

Conception d'une architecture distribuée avec routage en oignon

Projet d'implémentation d'un système anonyme client-serveur

R3.09 | Programmation événementielle

SAÉ 3.02 | Développer des applications communicantes

Frédéric DROUHIN - BUT 2 - 2025-2026

Introduction au projet

- **Objectif principal :**
Concevoir et implémenter un système de communication anonyme entre plusieurs clients à l'aide de routeurs virtuels.
- **Principe clé :**
Anonymat via le **routage en oignon** (chaînes de routeurs qui ne connaissent que leur voisin direct).
- **Pourquoi ?**
Garantir l'anonymat des communications sur un réseau distribué.

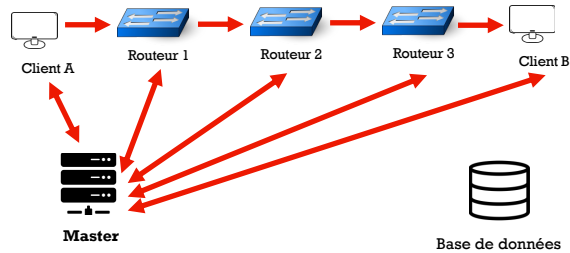
Créer des outils et applications informatiques pour les R&T

- AC23.01 | Automatiser l'administration système avec des scripts
- AC23.02 | Développer une application à partir d'un cahier des charges donné, pour le Web ou les périphériques mobiles
- AC23.03 | Utiliser un protocole réseau pour programmer une application client/serveur
- AC23.04 | Installer, administrer un système de gestion de données
- AC23.05 | Accéder à un ensemble de données depuis une application et/ou un site web
- CE3.01 | en étant à l'écoute des besoins du client
- CE3.02 | en documentant le travail réalisé
- CE3.03 | en utilisant les outils numériques à bon escient
- CE3.04 | en choisissant les outils de développement adaptés
- CE3.05 | en intégrant les problématiques de sécurité

Objectifs techniques et Intérêt pédagogique

- Implémenter un système client-serveur multi-routeurs utilisant les sockets Python sur au minimum 2 machines.
- Gérer la communication simultanée par threads.
- Concevoir un chiffrement asymétrique simplifié (clé publique/privée) et un système de couches de chiffrement.
- Développer une interface Qt pour visualiser les connexions et la topologie et le client
- Stocker les informations de routage et de clé dans une base de données MariaDB.
- **Objectif général :** concevoir une architecture distribuée composée de :
 - Plusieurs **routeurs virtuels** interconnectés ;
 - Un **serveur maître** (superviseur, générateur de règles et de routes) ;
 - Plusieurs **clients** souhaitant échanger des messages **de manière anonyme**, via plusieurs routeurs.
- **Principe :** chaque message traverse une **chaîne de routeurs**, où chaque routeur ne connaît **que son voisin précédant et suivant**. Les données sont **chiffrées par couches successives**, comme dans Tor (The Onion Router).

Objectifs techniques



- Routage multi-sauts avec sockets Python
- Gestion multithread des connexions
- Chiffrement asymétrique (clé publique/privée simplifiée)
- Anonymisation par couches (routage en oignon)
- Base de données MariaDB (clés, tables de routage)
- Interface Qt (visualisation des connexions, statistiques, client)

Gestion de projet

- Chaque étudiant ou groupe devra planifier et suivre le déroulement de son projet :
 - Définition des tâches et répartition claire des rôles
 - Utilisation d'un outil de suivi (Trello, GitHub Projects, etc.)
 - Rédaction d'un planning prévisionnel
 - Mise à jour du dépôt Git à chaque étape
 - Production d'un journal de bord (commit messages explicites à extraire et à rendre dans le rapport de planning)

Rendu d'un planning : 20 novembre à 23:59

Durée et gestion de projet

FA

- Durée :
 - 15,75h encadrés de R3.09
 - 8,75h encadrés de SAE
 - 18,5h d'autonomie
- Groupe (au choix)
 - 1 étudiant : notation sur 30
 - 2, 3, 4 étudiants : notation sur 20

FI

- Durée
 - 15,75h encadrés de R3.09
 - 10,5h encadrés de SAE
 - 39,5h d'autonomie
- Groupe (aux choix)
 - 1 étudiant : notation sur 25
 - 2 étudiants : notation sur 20

Date de rendu :

Gestion de projet : 20 novembre à 23:59

Projet final : 31 décembre à 23:59

Architecture logique

- **Master** : définit la topologie et distribue les clés de chiffrement à chaque routeur.
- **Clients** : construisent les messages "en oignon" (plusieurs couches de chiffrement).
- **Routeurs** : déchiffrent **une seule couche**, découvrent l'adresse du **prochain saut**, et transmettent le reste du message.
- **Base de données** : stocke les tables de routage et les clés de chiffrement.

