Documentation Technique du Projet

SAE 302 – Conception d’une architecture distribuée
avec routage en oignon

Théo-Félix ADAM
BUT RT21 – DevCloud

Table des matières

1. DESCRIPTION

ROTATOR© v1 ou *Routage en Oignon de la Télécommunication avec Automatisation de la Transmission Ou de la Réception version 1* est une messagerie chiffrée utilisant le routage en oignon.

2. INSTALLATION

2.1. INSTALLATION DE LA BASE DE DONNÉE

Pour installer MariaDB, il suffit de se rendre sur le site https://mariadb.com/downloads/. Pour ce projet, nous allons utiliser la dernière version en date, à savoir la 11.8.5.

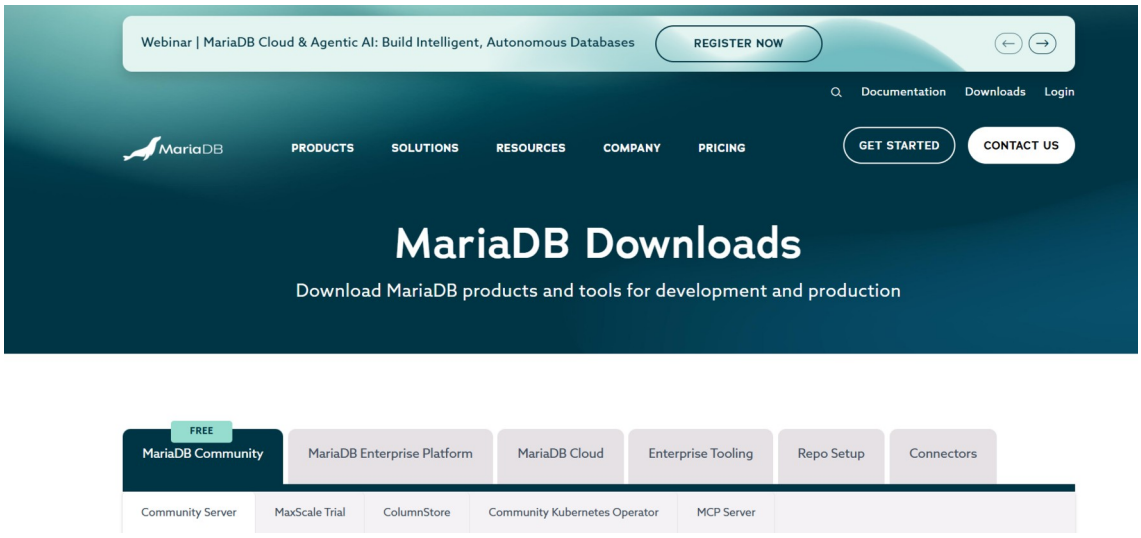


FIGURE 1: SITE DE TÉLÉCHARGEMENT MARIADB

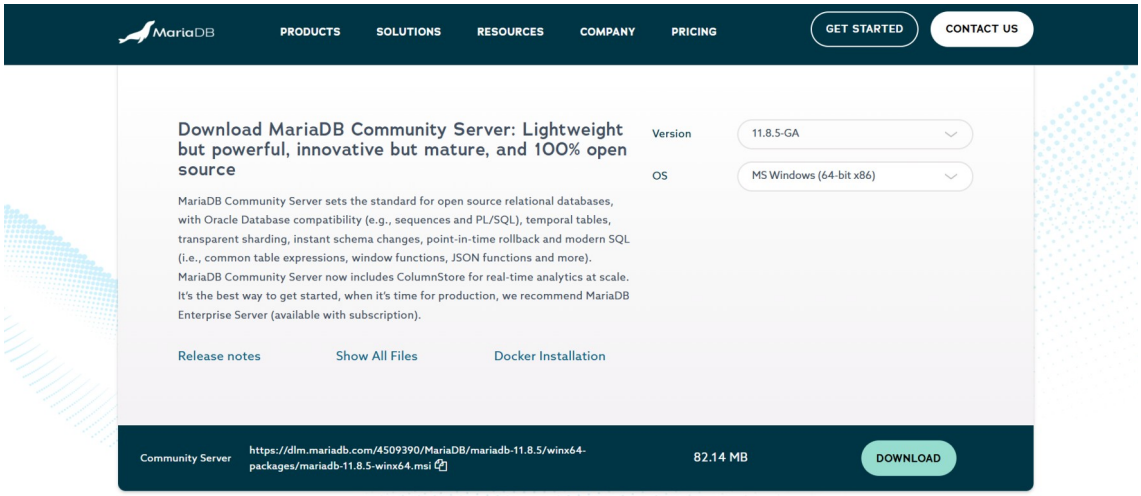
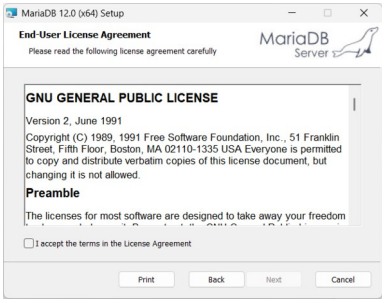
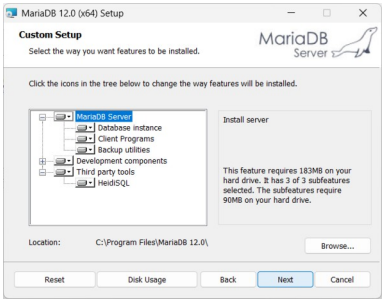
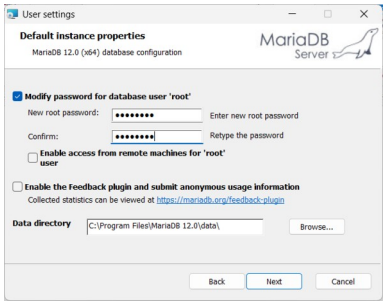
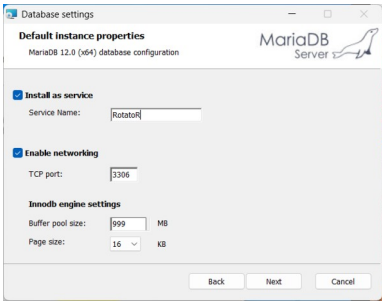


