

Un Mini Serveur dédié au “Module Réseau”

**Fabrice Bouvier, Frédéric Eynard, Frédéric
Gutierrez et Renaud Waldura**

**Une étude de cas “Réseau Couches
Hautes” réalisée en 3^{ème} année de
l’École Supérieure en Sciences
Informatiques**

Cette étude présente les spécifications d’une architecture logicielle destinée à faciliter la communication entre les étudiants et les enseignants du module “Réseau” de l’ESSI. Cette architecture trouve son application dans un mini-serveur dédié, le *MSR* (Mini Serveur Réseau).

Le but de ce serveur est de fournir un service de documentation distribué accessible de manière sûre et facile aussi bien par les étudiants que par les enseignants :

- les objets du MSR sont nommés suivant le modèle X.500 ce qui facilite leur manipulation par des utilisateurs humains,
- les accès aux documents sont certifiés et garantis uniques,
- les transferts de données sont sécurisés.

Nous avons choisi de décomposer l’architecture ce serveur en 4 sous-systèmes distincts, reprenant chacun une des fonctionnalités principales du MSR :

1. un serveur de documents,
2. un serveur de sécurité,
3. un serveur de noms,
4. un serveur d’accessibilité.

Nous donnerons tout d’abord une vue d’ensemble de l’architecture du MSR, pour détailler après ses différents composants. Les spécifications du protocole Client-MSR seront données, ainsi qu’un exemple de scénario.

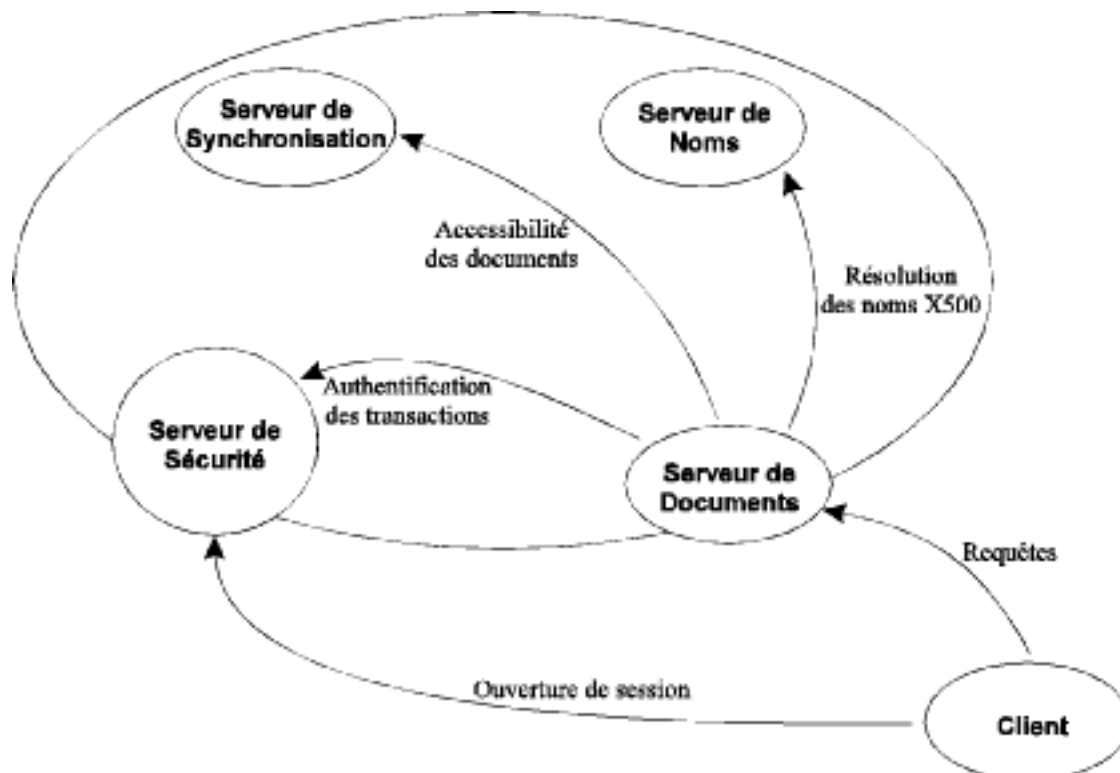
1.0	Vue globale de l'architecture	3
1.1	Composants du MSR	3
1.1.1	Le Serveur de Documents	3
1.1.2	Le Serveur de Sécurité	3
1.1.3	Le Serveur de Noms	4
1.1.4	Le Serveur d'Accessibilité	4
1.2	Fonctionnalités du MSR	4
1.2.1	La méthode lire_document()	4
1.2.2	La méthode écrire_document()	4
1.2.3	La méthode change_permissions()	4
2.0	Le Serveur de Sécurité	4
2.1	Le système de chiffrement à clef publique RSA	5
2.1.1	Pourquoi un système de chiffrement à clef publique ?	5
2.1.2	Pourquoi RSA ?	5
2.1.3	L'algorithme RSA	5
2.1.4	Considérations d'efficacité	5
2.2	Échanges Client-MSR	6
2.2.1	Initialisation du système—Ouverture de session	6
2.2.2	Services du Serveur de Sécurité	7
2.3	Échanges MSR-MSR	7
2.4	Contraintes sur le Client	8
3.0	Le Serveur de Noms	8
4.0	Le Serveur d'Accessibilité	9
4.1	Permissions d'accès	9
4.2	Synchronisation	9
4.2.1	Modification d'un document	9
4.2.2	Lecture d'un document	9
4.2.3	Tolérance aux pannes	10
5.0	Le Serveur de Documents	10
5.1	Les méthodes du Serveur de Documents	10
5.1.1	lire_document()	10
5.1.2	écrire_document()	12
5.1.3	change_permissions()	13
6.0	Conclusion	14
6.1	Remarques d'implémentation	14

