Documentation Technique - SAE 3.02

Routeur Virtuel Distribué avec Anonymisation

---

1. Livrables et Périmètre

1.1 Fichiers Fournis

- `master_gui.py` : Serveur central avec interface PyQt + accès base de données
- `routeur.py` : Routeur virtuel, gestion des couches d'oignon et relais
- `client_gui.py` : Client avec interface PyQt, envoi/réception
- `README.txt` : Documentation d'installation, configuration, lancement
- `CAHIER_DES_CHARGES.md` : Documentation technique complète

1.2 Couverture du Cahier des Charges

Exigence	**Statut**	**Détail**
Code source complet	✓	3 fichiers Python principaux, versionnés
Documentation réponse	✓	Ce document + README
Éléments impl./non impl.	✓	Section 2
Structure, modules, API	✓	Section 3
Algorithme chiffrage	✓	Section 4

Exigence	Statut	Détail
Gestion de projet	✓	Section 5
Installation et utilisation	✓	README + Section 6

---

2. Fonctionnalités : Implémenté / Non Implémenté

2.1 Implémenté

Architecture et Réseau

- Architecture master – routeurs – clients avec communications TCP
- Routage en oignon : chaque routeur ne voit que la couche qu'il déchiffre
- Chiffrement XOR symétrique par routeur, clé propre par routeur
- Enregistrement dynamique (routeurs_dyn, clients_dyn dans MariaDB)
- Monitoring des routeurs (mise à jour du champ alive)
- Réponse ASK_ROUTERS filtrée sur routeurs vivants

Interfaces

- GUI Master : supervision simple du système
- GUI Client : choix de route, IP/port destinataire, message, historique
- Enregistrement automatique au démarrage

2.2 Non Implémenté / Limites

- Pas d'authentification forte ni de chiffrement industriel (AES, TLS)

---

3. Architecture et Structure

3.1 Organisation Générale

```
text
SAE3.02/
├── master_gui.py        # Master + GUI + BDD
├── routeur.py           # Routeur virtuel
├── client_gui.py        # Client + GUI
├── README.txt           # Installation
└── CAHIER_DES_CHARGES.md  # Documentation
```

3.2 Modules et Dépendances

Imports Standard:

- socket : Communication TCP
- threading : Threads pour écoute serveur
- time : Délais, monitoring

Imports Tiers:

- PyQt5.QtWidgets, PyQt5.QtCore : GUI
- mysql.connector : Base de données

3.3 Protocole de Communication

Enregistrement Routeur

```
text
REGISTER_ROUTER|nom_routeur|ip_local|port_local|clef_secrète
```

Exemple: REGISTER_ROUTER|R1|192.168.1.21|5101|cléR1

Enregistrement Client

```
text
REGISTER_CLIENT|nom_client|port_local
```

Exemple: REGISTER_CLIENT|CLIENT_A|5200

Récupération Routeurs

```
text
Requête: ASK_ROUTERS
```

Réponse:

```
ROUTERS
nom;ip;port;clef
...
END
```

3.4 Flux d'un Message

1. Client émet : `build_onion()` crée l'oignon couche par couche
2. Routeur 1 reçoit : déchiffre couche 1 → voit adresse R2
3. Routeur 2 reçoit : déchiffre couche 2 → voit `0.0.0.0:0000`
4. Routeur 2 extrait : IP, port, message du centre
5. Routeur 2 livre : connexion TCP et envoi du message
6. Client destinataire : reçoit sur son socket serveur

---

4. Algorithme de Chiffrage

4.1 Principe

Type : Chiffrement par flux (stream cipher) XOR symétrique
Clé : Chaîne de caractères fournie par l'administrateur
Mode : Répétition cyclique de la clé sur le plaintext

4.2 Implémentation

```python
def xor_layer(data, key_str):
    key = key_str.encode() if isinstance(key_str, str) else key_str
    return bytes(b ^ key[i % len(key)] for i, b in enumerate(data))
```