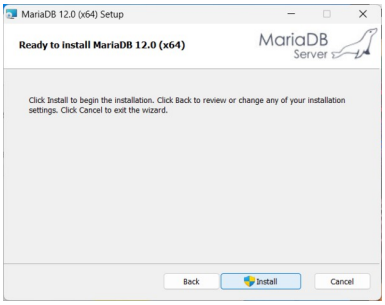
FIGURE 2: TÉLÉCHARGEMENT DE MARIADB

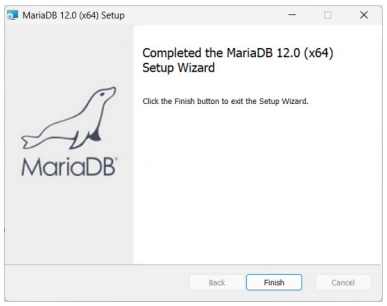












Pour accéder directement depuis le cmd, il est possible d'inclure le chemin vers MariaDB dans le cmd. Pour ce faire, il faut ouvrir l'éditeur de variables d'environnement et de copier le chemin du fichier bin, pour mon cas : C:\Program Files\MariaDB 12.0\bin.

Ensuite, dans un invité de commande, vous pouvez vous authentifier avec root et le mot de passe que vous avez configuré.

2.1.1. Création De L'utilisateur Maître

Pour que le programme ait accès à une base de données, il vous faut créer l'utilisateur maître. D'abord, connectez-vous à la base de données.

```
mysql -u root -p
```

Il vous est demandé un mot de passe. Vous devrez alors renseigner le mot de passe que vous avez crée lors de la phase précédente.

Ensuite, vous devez créer l'utilisateur « master ».

```
CREATE USER 'master'@'localhost' IDENTIFIED BY '*7B70C4544E0E8497ECA5D359C9D44E7DCF7282A9';
```

Si jamais vous n'arrivez pas à créer l'utilisateur, c'est qu'il existe peut-être déjà. Vérifier alors les utilisateurs avec la commande suivante.

```
SELECT User, Host FROM mysql.user;
```

Pour supprimer cet utilisateur :

```
DROP USER 'master'@'host';
```

Et si un utilisateur porte déjà le nom de « master » et que vous ne voulez pas le supprimer, vous pouvez alors modifier le nom, mais pas le mot de passe. Dans l'interface graphique du maître, vous devrez modifier le nom d'utilisateur, qui de base est « master ».

Maintenant que vous avez créé l'utilisateur maître, vous devez lui attribuer des droits.

```
GRANT ALL PRIVILEGES ON *.* TO 'master'@'localhost' WITH GRANT OPTION;
```

Pour mettre à jour les droits.

```
FLUSH PRIVILEGES;
```

Pour vérifier que le maître a tous les droits, vous pouvez utiliser :

```
SHOW GRANTS FOR 'master'@'localhost';
```

Vous devriez voir « GRANT ALL PRIVILEGES » en début de ligne.

2.1.2. Diverses Commandes

Si vous voulez regarder les tables générées par le maître, vous pouvez utiliser la commande suivante.

```
SHOW TABLES;
```

Pour voir les données de la table :

```
SELECT * FROM nom_table;
```

Pour regarder l'historique, entrez dans la table historique :

```
SELECT * FROM historique;
```

Vous trouverez un contenu comme suivant.

```
MariaDB [rotator]> select * from historique;
+----+---------------------+-------------------------------------------------------------------------------+
| id | date                | description                                                                   |
+----+---------------------+-------------------------------------------------------------------------------+
|  3 | 2025-12-22 11:51:22 | Démarrage serveur                                                             |
|  4 | 2025-12-22 11:53:57 | Création de la base de données.                                               |
|  5 | 2025-12-22 11:56:05 | Démarrage du maître.                                                          |
|  6 | 2025-12-22 11:58:12 | Démarrage du maître.                                                          |
|  7 | 2025-12-22 12:15:10 | Démarrage du Maître.                                                          |
|  8 | 2025-12-22 12:15:10 | Maître en écoute sur le port 5027 de 192.168.0.195.                           |
| 11 | 2025-12-22 12:19:52 | Démarrage du Maître.                                                          |
| 12 | 2025-12-22 12:19:52 | Maître en écoute sur le port 24944 de 192.168.0.195.                          |
| 15 | 2025-12-22 12:28:53 | Démarrage du Maître.                                                          |
| 16 | 2025-12-22 12:28:53 | Maître en écoute sur le port 19883 de 192.168.0.195.                          |
| 17 | 2025-12-22 12:29:05 | Connexion d'un appareil ('192.168.0.127', 63545).                             |
| 18 | 2025-12-22 12:29:05 | L'appareil ('192.168.0.127', 63545) a demandé la liste des appareils.         |
| 19 | 2025-12-22 12:38:32 | Démarrage du Maître.                                                          |
| 20 | 2025-12-22 12:38:32 | Maître en écoute sur le port 28277 de 192.168.0.195.                          |
+----+---------------------+-------------------------------------------------------------------------------+
14 rows in set (0.002 sec)
```