1.0 Vue globale de l'architecture

Les différents composants du MSR définis en introduction interagissent de la manière explicitée par le schéma suivant.

Nous préciserons ensuite le rôle de chacun.

FIGURE 1. L'architecture du MSR



1.1 Composants du MSR

1.1.1 Le Serveur de Documents

Le serveur de documents maintient une base de données des documents gérés par le MSR.

Il est l'entité de dialogue privilégiée des clients du MSR ; c'est par lui que transiteront toutes les requêtes faites au MSR, qu'il éclatera vers les autres serveurs.

1.1.2 Le Serveur de Sécurité

Son rôle est d'assurer la confidentialité et l'authentification des transactions entre le MSR et ses clients. Nous avons choisi de définir un schéma de sécurité basé sur un système de chiffrement à clef publique, RSA.

Le Serveur de Sécurité est en relation avec les clients uniquement lors de l'ouverture d'une session de dialogue avec le MSR.

1.1.3 Le Serveur de Noms

Il permet de faire la relation entre les noms X.500 des objets manipulés par le système (documents, personnes), et leurs identificateurs internes au MSR.

Nos clients n'accèdent jamais directement au Serveur de Noms ; toutes les requêtes lui sont adressées via le Serveur de Documents du MSR.

1.1.4 Le Serveur d'Accessibilité

Ce serveur contrôle l'accessibilité d'un document. Il gère les permissions associées à chaque document, et les problèmes de synchronisation entre accès concurrents à un même objet.

Ainsi que pour le Serveurs de Noms, ce serveur est transparent aux clients du MSR.

Nous détaillerons de manière plus précise chaque serveur dans la partie qui lui est consacrée.

1.2 Fonctionnalités du MSR

Nous avons choisi de simplifier les fonctionnalités / fertes par le MSR en trois primitives :

- `lire_document()`
- `écrire_document()`
- `change_permissions()`

Il est à noter que cette simplification n'occulte aucun des problèmes fondamentaux liés au fonctionnement de l'architecture ; elle ne l'avons choisi que dans le but d'en raccourcir les spécifications.

1.2.1 La méthode `lire_document()`

Cette méthode permet à un client d'avoir accès à un document du MSR.

Prototype. `lire_document(nom_client, nom_document)`

1.2.2 La méthode `écrire_document()`

Elle autorise un client à modifier ou ajouter un document dans la base des documents gérée par le MSR.

Prototype. `écrire_document(nom_client, nom_document, données_document)`

1.2.3 La méthode `change_permissions()`

Elle est utilisée par un client pour spécifier les droits d'accès à ses documents.

Prototype. `change_permissions(nom_client, nom_document, type_permission, liste_personnes)`

2.0 Le Serveur de Sécurité

Nous avons choisi de sécuriser l'architecture du MSR en intégrant dans notre protocole un système de cryptologie à clef publique, RSA.

2.1 Le système de chiffrement à clef publique RSA

2.1.1 Pourquoi un système de chiffrement à clef publique ?

Contrairement aux systèmes de cryptage ordinaires “à clef secrète”, les systèmes dits “à clef publique” séparent la clef utilisée par l’algorithme en deux parties, une *clef publique* qui est comme son nom l’indique librement diffusée, et une *clef privée* qui est gardée secrète.

Cette séparation augmente la sécurité globale du système, et permet également le découplage de deux services précédemment liés qui répondent à des besoins bien différents :

- le *chiffrement* ou *encryptage* des données, pour garantir la confidentialité d’un message,
- l’*authentification* des données, afin de créer une signature digitale authentifiant l’auteur d’un document.

