# Réponse au Cahier des Charges et Rapport de Projet

Projet : Conception d'une architecture distribuée avec routage en oignon

Auteur : Lucas HERCHUEL
Date : Décembre 2025

## 1. Introduction

Ce document présente la réponse au cahier des charges du projet « Conception d'une architecture distribuée avec routage en oignon » réalisé dans le cadre du BUT Réseaux & Télécommunications (R3.09 / SAE 3.02). L'objectif est de concevoir et implémenter un système permettant à un client A d'envoyer un message à un client B de manière anonyme en s'appuyant sur une chaîne de routeurs virtuels et un routage en oignon.

Le principe repose sur un serveur maître central (Master) qui maintient un annuaire de routeurs, distribue leurs clés de chiffrement aux clients, et sur des routeurs qui ne connaissent que leur voisin précédent et suivant. Le client construit un « oignon » de N couches chiffrées, qu'il envoie au premier routeur ; chaque routeur retire sa couche grâce à sa clé de chiffrement et transmet le reste au suivant, jusqu'à ce que le destinataire final reçoive le message.

J'ai mené ce projet seul, en parallèle de mon alternance, ce qui signifie que toutes les parties (Master, routeur, client, base de données, interfaces graphiques, tests et documentation) ont été développées et intégrées par moi-même.

## 2. Éléments Implémentés et Non Implémentés

Le tableau ci-dessous résume la réponse point par point aux exigences techniques.

| Exigence du sujet | État | Commentaire |
|---|---|---|
| Routage multi-sauts avec sockets Python | Implémenté | Le client sélectionne un chemin aléatoire de N routeurs et envoie un oignon qui transite via R1→R2→… jusqu'au destinataire. |
| Gestion multi-threads | Implémenté | Master et routeurs utilisent des threads pour gérer plusieurs connexions simultanées. |

| | | |
|---|---|---|
| Chiffrement par couches (routage en oignon) | Implémenté | Le client construit N couches chiffrées, chacune contenant l'adresse du prochain saut + le payload chiffré pour le routeur suivant. |
| Chiffrement asymétrique simplifié | Implémenté (simplifié) | Implémenté via un chiffrement XOR symétrique par routeur (CryptoSym), plutôt qu'un RSA maison. |
| Base de données MariaDB pour routeurs et logs | Implémenté | Le Master crée la BDD, les tables routeurs et logs, stocke IP/ports/clé/statut et journalise les événements. |
| Interface graphique Master (Qt) | Implémenté | Fenêtre PyQt6 affichant l'état du serveur, la liste des routeurs, et les logs. |
| Interface graphique Client (Qt) | Implémenté | Fenêtre PyQt6 permettant la connexion au Master, la configuration du client, la sélection du destinataire, l'envoi et la réception de messages. |
| Anonymisation du chemin | Implémenté | Chaque routeur ne voit que son voisin précédent et suivant, le destinataire final ne connaît pas l'origine réelle. |
| Gestion de pannes de routeurs | Implémenté | Les routeurs/clients peuvent signaler un routeur HS, le Master met à jour son statut en DOWN et enregistre l'événement. |
| Démarrage des routeurs depuis l'interface Master | Non implémenté | Initialement prévu (M-07) dans le rendu préparatoire comme facultatif, non réalisé pour prioriser les fonctionnalités essentielles par manque de temps. |
| Chiffrement asymétrique « fort » (RSA ou équivalent) | Non implémenté | Non réaliste dans le temps imparti et sans librairie crypto ; remplacé par un XOR. |

# 3. Structure du Code, Modules, Protocole et API

## 3.1 Structure des Fichiers

- master_crypt.py : serveur central (Master) gérant la BDD, l'annuaire des routeurs et les logs.
- routeur_crypt.py : implémentation d'un routeur (génération de clé, écoute, déchiffrement, transfert, signalement de pannes).
- client.py : client graphique PyQt6 (configuration, construction d'oignon, envoi, réception).
- lancer_routeurs.py : script utilitaire lançant N routeurs en parallèle sur des ports consécutifs.
- interface_master.py : interface graphique du Master (contrôle et visualisation).
- crypt.py : module de chiffrement symétrique XOR (CryptoSym).