FIGURE 3: HISTORIQUE

2.2. BIBLIOTHÈQUES PYTHON

Ouvrez n'importe quel programme avec Thonny. Aller dans « Outils » et « Ouvrir la console du système… » et entrez la commande :

```
pip install -r requis.txt
```

Vous pouvez vérifier que tous les paquets soient correctement installés, à savoir sympy, PyQt6, mariadb grâce à la commande suivante :

```
python -m pip check
```

3. INTERFACES GRAPHIQUES

3.1. MAÎTRE

3.1.1. Page Principale

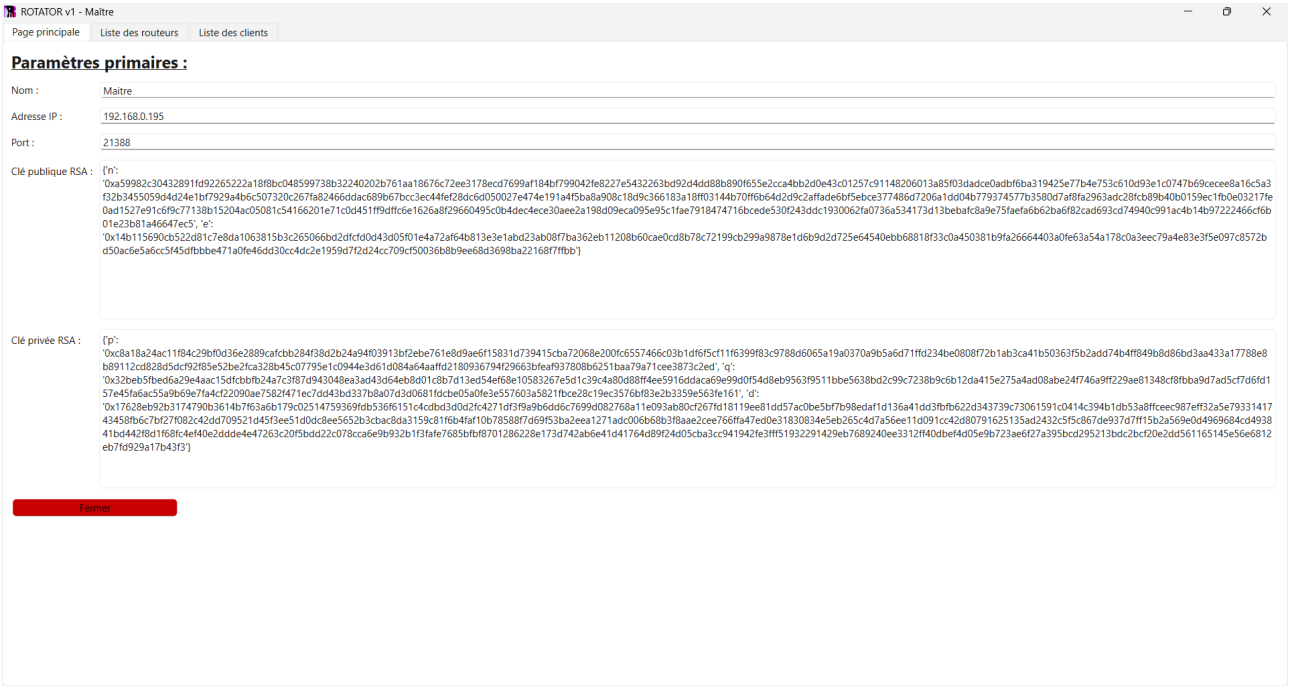


FIGURE 4: PAGE PRINCIPALE - GUI MAÎTRE

La page principale de l'interface du Maître comprend ses paramètres primaires, à savoir son nom, son adresse IPv4, son port et ses clés publique et privée RSA.

Pour fermer l'interface et interrompre le programme proprement, il faut appuyer sur le bouton rouge.