2.1.2 Pourquoi RSA ?

RSA est sans doute aujourd’hui l’algorithme de chiffrement à clef publique le plus connu et le plus prometteur. Les atouts de RSA qui nous ont incité à l’utiliser dans le cadre de cette application sont les suivants :

- sa sûreté : le cryptage RSA repose sur une conjecture mathématique simple mais qui tient depuis deux millénaires, la difficulté de factorisation d’un produit de nombres premiers,
- son adaptabilité à une application précise : la force du cryptage, et donc la puissance de calcul nécessaire à l’utilisation de RSA, dépend de la longueur des clefs, paramètre laissé à la discrétion de l’utilisateur.

2.1.3 L’algorithme RSA

La première étape de l’algorithme de génération des clefs consiste à choisir un nombre entier $n = pq$, p et q étant deux “grands” nombres premiers tenus secrets. La clef publique est un couple $\langle e, n \rangle$ tel que $e = (p - 1)(q - 1) \bmod n$. La clef privée est un couple $\langle d, n \rangle$ tel que $d = 1 / e \bmod (p - 1)(q - 1)$.

L’encryptage d’un mot m consiste à générer un chiffre c tel que $c = m^e \bmod n$, nécessitant donc une clef publique, et la signature s d’un mot m est calculée par $s = m^d \bmod n$, qui a ainsi besoin d’une clef privée.

L’utilisation de l’algorithme RSA est résumée dans le tableau suivant.

TABLE 1.

Utilisation des clefs RSA

	Émetteur	Destinataire
Cryptage	clef publique du destinataire	clef privée
Signature	clef privée	clef publique de l’émetteur

2.1.4 Considérations d’efficacité

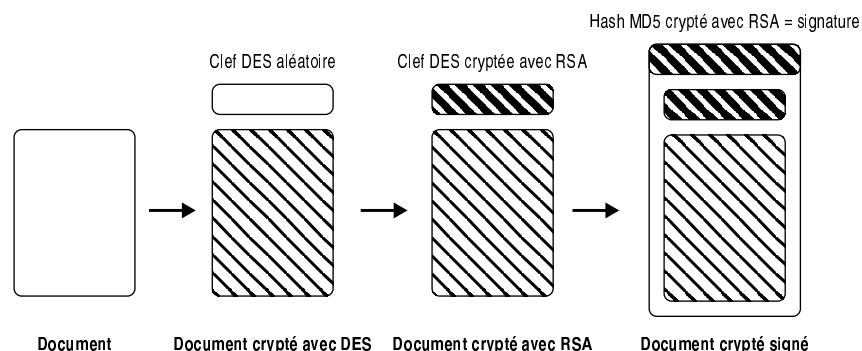
Le cryptage RSA est une opération extrêmement coûteuse en temps de calcul, et inutilisable en pratique pour des gros volumes de données. Aussi on associe souvent système à clef publique et système à clef secrète pour le chiffrement des documents

: le cryptage RSA permet de créer un canal sûr pour la transmission de la clef secrète aléatoire utilisée pour crypter les données du document elles-même.

Nous avons choisi dans le cadre de cette application le standard DES comme système de chiffrement à clef secrète.

FIGURE 2.

Cryptage et signature d'un document avec RSA



2.2 Échanges Client–MSR

Voici comment nous avons appliqué le modèle RSA à l'architecture du MSR dans le but de sécuriser toutes les transactions client–serveur.

2.2.1 Initialisation du système—Ouverture de session

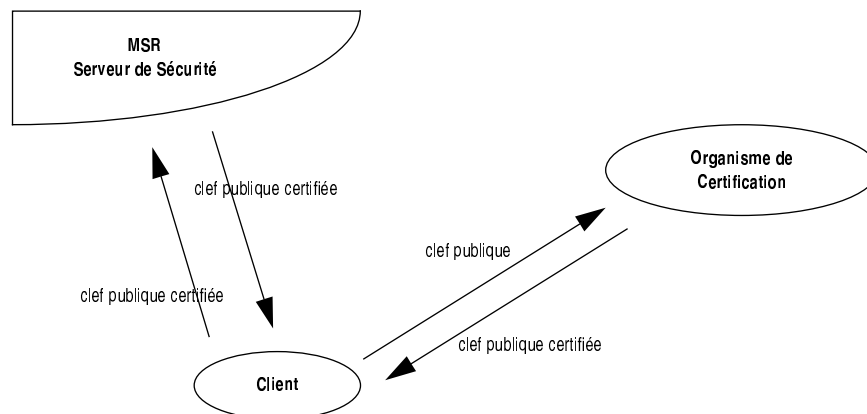
Le Serveur de Sécurité est le premier acteur du MSR à interagir avec le client lors d'un début de "session". En fait cette phase d'ouverture de session consiste simplement en l'échange des clefs publiques de chaque intervenant.

