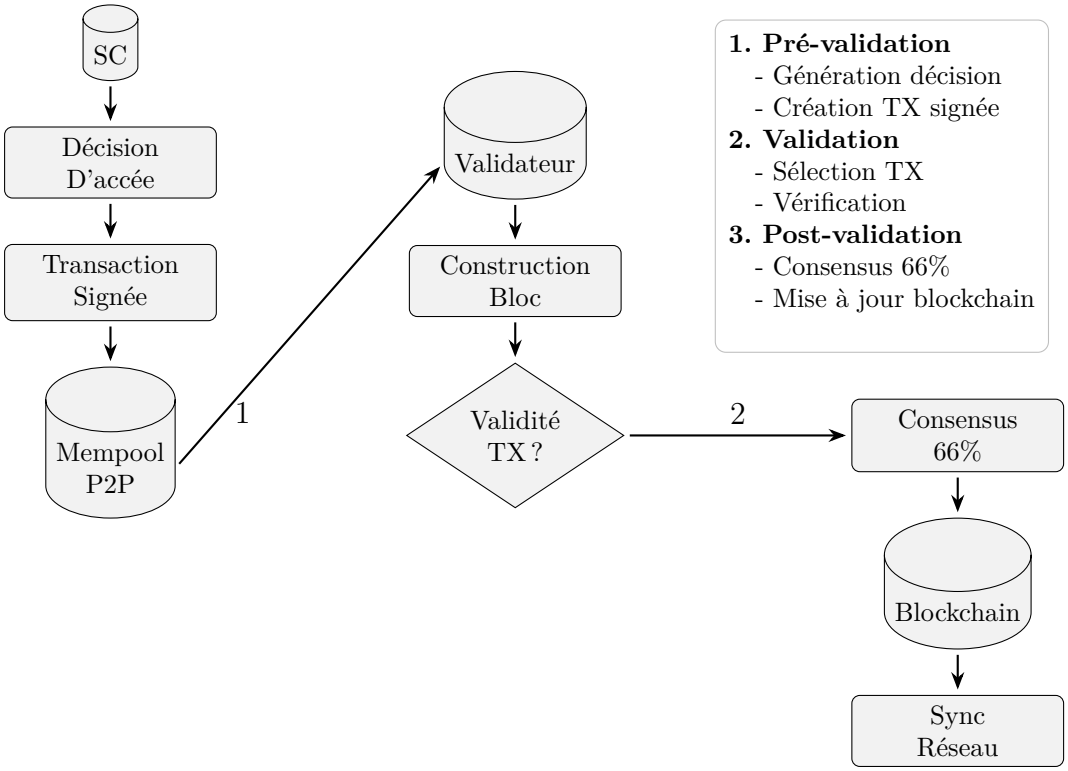

Figure 3.7 – Validation Blockchain

3.12 Retour de la réponse au client

Le smart contract écrit le résultat final on-chain : Après avoir reçu toutes les réponses (via request_id_parent et sub_request_id), Il effectue le calcul/fusion/validation, Et stocke le résultat final dans une variable d'état. Le dAPI ou un Airnode surveille le smart contract : Il scrute les événements émis par le contrat *event*
DecisionFinale (string idRequete , string decision).
Le dAPI notifie le client : Une fois que le résultat est disponible on-chain et que le bloc est validé, le dAPI peut : Soit pousser la réponse via un webhook, Soit le client fait un polling (interrogation périodique) sur le dAPI.

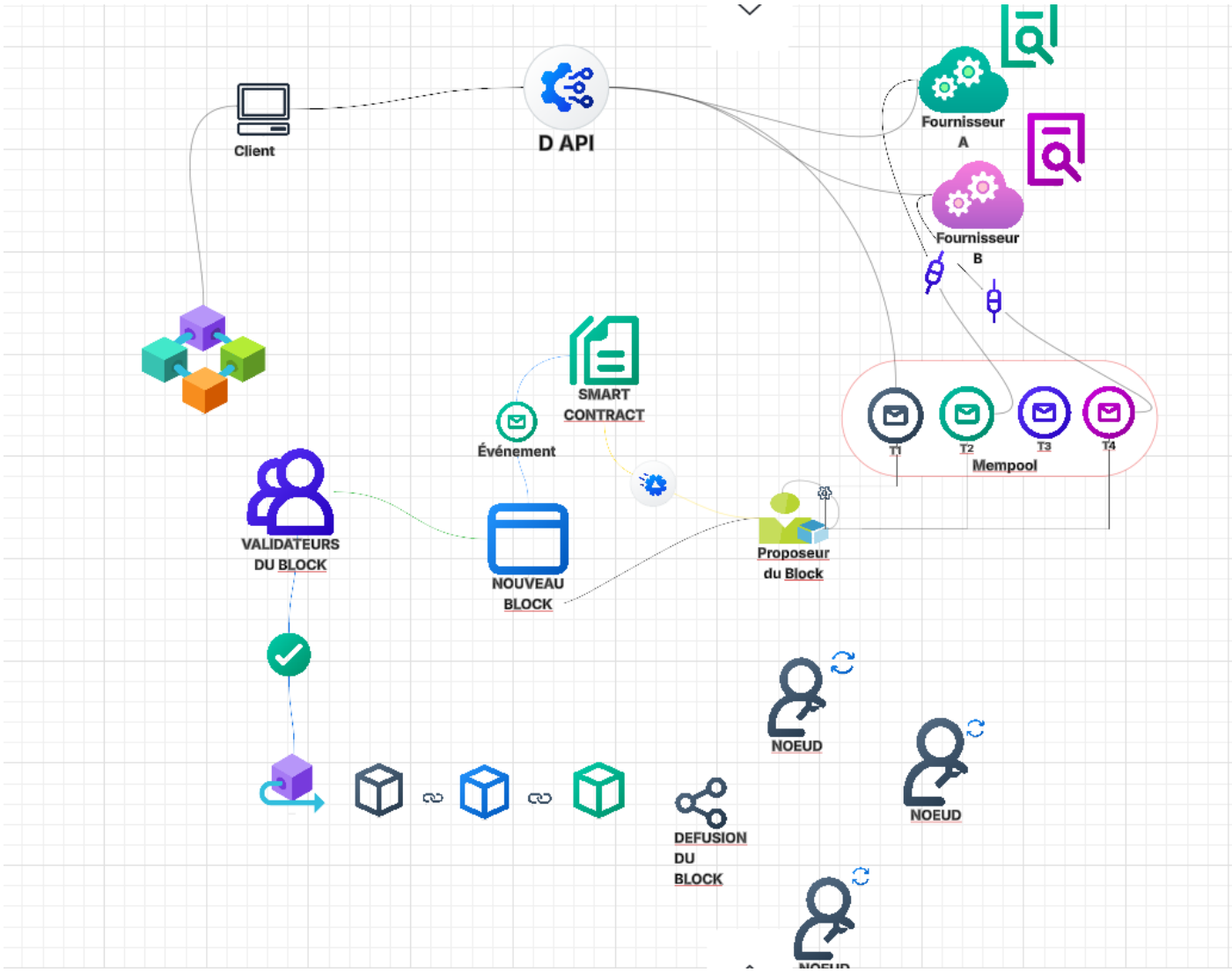

FIGURE 3.8 – Processus de gestion décentralisée des transactions dans l'architecture proposée