3.1.2. Liste Des Routeurs

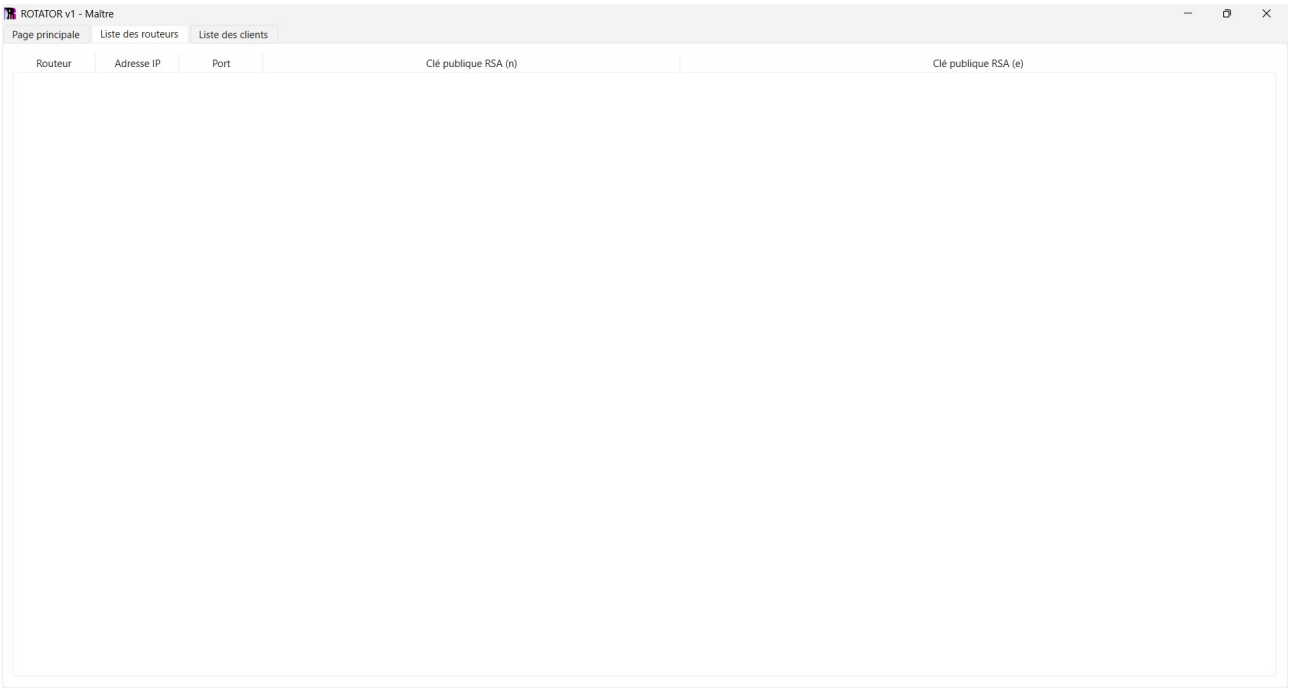


FIGURE 5: LISTE DES ROUTEURS - GUI MAÎTRE

3.1.3. Liste Des Clients

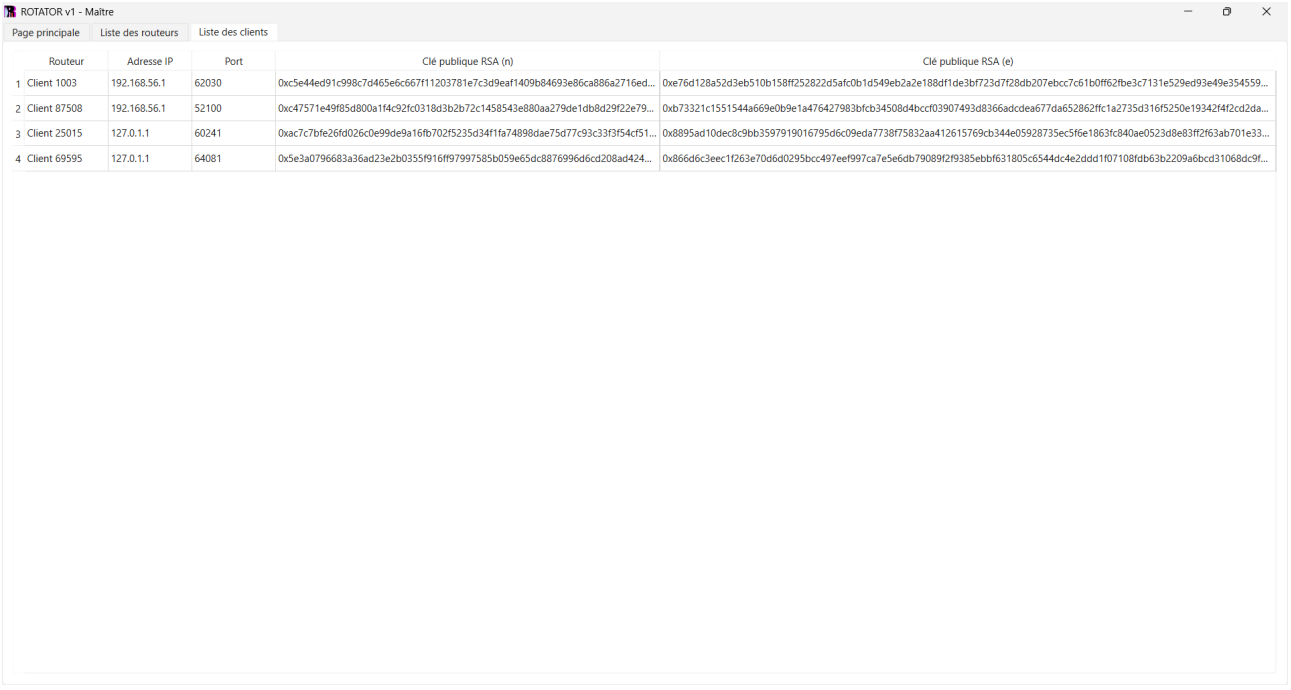


FIGURE 6: LISTE DES CLIENTS - GUI MAÎTRE

La liste des routeurs et des clients, enregistrés depuis la base de données. Le programme récupère toutes les minutes les données. Ainsi, l'utilisateur peut savoir qui est connecté.