Il est possible d'authentifier les clefs publiques passées dans ce premier échange en les faisant *certifier* par un Organisme de Certification extérieur—une partie tierce, à laquelle le MSR et son client font confiance, et dont ils détiennent chacun la clef publique. Dans le cadre d'une application dédiée au module "réseau" de l'ESSI, cet organisme de certification serait sans doute l'administration système de l'École, qui aurait ainsi la charge de certifier (en les signant tout simplement) les clefs publiques des gens de l'ESSI. Cette certification est datée et permet ainsi aux entités de l'architecture une vérification de la validité temporelle des clefs.

Le Serveur de Sécurité du MSR possède pour sa part une paire de clefs publique/privée qui est générée lors de l'installation du logiciel. Il gère en outre un *cache* des clefs publiques des intervenants du module, qui évite des échanges redondants de clefs (objets souvent lourds) lors de l'ouverture d'une session. Cette phase est alors réduite à une simple vérification de présence dans le cache de la clef.

FIGURE 3.

Ouverture de session MSR



Ainsi, avant toute transaction, le MSR est initialisé avec les objets suivants :

- sa clef privée, sa clef publique signée par l’Organisme de Certification,
- la clef publique de l’Organisme de Certification afin de vérifier la validité des clefs publiques des clients,
- un certain nombre (éventuellement nul) de clefs publiques appartenant à des clients ; c’est le cache des clefs dans un but d’optimisation.

Suite à une ouverture de session, le Client détient pour sa part :

- sa clef privée, sa clef publique signée par l’Organisme de Certification,
- la clef publique de l’Organisme de Certification afin de vérifier la validité de la clef publique du MSR,
- la clef publique du MSR.

Cette étape d’ouverture de “session” réussie, des échanges de données sécurisés peuvent prendre place entre le Client et le MSR.

2.2.2 Services du Serveur de Sécurité

Le Serveur de Sécurité, détenteur de toutes les clefs utilisées par le MSR, propose les services suivants aux sous-systèmes composant celui-ci :

- un service de cryptage/décryptage grâce aux méthodes `crypte()` et `décrypte()`
- un service de validation de document fonctionnant grâce à une vérification de signature grâce à la méthode `vérifie_signature()`

2.3 Échanges MSR–MSR

Nous avons considéré jusqu’à présent dans cette étude que l’environnement d’exécution du MSR était sûr : les transactions internes aux sous-systèmes (entre le serveur de document et le serveur de sécurité par exemple) ne pouvaient pas être compromis. Il n’y a alors pas lieu de sécuriser ces transactions.

Dans le cas d’une architecture complètement distribuée, où les différents serveurs composant le MSR seraient sur des machines distinctes, il deviendrait utile de

sécuriser totalement l'architecture en signant et/ou cryptant même les échanges internes au MSR : chaque serveur disposerait alors de sa paire de clefs publique/privée et du code d'utilisation correspondant.

Nous ne mentionnons cette extension que par souci de complétude : elle alourdit simplement l'architecture, sans pour autant la compliquer conceptuellement.

2.4 Contraintes sur le Client

Adopter une architecture sécurisée demande bien sûr que le Client offre plus de capacités : celui-ci doit disposer des même fonctionnalités de vérification de signature et de cryptage/décryptage que le serveur.

En fait, le code RSA correspondant est disponible sous forme de bibliothèque et ne pose pas de problèmes d'intégration dans une architecture telle que la nôtre.

3.0 Le Serveur de Noms

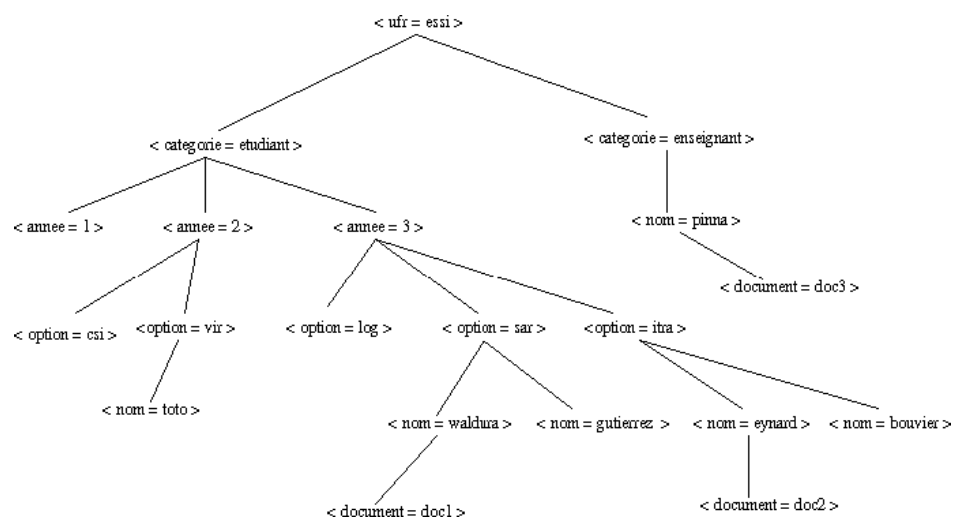
Le serveur gère les noms de personnes ainsi que les noms de documents au format X.500. Il associe à chaque objet (documents et utilisateurs) un nom interne à notre MSR. Cet objet a une entrée dans la base avec un nom unique, il est constitué d'une liste d'attributs.