La figure 3.8 illustre le **flux global d'exécution** dans l'architecture décentralisée proposée pour le contrôle d'accès multi-cloud basé sur la blockchain.

Le processus débute par la soumission d'une requête par le **client** via le **dAPI** (Decentralized API), qui décompose cette requête en sous-requêtes spécifiques destinées aux différents **fournisseurs cloud** (Fournisseur A et Fournisseur B).

Chaque fournisseur analyse la sous-requête reçue et publie une **transaction signée** dans le **mempool**, un espace temporaire où les transactions en attente de validation sont stockées.

Le **proposeur de bloc** sélectionne les transactions pertinentes liées à une même requête globale, identifiées par un `request_id_parent`, puis soumet ces transactions au **smart contract**.

Le smart contract vérifie les signatures, applique les règles d'accès, et émet une **décision finale** sous forme d'un **événement blockchain**.

Le **nouveau bloc** est ensuite constitué par le proposeur et soumis à l'approbation des **validateurs**, qui assurent la cohérence et l'intégrité des transactions avant d'intégrer le bloc dans la blockchain.

Enfin, le bloc validé est diffusé auprès de l'ensemble des **nœuds du réseau** pour mise à jour du registre distribué et synchronisation globale.

Cette architecture garantit la **traçabilité**, la **sécurité**, et une **décision automatique et transparente** dans un environnement multi-cloud décentralisé.

Analyse Critique

L'architecture **centralisée**, bien qu'encore largement utilisée, présente des limites structurelles face aux environnements multi-cloud modernes et aux exigences croissantes en matière de transparence et de sécurité. Sa dépendance à un tiers de confiance (le broker) constitue une vulnérabilité majeure, notamment en cas de défaillance ou d'attaque ciblée. En revanche, sa simplicité et sa rapidité en font une solution efficace pour des environnements cloud simples et peu critiques.

À l'opposé, l'architecture **décentralisée proposée** offre une solution innovante qui s'appuie sur la blockchain pour garantir une traçabilité complète, une sécurité renforcée et une résilience accrue. L'absence de point unique de défaillance améliore significativement la robustesse globale du système. Cependant, cette approche n'est pas sans coût : elle introduit une complexité technique notable, des exigences accrues en termes de ressources, et une légère latence due aux mécanismes de consensus blockchain.

En somme, l'architecture décentralisée se positionne comme une solution prometteuse pour des systèmes critiques, multi-clouds et sensibles à la sécurité, tandis que l'architecture centralisée reste adaptée pour des cas d'usage moins complexes et à faible niveau d'exigence de confiance.

3.13 Conclusion

DAccess-ChainMC présentée dans ce chapitre constitue la base technique du système de contrôle d'accès décentralisé que nous proposons. En combinant une passerelle décentralisée (dAPI), des contrats intelligents, une infrastructure blockchain, et une intégration directe des fournisseurs via Airnode, cette solution permet de répondre efficacement aux enjeux complexes du contrôle d'accès en environnement multi-cloud. Chaque composant remplit un rôle spécifique et coordonné : la DAPI orchestre les requêtes, les fournisseurs assurent l'évaluation locale, le smart contract centralise et harmonise les décisions, tandis que la blockchain garantit transparence, traçabilité et immutabilité. Cette architecture modulaire et interopérable ouvre la voie à une gouvernance plus équitable des accès, tout en assurant sécurité, scalabilité et conformité. Le chapitre suivant illustrera concrètement son fonctionnement à travers un scénario détaillé de bout en bout.

Chapitre 4

Scénario de Simulation dans un Environnement Multi-Cloud Décentralisé

4.1 Introduction

Dans ce chapitre, nous présentons un scénario de simulation illustrant l'application d'un système de contrôle d'accès décentralisé basé sur la blockchain dans un environnement multi-cloud. Nous nous concentrons sur une situation où un client souhaite louer des ressources auprès de plusieurs fournisseurs cloud pour les utiliser de manière conjointe. Une dépendance fonctionnelle est introduite entre les services : le service 2 dépend du bon fonctionnement du service 1, mais l'inverse n'est pas requis.

4.2 Présentation Générale du Scénario

4.2.1 Objectif du Scénario

L'objectif est de simuler la réservation et la coordination de ressources multi-cloud via un smart contract déployé sur une blockchain. Ce contrat intelligent contrôle la cohérence des autorisations et assure la traçabilité des décisions des différents fournisseurs cloud.

4.2.2 Hypothèses de Travail

— Le client souhaite réserver simultanément des ressources chez deux fournisseurs cloud : AWS et Azure.
— Le service proposé par AWS est considéré comme **critique** pour l'exécution globale.
— Le service proposé par Azure dépend du bon fonctionnement des ressources AWS.
— La plateforme utilise un dAPI décentralisé pour communiquer avec les fournisseurs cloud.
— Un smart contract orchestre la validation des autorisations.

4.3 Acteurs Impliqués

— **Client** : Demandeur des ressources cloud multi-fournisseurs.

— **dAPI** : Interface décentralisée responsable de la gestion des requêtes et de leur décomposition.

— **Fournisseur 1 (AWS)** : Fournit des machines virtuelles critiques.

— **Fournisseur 2 (Azure)** : Fournit un espace de stockage dépendant des machines virtuelles d'AWS.

— **Smart Contract** : Exécute la logique de contrôle d'accès et de dépendance.

— **Validateur Blockchain** : Regroupe et valide les transactions dans un bloc.

4.4 Déroulement du Scénario

4.4.1 Soumission de la Demande par l'utilisateur

Le client initie la requête multi-cloud suivante :

— Location de **2 VM critiques chez AWS**.