## 3.2 Rôle de Chaque Module

- Master (master_crypt.py) :
  - Initialise la base MariaDB (sae_routage_oignon_lh) et les tables routeurs et logs.
  - Écoute sur un port TCP (par défaut 9016) et gère les messages REGISTER, LIST et REPORTDOWN.
  - Journalise les événements (démarrage, inscriptions, erreurs, pannes).
- Routeur (routeur_crypt.py) :
  - Génère une clé symétrique via CryptoSym et s'inscrit auprès du Master.
  - Écoute sur un port TCP (8000, 8001, …), déchiffre les paquets reçus avec sa clé, extrait l'adresse du prochain saut, transfert le reste.
  - Peut signaler au Master qu'un autre routeur est HS (REPORTDOWN).
- Client (client.py) :
  - Interface Qt6 pour configurer le port d'écoute local, se connecter au Master et récupérer la liste des routeurs.
  - Construit un circuit aléatoire de N routeurs à partir de la liste du Master, construit l'oignon de N couches chiffrées, envoie au premier routeur.
  - Écoute sur un port local (6016, 6017, …) pour recevoir les messages et les afficher dans les logs.
- Lancement de plusieurs routeurs (lancer_routeurs.py) :
  - Lance automatiquement N instances de Routeur sur des ports consécutifs, les inscrit au Master et affiche un résumé (nombre de routeurs inscrits).
- Interface Master (interface_master.py) :
  - Fournit une interface Qt6 pour démarrer/arrêter le serveur, visualiser les routeurs et consulter les logs.
- Crypto (crypt.py) :
  - Implémente CryptoSym : génération de clé, chiffrement XOR, déchiffrement XOR.

## 3.3 Protocole de Communication

Master ↔ Routeurs

- Inscription d'un routeur :
  - Message :
    REGISTER,<PORT_ROUTEUR>,<CLE_SYM>
  - Effet côté Master :
    - Enregistrement (ou mise à jour) du routeur dans la table routeurs (IP réelle, port, clé, statut ACTIVE).
    - Réponse : OK ou UPDATED.
- Signalement d'une panne (REPORTDOWN) :
  - Message :
    REPORTDOWN,<IP_ROUTEUR_HS>,<PORT_ROUTEUR_HS>
  - Effet côté Master :
    - Mise à jour du statut du routeur à DOWN dans la BDD.
    - Journalisation dans logs.

Master ↔ Clients

- Récupération de la liste des routeurs (LIST) :
  - Message client → master :
    LIST
  - Réponse master → client :
    - Soit une chaîne du type :
      IP1,PORT1,CLE1|IP2,PORT2,CLE2|...
    - Soit EMPTY s'il n'y a aucun routeur actif.

Clients ↔ Routeurs

- Format d'un paquet routé :
  - Les données en clair avant chiffrement ont la forme :
    IP_SUIVANT,PORT_SUIVANT,PAYLOAD
  - PAYLOAD peut être soit :
    - une autre couche chiffrée (pour un routeur suivant),
    - le message final avec IP et port du destinataire.
- Pour le dernier routeur :
  - Après déchiffrement, il obtient :
    IP_DESTINATAIRE,PORT_DESTINATAIRE,MESSAGE_CLAIR
  - Il envoie alors MESSAGE_CLAIR directement au client destinataire.

# 4. Algorithme de Chiffrage : Implémentation, Forces, Faiblesses

## 4.1 Choix d'Implémentation

Le cahier des charges impose un chiffrement « asymétrique simplifié » sans utiliser de librairie de cryptographie. Après de nombreuses recherches et tests d'implémentation de chiffrement asymétrique, j'ai décidé de réaliser un chiffrement symétrique qui est plus simple à réaliser et à comprendre en terme de mathématique. J'ai fait ce choix car je n'arrivais pas à comprendre les calculs à réaliser pour du chiffrement asymétrique et je passais beaucoup trop de temps dessus au détriment des autres fonctionnalités du projet.

- utilisation d'un chiffrement symétrique XOR par routeur via la classe CryptoSym
- chaque routeur génère une clé aléatoire (chaîne de caractères) au démarrage et l'envoie au Master lors du REGISTER
- le client récupère cette clé pour chaque routeur via la commande LIST et l'utilise pour chiffrer la couche correspondante.

Le chiffrement/déchiffrement repose sur une opération XOR entre chaque octet du texte et un octet de la clé.