Chaque attribut contient une partie de l'information de l'objet et un objet contient des informations sur sa classe. Une classe d'objets X.500 regroupe un ensemble d'objets. Les objets ont des caractéristiques communes et chaque information est une instance de classe d'objets normalisés.

Cet arbre est représenté la hiérarchie des classes X.500. Il est géré exclusivement par l'administrateur système et mis à jour au début de chaque année scolaire.

FIGURE 4.

Arbre de nommage



A chaque création d'un document, il est rajouté une feuille dans l'arbre de nommage.

4.0 Le Serveur d'Accessibilité

4.1 Permissions d'accès

Le Serveur d'Accessibilité a charge de vérifier si un client donné peut obtenir les informations qu'il demande. Pour cela, il dispose, pour chaque document, d'une liste de personnes avec leurs droits d'accès.

TABLE 2.

Droits d'accès par document

	Lecture	Écriture
Document A	utilisateur 1, utilisateur 2, ...	utilisateur 3, utilisateur 4 ...
Document B	utilisateur 3, utilisateur 6 ...	utilisateur 2, utilisateur 9 ...

Nous avons choisi de ne pas définir de notion de *groupe* d'utilisateurs qui permettrait d'affecter des permissions à plusieurs personnes : il faut spécifier explicitement une liste de tous les utilisateurs auxquels on donne accès à ce document.

Les accès vérifiés, deux cas de figures se présentent : soit le client veut modifier le document, soit le client veut le lire ; nous étudions ces deux situations dans les paragraphes suivants.

4.2 Synchronisation

Les informations sont manipulées par un grand nombre d'entités. Le serveur doit garantir la cohérence des données et leur confidentialité. En effet dans un système réparti, il est envisageable que plusieurs clients veuillent accéder à la même ressource en même temps. Il faut alors mettre en place un système assurant une synchronisation des accès faits aux données.

4.2.1 Modification d'un document

Un verrou est associé au fichier que l'on veut modifier : le document n'est plus accessible tant que l'utilisateur n'a pas terminé son opération. Les clients, désirant obtenir la même information, sont placés dans une file d'attente. La gestion de celle-ci peut s'effectuer de plusieurs manières :

1. on place les clients par leur ordre d'arrivée. Cette méthode sera alors simple d'implémentation,
2. les requêtes des clients sont datées. Cela suppose que les acteurs entretiennent une horloge répartie strictement croissante. La file est alors triée en fonction des dates croissantes. L'initialisation de l'horloge sera donc importante.

Une fois l'écriture effectuée, on déverrouille ce fichier. Le serveur attribue ensuite le document au premier de la file (s'il existe).

4.2.2 Lecture d'un document

Si aucun verrou n'est présent sur l'information désirée, il obtient une copie de celle-ci. Sinon le client est mis en attente. Il peut cependant, par l'envoi d'une com-

mande, se retirer de la file. Il réitérera sa demande ultérieurement s'il le désire. On peut aussi envisager une commande permettant de visualiser l'état de liste d'attente.

4.2.3 Tolérance aux pannes

Plusieurs problèmes de fonctionnement peuvent survenir dans la vie du Serveur d'Accessibilité, auxquels nous essayons d'apporter des solutions dans les paragraphes suivants.

Blocage du client. Le client ne donne pas l'ordre d'enlever le verrou sur le fichier. Si au bout d'un certain temps, fixé par l'administrateur système, le document est toujours indisponible, le serveur déclenche une procédure pour libérer cette information. Il rejette le client. Il restaure la dernière version du document qu'il aura sauvegardé juste avant de mettre le verrou.

Panne du serveur. En cas de panne, il faut pouvoir redémarrer le MSR proprement. Il est donc nécessaire afin d'assurer un re-démarrage correct que toutes les informations sur les accessibilités des documents soient sauvegardées sur disque. Il conserve aussi, en mémoire non-volatile, la liste des opérations effectuées avant le crash. Une fois le système redémarré, il restaurera l'état du système selon le journal qu'il conserve.