— Réservation de **100 Go de stockage chez Azure**.

— Durée : **72 heures**.

La requête globale est identifiée par `request_id_parent = req23`.
Dans la Figure 4.1 nous présentant un exemple d'interface utilisateur pour la demande d'accés. Voici dans Listing 4.1 la structure de la requete principale.

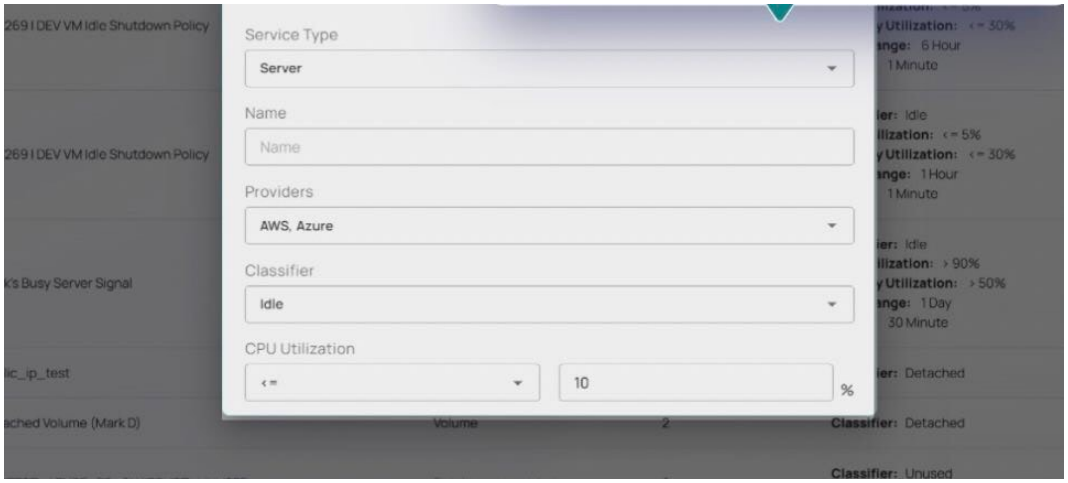

FIGURE 4.1 – Exemple interface utilisateur [37]

```
{  "id_requete": "req123"
  "identifiant_utilisateur": "client123",
  "jeton_authentification": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",
  "horodatage": "2025-06-25T14:00:00Z",

  "services_demand s": [
    {
      "fournisseur": "AWS",
      "type_service": "VM",
      "param tres": {
        "cpu": "4vCPU",
        "m moire": "16GB",
        "stockage": "100GB SSD"
```

```
      }
    },
    {
      "fournisseur": "Azure",
      "type_service": "Base de Donn es",
      "param tres": {
        "type_bd": "PostgreSQL",
        "capacit ": "50GB"
      }
    }
  ],

  "contraintes": {
    "co t_max": 500,
    "r gion": "Europe",
    "conformit ": "RGPD"
  },

  "fournisseurs_preferes": ["AWS", "Azure"]
}
```

Listing 4.1 – Structure de la Requête Globale Client

4.4.2 Décomposition des Sous-Requêtes par dAPI

Le dAPI décompose la demande en deux sous-requêtes distinctes :les Listing 4.2 et Listing 4.3 illustres les sous-requêtes envoyées aux fournisseurs .
Sous-requête 1 : demande une VM, indépendante des autres services.

```
{
  "id_sous_requete": "subreq001",
  "id_requete": "req123",
  "timestamp": "2025-06-25T14:05:00Z",

  "service_spec": {
     "fournisseur": "AWS",

    "service_type": "VM",
    "service_parameters": {
      "cpu": "4vCPU",
      "m moire": "16GB",
      "stockage": "100GB SSD"
    },
    "dependency_level": "independant"
  },

  "security": {
    "signature": "3045022100dff9a8b6f3d3cbe2a7a65c6234f49d8971"
  }
}
```

Listing 4.2 – Sous-requête DAPI pour le service VM

Sous-requête 2 : demande une base de données, dépendante de la VM de la sous-requête 1.

```
{
  "id_sous_requete": "subreq002",
```

```
  "id_requete": "req123",
  "timestamp": "2025-06-25T14:05:10Z",

  "service_spec": {

    "service_type": "Base de Donn es",
    "service_parameters": {
      "type_bd": "PostgreSQL",
      "capacit ": "50GB"
    },
    "dependency_level": "depend de subreq001"
  },

  "security": {
    "signature": "3046022100adf8e7c3b5d2c4e9f7a4b3c2a1e0d4f9c3a7b5c6"
  }
}
```

Listing 4.3 – Sous-requête DAPI pour le service Base de Données

4.4.3 Traitements locales des sous requêtes par les Fournisseurs

Chaque fournisseur procède à une vérification locale de la demande. Dans ce scénario, le service critique (AWS) ne fournit pas immédiatement l'accès total, mais conditionne l'accès à la satisfaction de certaines exigences techniques et de sécurité.

4.4.3.1 Contrôle effectué par AWS (Service Critique) avec Accès Conditionnel

AWS impose une procédure d'accès conditionnel structurée en plusieurs étapes :

1. **Vérification des droits d'accès initiaux** : AWS confirme que le client possède un abonnement Premium valide, permettant de soumettre une demande de location pour des machines virtuelles critiques.
2. **Analyse de conformité de la configuration** : AWS exige que le client configure un environnement sécurisé respectant des standards spécifiques (par exemple : déploiement d'un VPN [1] , chiffrement des volumes de stockage, mise en place de groupes de sécurité stricts).
3. **Activation d'un réseau privé** : AWS impose que le client déploie les VM dans un réseau privé (VPC – Virtual Private Cloud) et configure un accès limité par adresse IP approuvée.
4. **Validation de l'authentification multi-facteur (MFA)** [2] : AWS demande que le compte client utilise l'authentification multi-facteur pour toutes les connexions associées aux VM réservées.
5. **Remplissage d'un journal de conformité** : Le client doit soumettre une preuve de configuration et remplir un journal d'audit qui sera vérifié par AWS avant l'activation finale.

1. VPN :Virtual Private Network .
2. MFA : Multi-Factor Authentication

6. **Décision** : AWS accepte la demande sous condition. La transaction soumise sur la blockchain précise que l'accès définitif sera accordé uniquement après la validation de ces étapes.