Destination	Next Hop	Interface	RuleID
clientA	routeurB	Eth0	1
ClientB	routeurC	Eth1	2

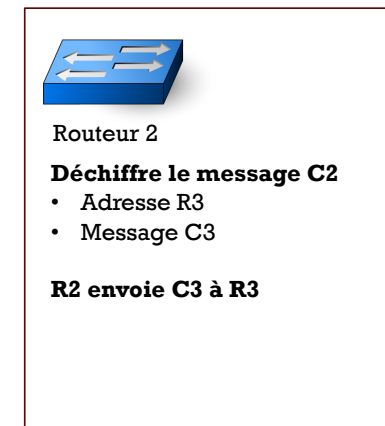
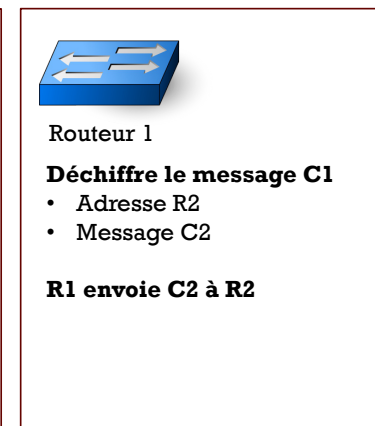
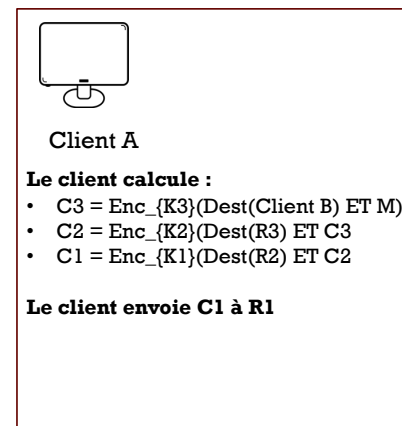
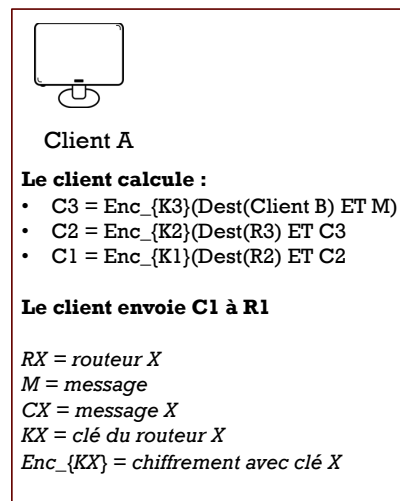
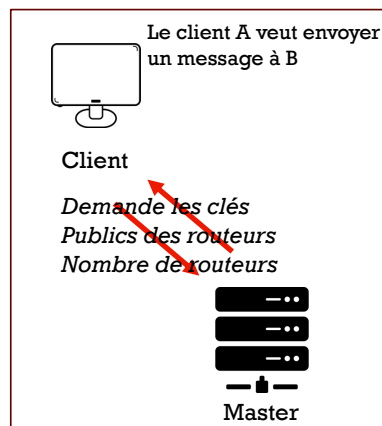
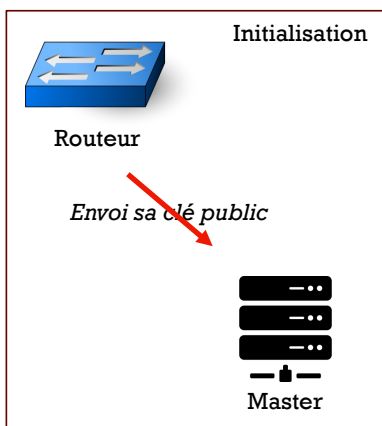
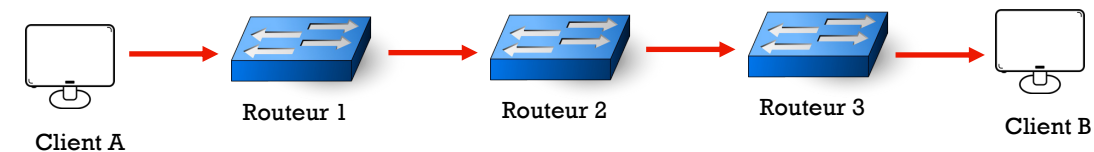
Tableau 1 exemple de table de routage