5.0 Le Serveur de Documents

Ce sous-système maintient une base de données de tous les documents gérés par le MSR, identifiés par un identificateur unique connu du Serveur de Noms. Ces documents peuvent être de tout type (ils sont multimédias).

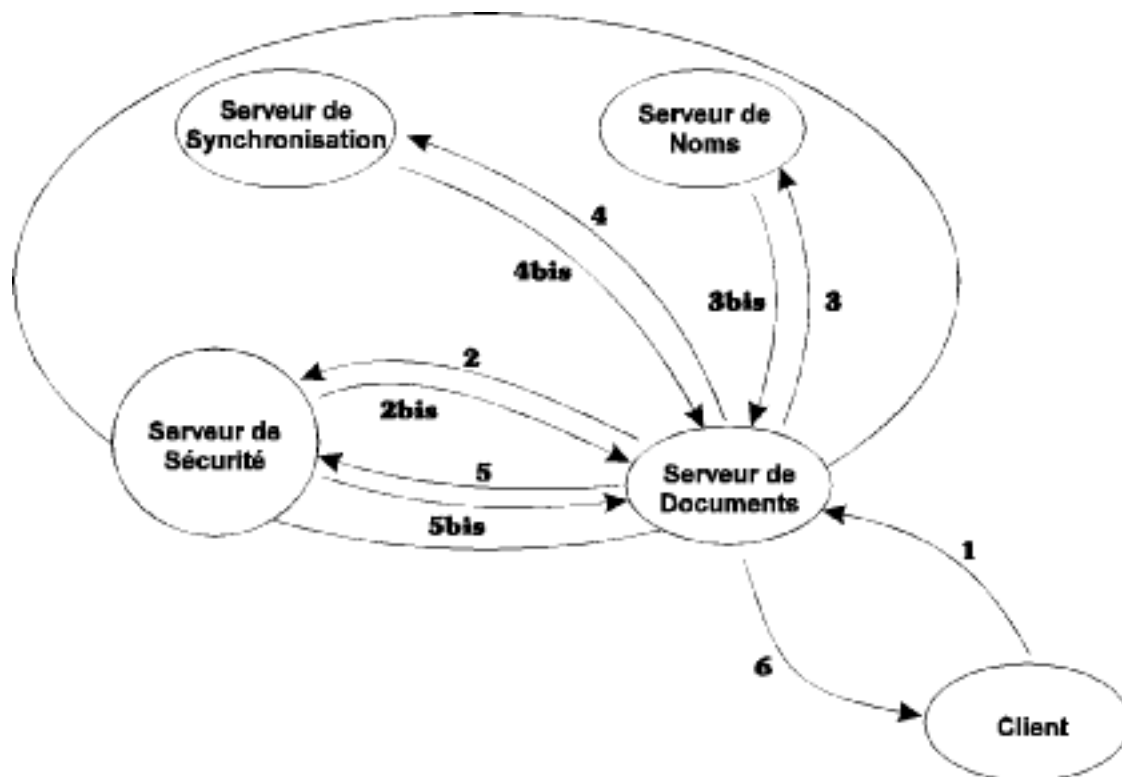
Le Serveur de Documents fait interface entre les différents services du MSR et les clients de celui-ci.

5.1 Les méthodes du Serveur de Documents

5.1.1 lire_document()

Nous détaillons ici le fonctionnement interne au MSR de cette requête.