## 4.2 Processus de Construction de l'Oignon

Pour un circuit R1 → R2 → R3 et un message M destiné à B_IP:B_PORT :

1. Couche 3 (la plus profonde)
    - Données : B_IP,B_PORT,M
    - Chiffrement : C3 = Enc_XOR(K_R3, "B_IP,B_PORT,M")
2. Couche 2
    - Données : R3_IP,R3_PORT,C3
    - Chiffrement : C2 = Enc_XOR(K_R2, "R3_IP,R3_PORT," + C3)
3. Couche 1 (la plus externe)
    - Données : R2_IP,R2_PORT,C2
    - Chiffrement : C1 = Enc_XOR(K_R1, "R2_IP,R2_PORT," + C2)

Le client envoie alors C1 au routeur R1.

## 4.3 Processus de Déchiffrement par les Routeurs

Chaque routeur retire une couche :

- R1 :
    - reçoit C1
    - calcule Dec_XOR(K_R1, C1) → "R2_IP,R2_PORT,C2"
    - extrait R2_IP, R2_PORT et C2

    - envoie C2 à R2
- R2 :
    - reçoit C2
    - calcule Dec_XOR(K_R2, C2) → "R3_IP,R3_PORT,C3"
    - extrait R3_IP, R3_PORT et C3
    - envoie C3 à R3
- R3 :
    - reçoit C3
    - calcule Dec_XOR(K_R3, C3) → "B_IP,B_PORT,M"
    - extrait B_IP, B_PORT, M
    - envoie M au client destinataire.

Le client B reçoit M depuis R3, sans connaître le chemin complet ni l'IP réelle de A.

### 4.4 Forces

- XOR est simple à comprendre et à implémenter .
- Il permet de se concentrer sur la logique de couches et d'anonymisation sans se perdre dans la complexité d'un RSA.
- Chaque routeur ne peut déchiffrer qu'une couche et ne voit que le « prochain saut » ou le destinataire final.

### 4.5 Faiblesses

- Sécurité faible :
    - XOR seul est fragile et ne protège pas contre des attaques sérieuses.
    - L'algorithme n'est pas adapté à un usage réel de confidentialité, mais seulement à une démonstration de principe.

## 5. Difficultés Rencontrées et Choix Techniques

### 5.1 Multithreading et Sockets

- Gestion de plusieurs connexions simultanées sur le Master et les routeurs avec un thread par client ou par message, nécessitant une attention particulière à la concurrence sur la base de données (sémaphore pour les accès BDD).
- Gestion des erreurs réseau (timeouts, ports déjà utilisés, routeurs injoignables).

### 5.2 Gestion de Projet en Solo avec Alternance

- Charge de travail élevée : développement complet (serveur, routeurs, client, BDD, GUI), tests multi-machines, documentation détaillée, plus les contraintes de l'alternance (horaires de travail, transport, ...).
- Conséquences :
    - Respect des fonctionnalités essentielles.
    - Simplification de certains points (chiffrement, fonctionnalités bonus).
    - Adaptation du planning au fur et à mesure, en se concentrant sur un système fonctionnel et documenté.

# 6. Gestion de Projet : Écarts entre Planning et Réalité

Le planning initial était ambitieux pour un projet réalisé seul, en parallèle d'une alternance. En pratique :

Les jalons J1 à J3 ont été tenus :

- Prototype sockets/threads.
- Chaîne de routage simple.
- Master + BDD + enregistrement dynamique des routeurs.

Le rendu final était initialement prévu pour le 24 décembre, cependant une fois que j'ai repris l'alternance il m'a été impossible de suivre mon planning initial par manque de temps le soir en rentrant. Tout a donc été retardé et j'ai donc du adapter le planning en fonction. De plus j'ai dû finaliser le code, réaliser les tests et rédiger la documentation entre le 25 et le 31 décembre afin de pouvoir rendre le projet dans le temps imparti.

# 7. Bilan par Rapport au Cahier des Charges

Le projet réalisé satisfait les points principaux du cahier des charges :

- Routage en oignon multi-sauts : implémenté et démontrable (Client A → R1 → R2 → … → Client B).
- Architecture distribuée (Master, routeurs, clients) : fonctionnelle sur plusieurs machines/VM.
- Gestion multi-threads : Master et routeurs gèrent plusieurs connexions simultanées.
- Base MariaDB : utilisée pour stocker les routeurs et les logs, avec un schéma adapté.
- Interfaces graphiques : Master et client en PyQt6, permettant une démonstration claire.
- Documentation : guides d'installation, d'utilisation, et ce document de réponse technique et de gestion de projet.