3.2. CLIENT

3.2.1. Page Principale

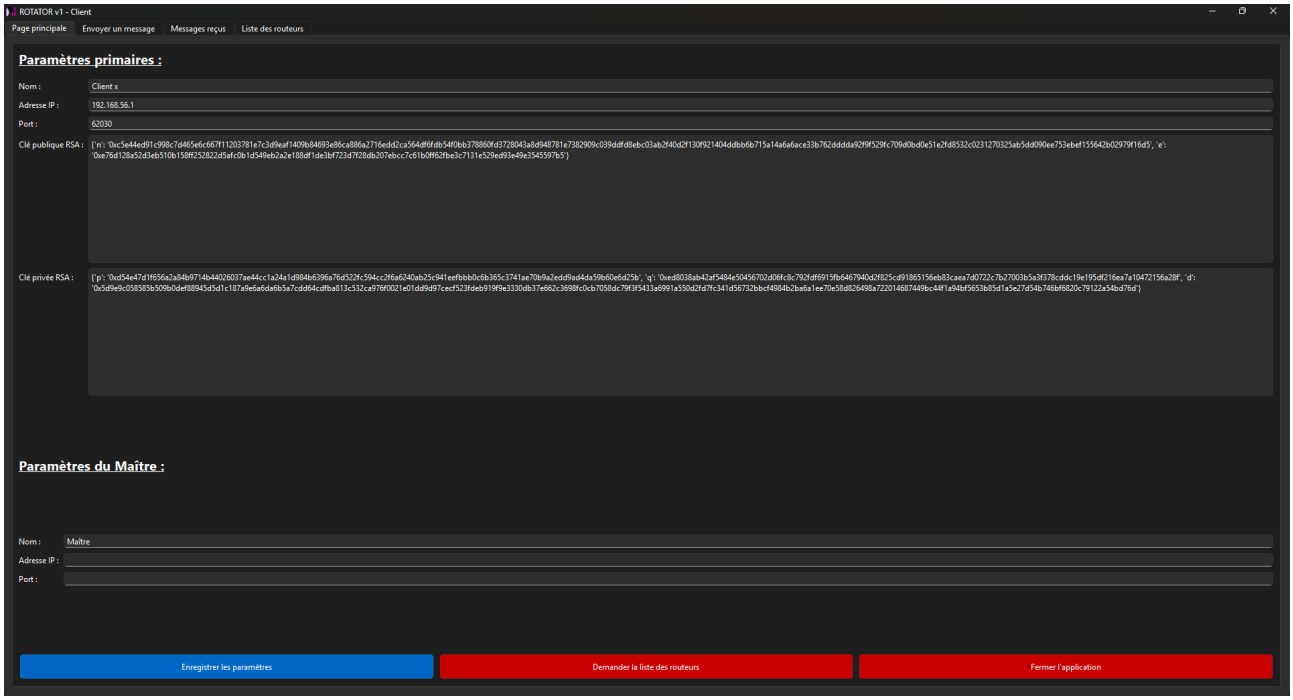


FIGURE 7: PAGE PRINCIPALE - GUI CLIENT

Sur la page principale, le client a accès à tous ses paramètres primaires, dont son nom, son adresse IPv4, son port d'écoute et ses clés privée et publique RSA générées. Plus bas, le client a accès aux paramètres du Maître (adresse IPv4 et port), qu'il doit renseigner pour assurer la communication.

Quand les paramètres du Maître ont été rentrés et que donc le Maître est en fonctionnement, alors le client doit cliquer sur « Enregistrer les paramètres », ainsi le programme envoie au Maître ses informations et nous rend visibles des autres.

À savoir que si les paramètres du Maître sont mauvais, alors la communication entre le client et le Maître ne peut pas se faire. Alors, les paramètres du Maître s'allument en rouge pour signifier au client qu'il a rentré les mauvais paramètres ou que le Maître ne communique plus.

Ensuite, le client peut demander la liste des routeurs au Maître en appuyant sur « Demander la liste des routeurs ». Il peut alors voir la liste des routeurs sur l'onglet « Liste des Routeurs ». De plus, le client peut maintenant voir les destinataires, routeurs…