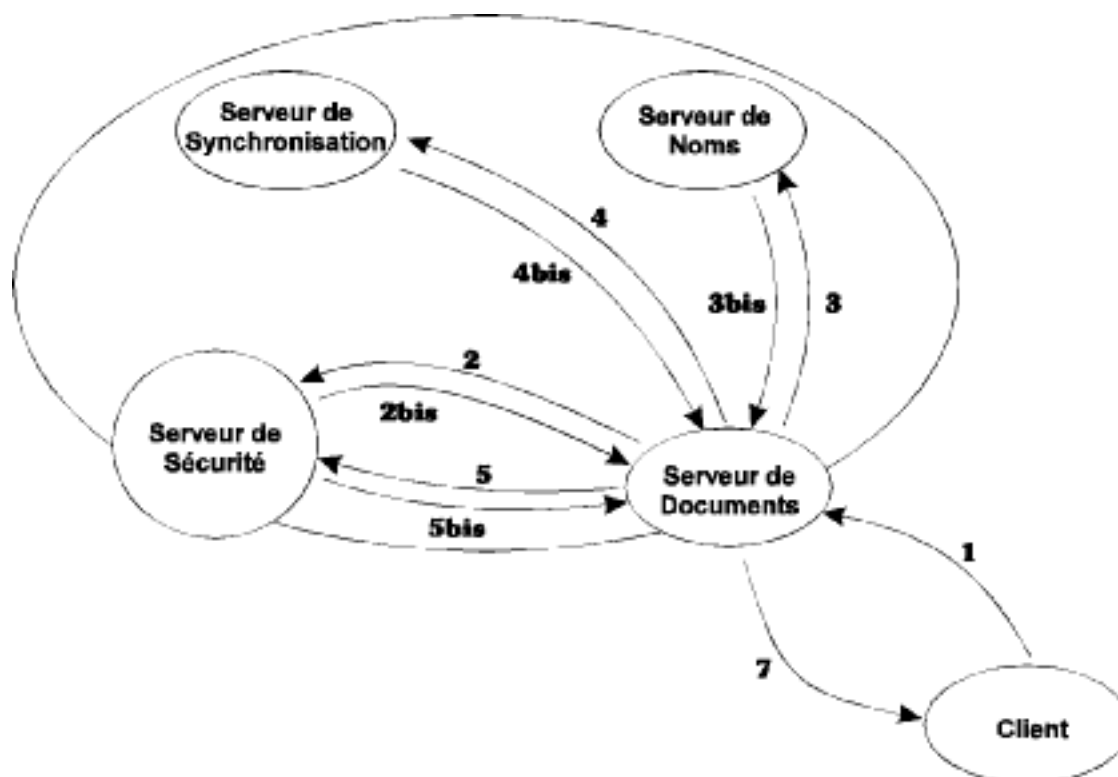
FIGURE 5. Déroulement de lire_document



1. Lors de la lecture d'un document, le client appelle la méthode avec deux arguments : nom_client et nom_document qui sont des noms X.500. La requête est signée par le client avec sa clef privée.
2. La première étape consiste à vérifier la signature de la requête en la transmettant au Serveur de Sécurité avec le nom du client, qui répond en confirmant ou non la validité de la signature.
3. Une fois la requête vérifiée, on demande au Serveur de Noms de résoudre le nom X.500 paramètre de la requête ; celui-ci retourne un identificateur unique pour le document.
4. L'identificateur de document obtenu, on s'adresse au Serveur d'Accessibilité pour connaître le statut du document : le client a-t-il le droit de lecture sur ce document, et le document n'est-il pas déjà ouvert par un autre client ?
5. Une réponse positive du Serveur d'Accessibilité déclenche la lecture des données du document dans la base du Serveur de Documents, qui fait passer ces données au Serveur de Sécurité en vue de leur encryptage.
6. Le document encrypté avec la clef publique du client est transmis à celui-ci. Cette requête est signée avec la clef privée du MSR.
7. Le client vérifie l'intégrité de la transaction en décodant la signature grâce à la clef publique du MSR, et décrypte le document à l'aide de sa clef privée.

5.1.2 écrire_document()

FIGURE 6. Déroulement de écrire_document()