Résultat : *Accès conditionnel accordé par AWS sous réserve de satisfaire les exigences de sécurité et de configuration.*

4.4.3.2 Contrôle effectué par Azure (Service Dépendant)

Azure effectue les vérifications suivantes :

1. **Vérification des quotas disponibles** : Azure détecte que le quota de stockage disponible est insuffisant pour la durée et la taille demandées.
2. **Vérification des politiques de sécurité** : Azure exige que le client utilise une connexion VPN sécurisée et que l'authentification multi-facteur soit activée.
3. **Vérification de la dépendance** : Azure s'assure que le service AWS (critique) a bien validé la demande. Même en cas de dépendance active, Azure rejette la requête si les quotas de stockage sont insuffisants.
4. **Décision** : Azure refuse la demande de location. La transaction de refus est soumise sur la blockchain.

Résultat : *Réservation refusée par Azure en raison de quotas insuffisants.*

4.4.3.3 Synthèse des Contrôles

— **AWS (Service Critique)** : Accès conditionnel accordé. Le client doit compléter les étapes définies avant l'activation finale des VM.

— **Azure (Service Dépendant)** : Accès refusé en raison de quotas insuffisants.

Chaque décision est soumise sous forme de transaction signée sur la blockchain et associée à `request_id_parent = REQ-123`. La décision finale du smart contract prendra en compte le caractère conditionnel de l'accès au service critique.

4.4.4 Soumission des Transactions sur la Blockchain

Dans le cadre du scénario simulé, chaque acteur interagit avec la blockchain en soumettant une transaction signée, liée à la requête globale par l'identifiant `request_id_parent`. La transaction initiale soumise par le dAPI (Tableau 4.1) enregistre les informations de la requête principale et des sous-requêtes associées. Les fournisseurs cloud, AWS et Azure, répondent via des transactions distinctes (Tableau 4.2 et Tableau 4.3) qui reflètent leurs décisions locales, qu'il s'agisse d'une acceptation conditionnelle ou d'un refus. Lorsque des conditions sont imposées, le client intervient à son tour en soumettant une transaction de réponse (Tableau 4.4) pour accepter ou refuser les exigences fixées. Ces transactions assurent une traçabilité complète et garantissent la cohérence du processus d'autorisation décentralisé.

Table 4.1 – Transaction initiale générée par le dAPI

Transaction dAPI	
Attribut	**Valeur**
transaction_id	0x9a45d3abef
from	0xF1e2B8c9F765a
to	0xDeAdBeEf1
value	0 ETH
requestIdParent	**REQ-123**
nbSubrequests	**2**
subRequestIds	subreq001, subreq002
providerIds	AWS, Azure
timestamp	2025-06-25T14 :05 :00Z
client	0xF1e2B8c9F765a
metadata	{coût : 500 USD, région : Europe, conformité : RGPD}
functionSelector	**registerRequest**
contractAddress	0xDeAdBeEf1
signature	3045...f2a1
nonce	12
gas	100000
gasPrice	1 Gwei

Table 4.2 – Transaction de réponse AWS : Acceptation conditionnelle

Transaction AWS	
Attribut	**Valeur**
transaction_id	0x7b91a1b2cd
from	0xAWSFournisseur01
to	0xDeAdBeEf1
value	0 ETH
requestIdParent	**REQ-123**
subRequestId	subreq001
vm_instance_id	i-1234567890abcdef0
status	**Conditionnel**
requirements	{Configuration de sécurité renforcée, Audit préalable des accès}
signature	3046...d5c4
nonce	5
gas	50000
gasPrice	1 Gwei

Table 4.3 – Transaction de réponse Azure : Refus

Transaction Azure	
Attribut	**Valeur**
transaction_id	0x6f81c2d4ef
from	0xAzureFournisseur02
to	0xDeAdBeEf1
value	0 ETH
requestIdParent	**REQ-123**
subRequestId	subreq002
status	**Refusé**
reason	**quotas insuffisants.**
signature	3045...f2a1
nonce	6
gas	50000
gasPrice	1 Gwei

Table 4.4 – Transaction de réponse du client : Acceptation des conditions AWS

Transaction Client	
Attribut	**Valeur**
transaction_id	0x4b23c1a9fe
from	0xClientAddress
to	0xDeAdBeEf1
value	0 ETH
requestIdParent	**REQ-123**
subRequestId	**subreq001**
confirmationData	{accepted : true, details : Conditions de sécurité validées}
signature	**3045...e2a9**
nonce	15
gas	50000
gasPrice	1 Gwei

4.4.5 Gestion de l'Accès Conditionnel par le Smart Contract

À l'issue de la première vérification locale, le fournisseur AWS a accordé un accès conditionnel. Le smart contract inscrit cette condition dans la blockchain et passe l'état de la requête dans un mode **"en attente de validation des conditions"**.LaFigure 4.2 illustre le processus suivis

4.4.5.1 Cas 1 : Le Client Satisfait les Conditions

Le client procède à l'exécution des étapes imposées par AWS :

— Configuration d'un réseau privé (VPC) [3]. avec restriction IP.

— Activation de l'authentification multi-facteur (MFA).

— Mise en place des règles de sécurité conformes aux exigences.

— Soumission des journaux de conformité à AWS.

Vérification finale : AWS analyse les journaux soumis et valide la bonne configuration de l'environnement.

Décision : AWS publie une nouvelle transaction sur la blockchain indiquant que **les conditions sont satisfaites.**

Action du Smart Contract :

— Si le service critique est validé, le smart contract active définitivement la réservation.

— Azure ayant refusé la demande, seul le service AWS est activé.

4.4.5.2 Cas 2 : Le Client Ne Satisfait Pas les Conditions

Le client ne complète pas les étapes requises dans le délai imparti (exemple : 24 heures).

Conséquence :

— AWS publie une transaction sur la blockchain indiquant que **les conditions n'ont pas été satisfaites.**

— Le smart contract annule la réservation partielle et considère la requête comme échouée.

Action du Smart Contract :

— La réservation des VM chez AWS est définitivement refusée.

— Le service Azure ayant déjà été refusé, la totalité de la demande est rejetée.

4.4.5.3 Synthèse des Scénarios

— **Si le client satisfait les conditions :** Accès final accordé uniquement au service critique (AWS), Azure reste refusé.

— **Si le client échoue :** Demande complètement annulée, aucun service n'est activé.

Événement émis	**Détails**
`DecisionFinale (cas 1)`	`emit DecisionFinale("REQ-123", "PARTIAL_ACCESS");`
`DecisionFinale(cas 2)`	`emit DecisionFinale("REQ-123", "ACCESS_DENIED");`

Table 4.5 – Événements émis par le smart contract selon le scénario

La gestion dynamique de l'accès conditionnel est entièrement tracée sur la blockchain, permettant une transparence totale du processus.