3.2.2. Page D’envoi De Message

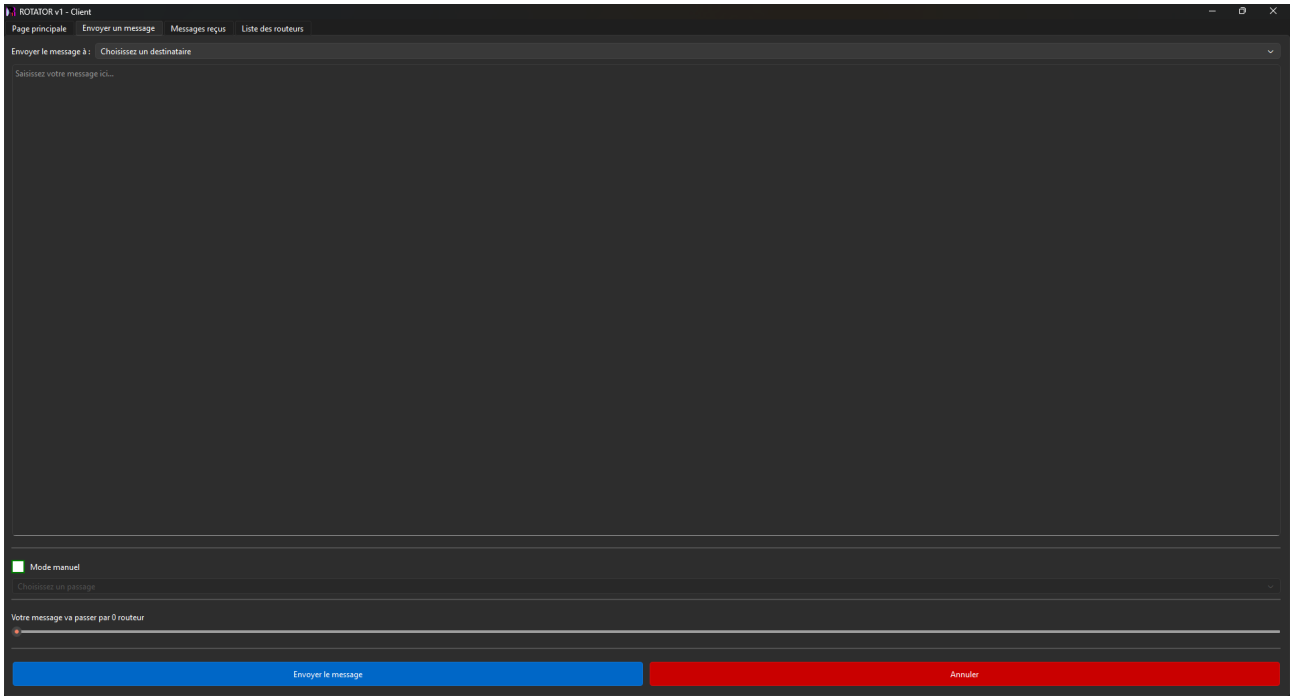


FIGURE 8: PAGE D'ENVOI DE MESSAGE - GUI CLIENT

Sur cette page, le client peut envoyer un message à n’importe quel client connecté et donc cité par le Maître. Le client doit renseigner le destinataire. Il doit ensuite rentrer le message à envoyer. À savoir que le message n’a pas de limite de taille et prend en compte le retour à la ligne. Ensuite, le client a le choix entre l’utilisation d’un mode automatique (par défaut) ou d’un mode manuel (en cochant la case).

Si le client utilise le mode automatique, il doit renseigner le nombre de routeurs par lesquels le message va passer, avant qu’il n’arrive au destinataire. Cela se fait en glissant le curseur vers la gauche ou la droite. Il est possible de choisir aucun routeur, alors le message est chiffré et envoyé directement au destinataire.

Si le client utilise le mode manuel, alors il devra choisir le passage parmi plusieurs, tel que le message ne peut passer que par un ou zéro même routeur.

Le client peut soit envoyer le message à un client, qui peut-être soi ou un autre, soit il peut annuler.

3.2.3. Messages Reçus

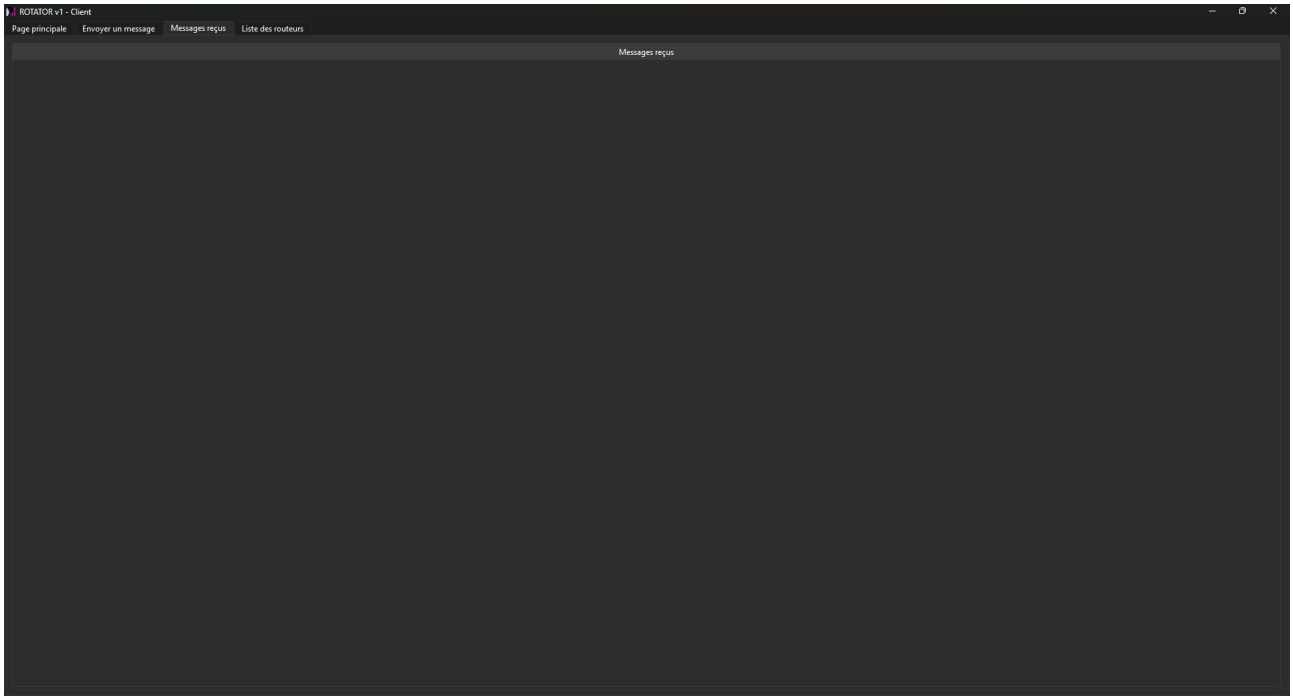


FIGURE 9: MESSAGES REÇUS - GUI CLIENT

Sur cet onglet, le client peut voir tous les messages qu'il a reçus. Ils sont d'ailleurs également enregistrés dans un fichier texte à la racine du fichier python. Cela permet de voir les messages passé et présent. Mais le client n'a aucune idée de l'émetteur du message, ainsi le principe d'anonymat est totalement respecté.