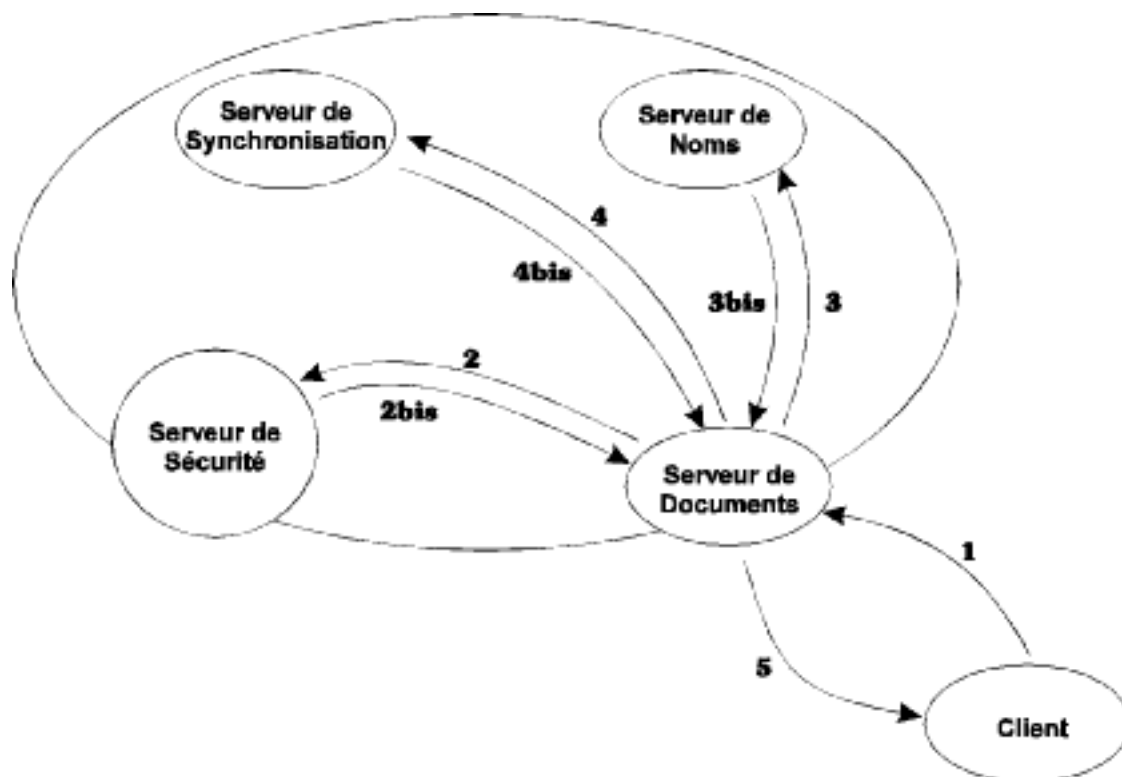


1. Lors de l'écriture d'un document, le client appelle la méthode avec trois arguments : nom_client et nom_document qui sont des noms X.500, et les données du document, encryptées avec la clef publique du serveur. La requête est signée par le client avec sa clef privée.
2. La première étape consiste à vérifier la signature de la requête en la transmettant au Serveur de Sécurité avec le nom du client, qui répond en confirmant ou en infirmant la validité de la signature.
3. Une fois la requête vérifiée, on demande au Serveur de Noms de résoudre le nom X.500 paramètre de la requête ; celui-ci retourne un identificateur unique pour le document. Éventuellement celui-ci est nouveau et ajouté dans l'arbre de nommage.
4. L'identificateur de document obtenu, on s'adresse au Serveur d'Accessibilité pour connaître le statut du document : le client a-t-il le droit d'écriture sur ce document, et le document n'est-il pas déjà ouvert par un autre client ? Dans le cas d'un nouveau document, des permissions par défaut sont générées (droit de lecture/écriture pour le créateur uniquement, par exemple).
5. Une réponse positive du Serveur d'Accessibilité déclenche l'envoi des données cryptées du document au Serveur de Sécurité pour leur décryptage grâce à la clef privée du MSR.
6. Le document décrypté est inscrit dans la base de documents du Serveur de Documents.

7. Un acquittement signé est retourné au client afin de l'informer du succès de l'opération.

5.1.1.3 `change_permissions()`

FIGURE 7. Déroulement de `change_permissions()`



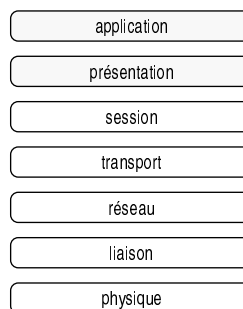
1. Lors du changement des permissions associées à un document, le client appelle la méthode avec trois arguments : `nom_client` et `nom_document` qui sont des noms X.500, et une liste de noms X.500 représentant les personnes ayant droit d'accès à ce document. Fort logiquement, le nom du client est ajouté à cette liste. Cette requête est signée avec la clef privée du client.
2. La première étape consiste à vérifier la signature de la requête en la transmettant au Serveur de Sécurité avec le nom du client, qui répond en confirmant ou en infirmant la validité de la signature.
3. Une fois la requête vérifiée, on demande au Serveur de Noms de résoudre le nom X.500 du client ainsi que la liste des personnes paramètres de la requête ; celui-ci retourne un identificateur unique pour le document ainsi que pour chaque nom de la liste. Si un nom de personne n'existe pas, l'opération est avortée.
4. Les identificateurs obtenus, on les envoie au Serveur d'Accessibilité avec le type de permission afin qu'il mette à jour sa base.
5. Un acquittement signé est retourné au client afin de l'informer du succès de l'opération.

6.0 Conclusion

L'architecture proposée dans cette étude de cas concerne les couches hautes du modèle OSI : nous nous basons sur une couche Session assurant un service fiable d'appel de procédures à distance.

FIGURE 8.

Les couches OSI étudiées



6.1 Remarques d'implémentation

La décomposition du MSR que nous avons présenté dans cette étude est bien entendu conceptuelle : elle ne doit pas certainement pas être considérée comme une solution d'implémentation idéale—ni même comme une solution d'implémentation tout court !

Il est évident qu'une distribution complète des composants de l'architecture telle que proposée dans ce document conduirait à un MSR lent, compliqué et sans doute difficilement gérable : des sous-systèmes répartis sur différentes machines d'un réseau local imposeraient un trafic réseau important et sans doute inutile.