3. VPC : Virtual Private Cloud

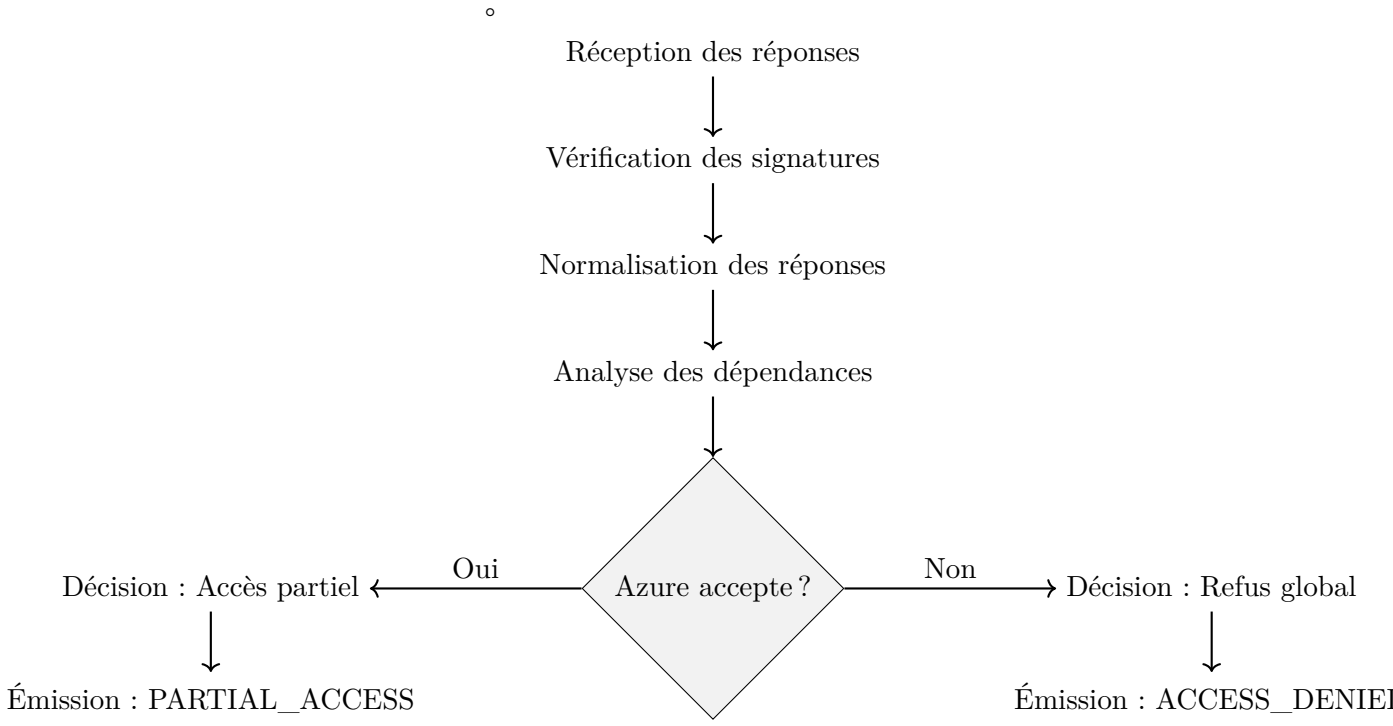

FIGURE 4.2 – Actions du Smart Contract : Refus global et Accès partiel

4.4.6 Preparation du nouveau bloc sur la blockchain

Dans le cadre de notre scénario, le validateur observe le mempool et regroupe les transactions liées à la requête globale identifiée par `request_id_parent = "REQ-123"`. Il calcule ensuite l'arbre de Merkle correspondant et génère un nouveau bloc.

Le bloc généré contient les transactions suivantes :

— La transaction initiale émise par le dAPI.

— La transaction de réponse d'AWS (Accès partiel ou conditionnel selon les cas).

— La transaction de réponse d'Azure (Refusée dans notre scénario).

— L'événement `DecisionFinale` émis par le smart contract.

Le tableau suivant résume la structure du bloc généré dans ce contexte.

Table 4.6 – Bloc blockchain correspondant au scénario simulé

Bloc N	
Section	**Contenu**
En-tête du bloc	**Identifiant du bloc :** 0xB123 **Hachage précédent :** 0xA12F **Timestamp :** 2025-06-25T14 :30 :00Z **Identifiant validateur :** 0xV789 **Merkle Root :** 0xM45678 (calculée à partir des transactions ci-dessous)
Corps du bloc	— **TX1** : Transaction dAPI (requête globale : REQ-123) — **TX2** : Réponse AWS (Accès partiel ou conditionnel) — **TX3** : Réponse Azure (Refusée) — **Log** : Événement `DecisionFinale` (ACCESS_DENIED ou PARTIAL_ACCESS)
Métadonnées	— **Request ID :** REQ-123 — **Utilisateur :** U-456 — **Fournisseurs impliqués :** AWS, Azure — **Nombre de sous-requêtes :** 2
Contrôle de cohérence	— Vérification d'unicité des identifiants de requête — Élimination des doublons de sous-requêtes — Validation des signatures des transactions

4.4.7 Validation des Blocs par les Nœuds

Les transactions soumises par les fournisseurs et les actions du smart contract sont regroupées par les validateurs du réseau et intégrées dans un bloc. La validation de ce bloc rend la décision définitive et immuable.

4.4.8 Mécanisme de Notification Décentralisée

Une fois la décision validée sur la blockchain, le smart contract déclenche une notification via le dAPI. Le client reçoit une réponse structurée contenant l'état final de sa demande (accepté, refusé ou expiré), garantissant la cohérence entre la blockchain et les systèmes externes.

Cas 1 : Le Client Satisfait les Conditions

Votre accès aux machines virtuelles chez AWS est définitivement activé.
Le service Azure reste non disponible.

Cas 2 : Le Client Ne Satisfait Pas les Conditions

Votre demande de réservation multi-cloud a été refusée.
Motif : Les conditions imposées par AWS n'ont pas été satisfaites dans les délais.

Conclusion

Ce scénario a permis d'illustrer de manière détaillée le fonctionnement d'un système de contrôle d'accès décentralisé basé sur la blockchain dans un environnement multi-cloud. À travers l'exemple d'une demande de ressources réparties entre deux fournisseurs, nous avons démontré comment les services critiques et dépendants interagissent avec la DAPI et le smart contract pour garantir la cohérence et la sécurité des accès.