3.2.4. Liste Des Routeurs

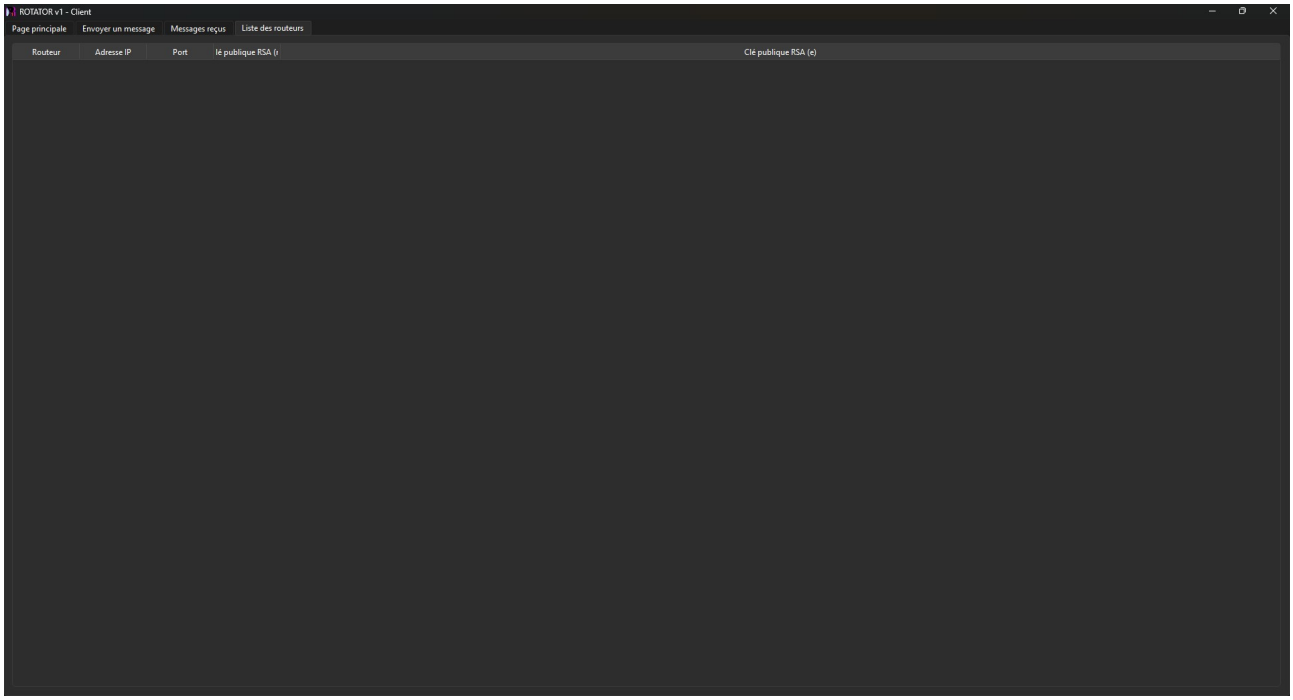


FIGURE 10: LISTE DES ROUTEURS - GUI CLIENT

Après s'être connecté au Maître, voir plus haut, le client peut voir la liste des routeurs présents sur le réseau. La liste contient le nom du routeur, son adresse IPv4, son port et ses clés RSA publiques.

4. BIBLIOTHÈQUE PYTHON

4.1. LISTE DES BIBLIOTHÈQUES UTILISÉS

Bibliothèque	Fonction
socket	Interfaçage entre plusieurs ports
random	Génère des séquences aléatoires.
sympy.isprime	Vérifie si un nombre est premier.
__logo	Affiche le logo ROTATOR.
__gui	Interface graphique du logiciel.
mariadb	Gestion de la base de données
threading	Gestion de plusieurs éléments

4.2. FONCTIONS INTRASÈQUES

Il existe des fonctions du programme qui ne sont dans aucune classe et qui sont accessibles de partout.

4.2.1. Nom

La fonction **nom()** permet de renvoyer le nom de l'ordinateur hôte.

4.2.2. IP

La fonction **ip()** permet de renvoyer l'adresse IPv4 de l'ordinateur hôte.

4.3. A64

Cette classe permet de convertir des entiers en format 64 symboles.

4.3.1. Alphabets

La classe comporte 10 alphabets, utilisant les mêmes caractères, mais dans un ordre différent. Selon l'alphabet « alfa » utilisé, la valeur de sortie du convertisseur n'est pas la même.

Il faut utiliser la fonction **alfa(numero_alphabet)**. Elle prend l'argument numero_alphabet qui doit être un entier tel que $1 \leqslant \text{numero} \leqslant 10$.