Exemple : Plaintext `"R2"` = [82, 50] avec clé `"cléR1"` donne [53, 126]

4.3 Points Forts

- Symétrique : Même clé pour chiffrer/déchiffrer
- Léger : Implémentation simple, pas de dépendances externes
- Pédagogique : Illustre bien la notion de couches successives
- Performant : Déchiffrement rapide

4.4 Faiblesses

1. Non cryptographiquement sûr : Trivial à casser si plaintext/ciphertext connus
2. Clés faibles : Courte clé se répète, analyse de fréquence possible
3. Pas d'intégrité : Pas de détection de modifications
4. Pas d'authentification : Pas de vérification d'identité
5. Réutilisation de clés : Même clé pour tous les messages

Ces limitations sont assumées dans le contexte pédagogique du projet.

---

5. Gestion de Projet

5.1 Phases de Développement

Phase	Tâches	Durée	Statut
1	Architecture, protocole, design BD	W1-2	✓
2	Master GUI + monitoring	W2-3	✓
3	Routeur avec oignon XOR	W3-4	✓
4	Client GUI + protocole	W4-5	✓
5	Tests multi-machines	W5-6	✓
6	Documentation	W6	✓

5.2 Ressources

- Langage : Python 3.10+
- Framework : PyQt5 (GUI), mysql-connector-python (BDD)
- Base de données : MariaDB 10.x
- Plateforme : Multi-plateforme (macOS, Linux, Windows)

5.3 Défis et Solutions

Défi	Problème	Solution
Connectivité	Clients distants non joignables	bind("", port) + pare-feu
BDD	Routeurs marqués alive=0	Heartbeat dans monitor_routers()
Oignon	Ordre/padding des couches	Documentation format 21 bytes

---

6. Installation et Utilisation

6.1 Prérequis

- Python 3.10+
- PyQt5 : `pip install PyQt5`
- MySQL : `pip install mysql-connector-python`
- MariaDB/MySQL installé localement

6.2 Configuration Base de Données

```sql
CREATE DATABASE sae3;
CREATE TABLE routeurs_dyn (
  id INT PRIMARY KEY AUTO_INCREMENT,
  nom VARCHAR(50), ip VARCHAR(50),
  port INT, clef VARCHAR(255),
  alive TINYINT DEFAULT 0
);
CREATE TABLE clients_dyn (
  id INT PRIMARY KEY AUTO_INCREMENT,
  nom VARCHAR(50), port INT
```

);

6.3 Configuration Scripts Python

Adapter dans chaque script :

```
python
# master_gui.py / routeur.py
DB_CFG = dict(user='saeuser', password='password',
              host='localhost', database='sae3')
MASTER_ADDR = ("192.168.1.64", 5100)  # Adapter IP

# Tous les fichiers
s.bind(("", port))  # Important: "" et non "localhost"
```

6.4 Lancement

1. Master : `python master_gui.py`
2. Routeurs : `python routeur.py 5101 R1`
3. Clients : `python client_gui.py 5200 CLIENT_A`

6.5 Utilisation Client

1. Bouton « Rafraîchir routeurs » pour récupérer la liste
2. Saisir : IP destinataire, port destinataire, route (R1,R2), message
3. Cliquer « Envoyer »
4. Client destinataire affiche le message reçu

---

7. Conclusion

Ce projet implémente une architecture de routage en oignon distribuée avec:

- Un serveur central (master) supervisant l'état du réseau
- Des routeurs virtuels relayant les messages de façon anonyme
- Des clients avec interface graphique permettant l'envoi via routes personnalisées
- Un protocole simple et extensible basé sur TCP
- Un chiffrement symétrique XOR illustrant le principe des couches successives

Les limites acceptées (pas d'AES, d'authentification, d'intégrité) ne sont pas demandées par le cahier des charges et n'impactent pas la démonstration du concept d'anonymisation par oignon.