Le processus de vérification locale par chaque fournisseur, la gestion des accès conditionnels, ainsi que la traçabilité des décisions via la blockchain assurent une transparence et une immutabilité des échanges. L'intégration de la logique de dépendance entre les services a également mis en évidence la capacité du smart contract à orchestrer efficacement des décisions multi-acteurs dans un contexte distribué.

Cette proposition pose ainsi les bases d'un système fiable, automatisé et vérifiable, capable de renforcer la confiance dans la gestion des ressources multi-cloud. Cependant, par manque de temps, nous n'avons pas pu réaliser de simulation complète ni d'implémentation pratique, ce qui nous a limité à la présentation d'un scénario théorique.

Conclusion générale

Au terme de ce mémoire, nous avons conçu, modélisé et étudié une architecture décentralisée de contrôle d'accès multi-cloud s'appuyant sur une DAPI (Decentralized API Gateway) et des smart contracts afin de répondre aux enjeux actuels de gouvernance dans des environnements hétérogènes. Cette solution offre une approche innovante en remplaçant les mécanismes centralisés classiques par une logique entièrement distribuée, traçable et infalsifiable grâce aux propriétés inhérentes de la blockchain. Ainsi, chaque décision d'accès est horodatée, signée numériquement et consignée dans un registre immuable, ce qui facilite les audits et le suivi de la conformité dans le temps.

L'architecture proposée DAccess-ChainMC (Decentralized Access Chain for Multi-Cloud) exploite les fonctionnalités d'automatisation, d'intégrité et de transparence des smart contracts, ce qui garantit que le processus d'évaluation des requêtes d'accès aux ressources cloud est à la fois fiable, contrôlé et conforme aux politiques de sécurité définies. De plus, le recours aux mécanismes de négociation intégrés permet de traiter efficacement les cas d'incertitude (réponses contradictoires, services conditionnels, dépendances manquantes), en sollicitant une intervention dynamique pour affiner la décision finale.

Le scénario illustré au fil du mémoire a permis de démontrer le caractère réaliste et applicable du modèle dans un contexte multi-fournisseurs. Concrètement, les résultats montrent qu'il est possible :

— d'orchestrer efficacement des sous-requêtes en parallèle entre plusieurs clouds,

— d'agréger de façon cohérente les réponses retournées,

— d'arbitrer les conflits entre les décisions partielles grâce au smart contract, sans dépendre d'une autorité centrale.

Cependant, ce travail reste une proposition conceptuelle qui ouvre de nombreuses pistes d'évolution.

Perspectives

Une première perspective consiste à développer un prototype fonctionnel basé sur des solutions existantes telles qu'API3 (Airnode) ou Chainlink Functions pour le DAPI [4], et Hyperledger Fabric, Polygon ou Avalanche pour le registre blockchain. Cette expérimentation permettrait d'évaluer la performance réelle du modèle en termes de latence, de coût transactionnel et de scalabilité dans un environnement multi-cloud concret.

Une autre piste concerne l'intégration de mécanismes avancés de gestion des identités décentralisées (SSI) [5] et de techniques de confidentialité telles que les preuves à divulgation nulle de connaissance (ZKP) [6]. De telles extensions permettraient de renforcer la protection de la vie privée des utilisateurs tout en respectant les contraintes réglementaires liées à la protection des données.

Par ailleurs, une validation expérimentale incluant des scénarios d'attaque (usurpation, collusion entre fournisseurs, déni de service) et une comparaison avec des modèles centralisés traditionnels offrirait une évaluation complète de la robustesse et de la valeur ajoutée de l'approche.

Enfin, les applications industrielles de cette architecture sont multiples : gouvernance des accès dans les administrations publiques, audit des accès aux données sensibles dans le secteur bancaire et de la santé, ou encore gestion sécurisée des autorisations dans l'Internet des objets (IoT) connecté au cloud.

Ces perspectives ouvrent également la voie à un approfondissement académique, notamment dans le cadre d'un projet doctoral, afin d'explorer plus en détail les synergies entre blockchain, multi-cloud et sécurité des systèmes distribués.

4. DAPI : Decentralized API Gateway
5. SSI : Self-Sovereign Identity
6. ZKP : Zero-Knowledge Proof

Bibliographie

[1] Y. Zhang et al. "A Blockchain-based Access Control Model for Cloud Data". In : *IEEE Transactions on Cloud Computing* 6.2 (2018), p. 472-483.

[2] Cisco Networking Academy. *Défense du réseau - Cours en ligne*. Disponible pour les utilisateurs inscrits. 2023. URL : https://www.netacad.com/ (visité le 17/12/2024).

[3] Rézau. *Les avantages du cloud privé VMware face au cloud public*. Consulté le 17 septembre 2025. 2025. URL : https://www.rezau.com/blogue/les-avantages-du-cloud-prive-vmware-face-au-cloud-public.

[4] K. Jamsa. *Cloud Computing : Theory and Practice*. 3e éd. Hybrid and Multi-Cloud Strategies. Jones & Bartlett Learning, 2021. Chap. 12.

[5] Cloudflare. *What is multi-cloud ?* Consulté le 17 septembre 2025. n.d. URL : https://www.cloudflare.com/learning/cloud/what-is-multicloud/.

[6] Jonathan Schabowsky. "The Benefits of Microservices Choreography vs Orchestration". In : *Solace Blog* (26 nov. 2019). URL : https://solace.com/blog/microservices-choreography-vs-orchestration/ (visité le 17/12/2024).

[7] Microsoft Docs. *Choreography pattern*. 2023. URL : https://learn.microsoft.com/fr-fr/azure/architecture/patterns/choreography (visité le 17/12/2024).

[8] Bitcot. *What is an API ? What are the types of APIs and their Uses ?* 24 juin 2021. URL : https://www.bitcot.com/types-of-api/ (visité le 07/01/2025).

[9] Amazon Web Services. *Qu'est-ce que la blockchain ?* 2023. URL : https://aws.amazon.com/fr/what-is/blockchain/?aws-products-all.sort-by=item.additionalFields.productNameLowercase&aws-products-all.sort-order=asc (visité le 17/12/2024).

[10] Pngtree. *Blockchain Vector Graphic*. Image consultée le 11 juin 2025. 2025. URL : https://pngtree.com/freepng/blockchain_8875358.html.