Architecture logique

- **Master :**
 - Le **master** génère une liste de routeurs virtuels, leurs adresses et une clé (symétrique ou RSA).
 - Il distribue les **clés publiques** des routeurs aux clients.
- **Routeur :**
 - Chaque routeur enregistre sa clé privée localement.
 - Le routeur reçoit un message, déchiffre et envoi au suivant
- **Client :**
 - Le client choisit un **chemin aléatoire** (ex. 3 routeurs).
 - Il chiffre le message **en plusieurs couches** :
 - Couche 3 : chiffrée avec la clé du Routeur 3 (contient message final + destinataire).
 - Couche 2 : chiffrée avec la clé du Routeur 2 (contient adresse du Routeur 3 + couche 3).
 - Couche 1 : chiffrée avec la clé du Routeur 1 (contient adresse du Routeur 2 + couche 2).
 - Il obtient un "oignon" qu'il envoie au premier routeur.



Un exemple de scénario minimal





Routeur 2

Déchiffre le message C2

- Adresse R3
- Message C3

R2 envoie C3 à R3



Routeur 3

Déchiffre le message C3

- Adresse Client B
- Message (M)

R3 envoie M à Client B



Client B

Client B reçoit le message

09/11/2025

Aspects techniques à implémenter

Composant	Objectif	Points techniques
Sockets TCP/UDP	Communication entre clients/routeurs/master	Gestion multi-threads, ports dynamiques
Threads	Gérer plusieurs connexions simultanées	Serveur multi-clients
Crypto	Implémenter un chiffrement simple	Chaque routeur a une paire de clés
Routing	Appliquer les règles du master	Table des prochains sauts
Base MariaDB	Stocker les clés, routes, logs	Table, requêtes SQL
Qt (optionnel)	Interface du master ou d'un client	Visualiser le réseau et les routes
Journalisation	Logs anonymisés	Aucun routeur ne connaît l'origine complète du message

09/11/2025

Contraintes de développement

- Langage : Python
- Bibliothèques :
 - Socket, threading, MariaDB, PyQt5/PyQt6
 - à l'exception de tout autre librairie (par exemple pas de librairie de cryptographie)
- Développement intégralement manuel (aucun code généré automatiquement)
- Structure de code claire et commentée
- Lien GitHub obligatoire pour le dépôt final
- L'environnement de test montrant l'usage de plusieurs machines ou VM.
- Le client et le master doivent être des interfaces graphiques :
 - Client : connexion et envoi de message (non bloquant)
 - Master : visualisation, logs, distribution des clés (éventuellement démarrage des routeurs)

09/11/2025

Livrables

- Code source complet au travers d'un lien git
- Documentation de réponse au cahier des charges
 - Eléments implémentés et non implémentés
 - Structure du code, modules, protocole, API
 - Description de l'algorithme de chiffrage (forces et faiblesses)
 - Rapport de projet avec gestion du projet
- Documentation d'installation et d'utilisation
 - Etapes pour installer et lancer le projet
 - Guide d'utilisation des interfaces et commandes
- Vidéo de démonstration (5-10 minutes)
 - L'environnement de test (machines, VM)
 - Le lancement du système complet
 - Le fonctionnement des clients, routeurs et du master
 - Une démonstration de l'anonymisation des flux (couches de chiffrement) avec par exemple une trame échangée montrant l'anonymisation
 - Un commentaire oral ou sous-titré expliquant les choix techniques
- Travail de groupe : fiche individuelle expliquant son rôle (introduction) dans le projet en décrivant de manière détaillée les éléments qu'il a travaillé (corps) et une conclusion sur son apprentissage en analysant sa compétence au travail sur du référentiel de compétences
- En fonction du planning une présentation de certaines parties de code peut être demandé à un ou plusieurs étudiants.