[11] Arshdeep Bahga et Vijay K. Madisetti. "Blockchain Platform for Industrial Internet of Things". In : *Journal of Software Engineering and Applications* 9 (2016), p. 533-546. URL : https://www.researchgate.net/publication/309543764_Blockchain_Platform_for_Industrial_Internet_of_Things (visité le 12/01/2025).

[12] Archipels. *Quels sont les différents algorithmes de consensus ?* 2023. URL : https://www.archipels.io/faq/quels-sont-les-differents-algorithmes-de-consensus (visité le 22/01/2025).

[13] Oussama Abderraouf Ayadi. "Chapitre III : État de l'art de la blockchain". Mém. de mast. Université Constantine Abdelhamid Mehri, 2019.

[14] M. Bedawala. *What are Consensus Mechanisms ?* Visa. 17 jan. 2023. URL : https://usa.visa.com/solutions/crypto/consensus-mechanisms.html (visité le 22/01/2025).

[15] CoinAcademy. *Les différents algorithmes de consensus sur la blockchain*. 2023. URL : https://coinacademy.fr/academie/differents-algorithmes-consensus-blockchain/ (visité le 22/01/2025).

[16] O. Depierre et al. *Consensus - Lexique de la blockchain*. CDBF. 2022. URL : https://cdbf.ch/lexique/consensus/ (visité le 10/01/2025).

[17] B. Li et al. "Incentive-based demand response program with self-reported baseline supported by blockchain technology". In : *IET Smart Grid* 6.2 (2023), p. 205-218. URL : https://www.researchgate.net/publication/367052235_Incentive-based_demand_response_program_with_self-reported_baseline_supported_by_blockchain_technology (visité le 30/01/2025).

[18] Satoshi Nakamoto. "Bitcoin : Un système de paiement électronique peer-to-peer". In : (2008). URL : https://bitcoin.org/bitcoin.pdf (visité le 02/01/2025).

[19] Blockchains Expert. *L'arbre de Merkle : Colonne vertébrale de blockchain*. 2023. URL : https://www.blockchains-expert.com/larbre-de-merkle-colonne-vertebrale-de-blockchain/ (visité le 02/01/2025).

[20] Sectigo. *Que sont les signatures numériques et comment fonctionnent-elles ?* 2020. URL : https://www.sectigo.com/fr/ressources/comment-fonctionnent-les-signatures-numeriques (visité le 20/01/2025).

[21] Alain Bonneaud. *Blockchain : Fonctionnement du minage en 7 étapes*. 2 mai 2019. URL : https://www.ab-consulting.fr/blog/blockchain/minage-7-etapes (visité le 04/01/2025).

[22] Bitpanda Academy. *Comment fonctionne une blockchain ?* 2025. URL : https://www.bitpanda.com/academy/fr/lecons/comment-fonctionne-une-blockchain/ (visité le 14/01/2025).

[23] NIST. "Guidelines on Security and Privacy in Public Cloud Computing". In : (2011). SP 800-144.

[24] *ISO/IEC 27001 :2013 - Information security management*. International Organization for Standardization, 2013.

[25] NIST. *Blockchain for Access Control Systems*. IR 8403. 2022.

[26] Pratap Kumar Behera et Pabitra Mohan Khilar. "A Novel Trust Based Access Control Model for Cloud Environment". In : *Proceedings of the International Conference on Signal, Networks, Computing, and Systems*. T. 395. Lecture Notes in Electrical Engineering. New Delhi : Springer India, 2017, p. 285-295. DOI : 10.1007/978-81-322-3592-7_29.

[27] Ravi S Sandhu et al. "Role-based access control models". In : *Computer* 29.2 (1996), p. 38-47.

[28] Wallarm. *Qu'est-ce que le contrôle d'accès basé sur les rôles (RBAC) ?* En ligne. Wallarm, Inc. 2023. URL : https://www.wallarm.com/what/what-exactly-is-role-based-access-control-rbac (visité le 15/01/2025).

[29] Vincent Hu et al. *Guide to attribute based access control (ABAC) definition and considerations*. Rapp. tech. NIST SP 800-162. National Institute of Standards et Technology, 2014.

[30] Auteur du site. *Mesures de sécurité mises en œuvre dans le cloud*. `https://www.exemple.com/cloud-securite-mesures`. Site consulté le 22/02/2025. 2023. url : `https://www.exemple.com/cloud-securite-mesures`.

[31] Vincent C. Hu. *Blockchain for Access Control Systems*. Springer, 2020.

[32] Swatisipra Das et al. "Leveraging Towards Access Control, Identity Management, and Data Integrity Verification Mechanisms in Blockchain-Assisted Cloud Environments : A Comparative Study". In : *Journal of Cybersecurity and Privacy* 4.4 (2024), p. 1018-1043. doi : `10.3390/jcp4040047`. url : `https://www.mdpi.com/2624-800X/4/4/47`.

[33] J. Liu et al. "A Blockchain-based Message Authentication Code for Secure Data Sharing". In : *IEEE Internet of Things Journal* 7.12 (2020), p. 12163-12175.

[34] Y. Zhang et al. "A Cloud-Blockchain Fusion Framework for Secure Data Sharing". In : *IEEE Transactions on Cloud Computing* 7.3 (2019), p. 768-781.

[35] X. Li et al. "A Blockchain-based Attribute-Based Access Control Scheme for Cloud Storage". In : *IEEE Transactions on Services Computing* 14.4 (2021), p. 985-998.

[36] Md. Sadek Ferdous, Andrea Margheri et Federica Paci. "Decentralised Runtime Monitoring for Access Control Systems in Cloud Federations". In : *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*. Juin 2017, p. 2453-2459. doi : `10.1109/ICDCS.2017.178`.

[37] CloudBolt Software. *CloudBolt - Hybrid Cloud Management Platform*. Consulté le 25 février 2025. 2024. url : `https://www.cloudbolt.io/`.

[38] CoinAcademy. *API3 : Comprendre les dAPI et l'agrégation décentralisée*. Consulté le 13 mars 2025. 2023. url : `https://coinacademy.fr/api3-api3-fondamental/`.

.1 Architecture technique

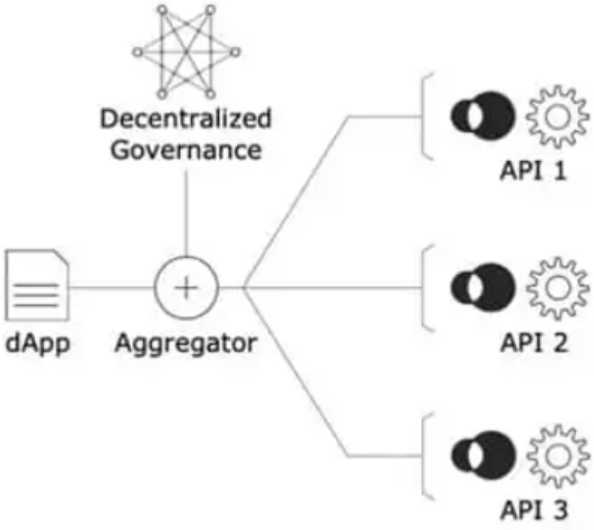

FIGURE 3 – Architecture d'un dAPI avec agrégateur [38]

.2 Détails sur la gestion des validateurs dans une blockchain PoA

.2.1 Smart Contract de gouvernance des validateurs

La gestion des validateurs est assurée par un **smart contract de gouvernance** déployé sur la blockchain. Il permet :

- **Ajout de nouveaux validateurs :**
  - Sur demande, après vérification d'identité et d'infrastructure,
  - Via un vote ou une approbation des validateurs en place.
- **Révocation de validateurs :**
  - En cas de comportement malveillant ou d'inactivité prolongée,
  - En cas de violation des règles du protocole.
- **Vote/quorum pour la mise à jour du registre :**
  - Selon des critères tels qu'une majorité simple ou pondérée.
- **Gouvernance automatisée :**
  - Par exemple, un administrateur élu peut déclencher les procédures après consensus.

.2.2 Rôle des validateurs dans PoA

Les validateurs sont des acteurs identifiés qui :

- Valident les transactions (signature, cohérence, syntaxe).

— Créent les blocs selon un ordre prédéfini.

— Exécutent les smart contracts lors de l'inclusion des transactions.

— Assurent le consensus en respectant les règles établies.

— Sont responsables publiquement, renforçant ainsi la sécurité et la confiance.

.2.3 Sélection du producteur de bloc

Le mécanisme utilisé repose sur des **slots temporels** :

— Le temps est divisé en intervalles fixes (slots).

— Chaque slot est attribué à un validateur spécifique.

— En cas d'absence, un validateur de secours prend la relève.

.2.4 Sélection dynamique des validateurs

La validation ne repose pas sur l'ensemble des validateurs, mais sur un **sous-ensemble dynamique** sélectionné pour chaque bloc via un smart contract.

Ce dernier définit :

— Les critères d'éligibilité (réputation, disponibilité, activité).

— La durée du mandat de validation.

— Le mode de désignation (vote, aléatoire contrôlé, etc.).

Avantages :

— Répartition équitable de la charge.

— Meilleure résilience en cas de défaillance.

— Réduction des collusions ou surcharges.

— Flexibilité pour suspendre ou remplacer des validateurs sans perturber le consensus global.

.3 Mécanisme de réception et déclenchement automatique

Structures de données utilisées

— `mapping(string => uint8) compteur_reponses :`
  — Structure de données permettant de suivre, pour chaque requête, le nombre de réponses effectivement reçues.
  — **Clé :** identifiant unique de la requête.
  — **Valeur :** compteur entier (non signé, 8 bits).

— `mapping(string => uint8) total_attendu :`
  — Stocke le nombre total de réponses attendues pour une requête donnée.
  — Ce nombre est défini lors de la création initiale de la requête.

Fonction `enregistrerReponse`

1. **Vérification de la légitimité :**
   — L'appelant (`msg.sender`) doit être un fournisseur cloud autorisé.
   — Cela empêche les tentatives d'enregistrement frauduleuses.
2. **Incrémentation sécurisée :**
   — Le compteur associé à la requête est incrémenté dès réception d'une réponse valide.
   — L'opération est atomique et fiable dans un environnement blockchain.
3. **Déclenchement conditionnel :**
   — Le contrat vérifie si toutes les réponses attendues ont été reçues.
   — Si oui, il appelle automatiquement la fonction `executerDecisionFinale`.
   — Cette dernière assure la normalisation des réponses, le contrôle des règles d'accès, et la prise de décision finale.

Ce mécanisme permet de garantir une logique entièrement décentralisée, dans laquelle le déclenchement du traitement n'a pas besoin d'intervention humaine ni de coordination externe. Tout est défini de manière déterministe et automatisée dans la blockchain.

Abstract

The present work is part of a graduation project carried out in order to obtain the Master's degree in Computer Science at Université de Abderrahmane MIRA -BEJAIA- .

The objective of this thesis is to design and implement a decentralized multi-cloud access control architecture based on blockchain technology. This architecture ensures secure, transparent, and tamper-proof management of access permissions across multiple cloud service providers (CSPs).

The thesis involves the simulation of a scenario where a client requests resources from different CSPs simultaneously. The access control process is automated using smart contracts deployed on the blockchain, which collect, verify, and aggregate responses from the involved providers. A decentralized API (dAPI) facilitates communication between the client, providers, and smart contracts.

The designed system offers advantages in terms of transparency, resilience, and distributed trust, reducing the risks associated with centralized architectures.

Keywords— Blockchain, Multi-Cloud, Access Control, Smart Contract, dAPI, Cloud Security

Résumé

Ce travail s'inscrit dans le cadre du projet de fin d'études en vue de l'obtention du diplôme de master Informatique à l'Université de Abderrahmane MIRA -BEJAIA- .

L'objectif de ce mémoire est de concevoir et de mettre en œuvre une architecture décentralisée de contrôle d'accès multi-cloud basée sur la technologie blockchain. Cette architecture vise à assurer une gestion sécurisée, transparente et immuable des autorisations d'accès auprès de plusieurs fournisseurs de services cloud (CSPs).

Le mémoire s'appuie sur la simulation d'un scénario où un client sollicite simultanément des ressources auprès de différents fournisseurs. Le contrôle d'accès est automatisé via des smart contracts déployés sur la blockchain, chargés de collecter, vérifier et agréger les réponses des fournisseurs concernés. Une API décentralisée (dAPI) facilite la communication entre le client, les fournisseurs et les smart contracts.

L'architecture proposée présente des avantages en matière de transparence, de résilience et de confiance distribuée, tout en limitant les vulnérabilités associées aux systèmes centralisés.

Mots clés— Blockchain, Multi-Cloud, Contrôle d'accès, Smart Contract, dAPI, Sécurité Cloud