4.3.2. Conversion De L'alphabet En Binaire

La fonction **binalfa(numero_alphabet)** permet de convertir un alphabet en dictionnaire, où chaque caractère est associé à une valeur binaire codée sur 6 bits.

Elle prend le même argument que la fonction précédente.

4.3.3. Conversion D'entier En Base 64

4.4. RSA

La classe RSA permet d'utiliser un chiffrage asymétrique, selon l'algorithme de Rivest, Shamir et Adleman.

La classe renvoie les attributs suivants :

- delta (int) : Le delta de la clé privée, tel que $\delta = |p - q|$
- fichier_Km : L'emplacement du fichier de la clé privée
- Km (dict) : Les valeurs de la clé privée
- fichier_Kp : L'emplacement du fichier de la clé publique
- Kp (dict) : Les valeurs de la clé publique

4.4.1. Génération Des Clés

La fonction **clés(nombre_bits)** permet de générer un jeu de clés. Elle prend l'argument « nombre_bits » qui est un entier supérieur ou égal à deux cents. Si la valeur est inférieure à deux cents, alors l'erreur ValueError apparaîtra. Cela permet d'obtenir des clés un minimum sécurisées. À chaque fois que vous utilisez cette fonction, les clés changent.

La **clé publique** est définie par le couple [n, e], fourni sous la forme de la liste Kp. Donc, $n = \mathrm{Kp}[0]$ et $e = \mathrm{Kp}[1]$.

La **clé privée** est définie par le triplet [p, q, d], fourni sous la forme de la liste Km. Donc, $p = \mathrm{Kp}[0]$, $q = \mathrm{Kp}[1]$ et $d = \mathrm{Kp}[2]$.

La génération des clés se fait en utilisant la classe random.SystemRandom(). Cette classe n'utilise pas la Mersenne Twister, ce qui rend la génération des clés plus sécurisée, car SystemRandom est CSPRNG.

4.4.2. Chiffrement

Le programme convertit un mot m en un entier naturel. Cet entier m doit être strictement inférieur à n $m < n$.

L'opération de chiffrement est $Chiffré \equiv Message^{e} \bmod n$.

Si vous chiffrez un message avec RSA, alors la taille maximale est $l_{max} = \frac{(2n) - 1}{8}$. Par exemple, si vous avez choisi **cles(200)**, alors $l_{max} = \frac{(2 \times 200) - 1}{8} = \frac{399}{8} \approx 49$. En testant, j'ai trouvé 50 caractères maximaux.

Pour utiliser les clés DH avec le paramètre premier $p = 2^{255} - 19$, on sait que la longueur des clés sera inférieure ou égale à 77 caractères. En arrondissant à 80 caractères, il nous faudrait générer des clés de $l_{max} \equiv \frac{(2n)}{8} = 80 \Leftrightarrow 2n = 8 \times 80 = 640 \Leftrightarrow n = \frac{640}{2} = 320$.

4.4.3. Déchiffrement

Pour déchiffrer un message c, le programme utilise

L'opération de déchiffrement est $m \equiv c^d \bmod n$.

4.4.4. Vérifier La Signature

Pour vérifier l'intégrité d'un message codé avec RSA, il faut utiliser la fonction **verifier(message_signe, methode_hache, Kp1)**.

Notre messagerie utilise TOR, les routeurs n'ont pas besoin de signer un message quand ils le transmettent à un autre routeur. Cette fonction sert uniquement maître qui vérifie que la clé publique RSA transmise est correcte.

4.4.5. Torage D'un Message

Pour envoyer un message encapsulé, il faut utiliser la fonction **torage(message)**.

4.4.6. Détorage D'un Message

Pour décapsulé un message, il faut utiliser la fonction **detorage(message_chiffre)**.

4.4.7. Sauvegarde Des Clés

Pour ne pas avoir à générer des clés RSA à chaque lancement de programme, la fonction **sauvegarde_cles(nom_fichier, alfa, K)** permet d'enregistrer chaque clé dans un fichier indépendant.

4.4.7.1. Exemple

La clé publique est bien enregistrée en claire.

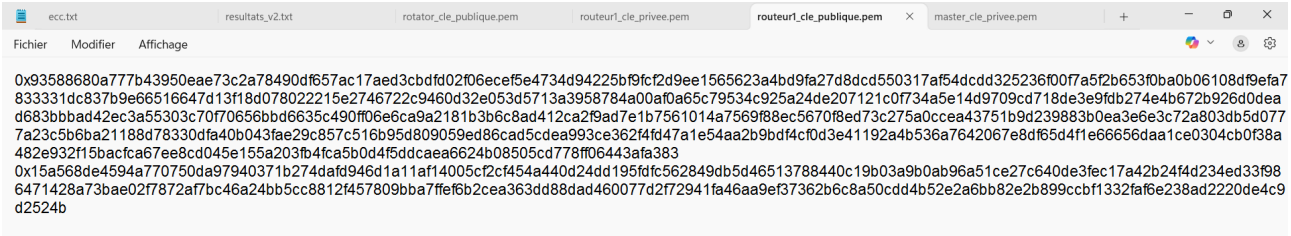


FIGURE 11: EXEMPLE D'ENREGISTREMENT DE CLÉ PUBLIQUE RSA

Tandis que la clé privée a été codée et n'est pas déchiffrable sans la clé K et l'alphabet alfa.

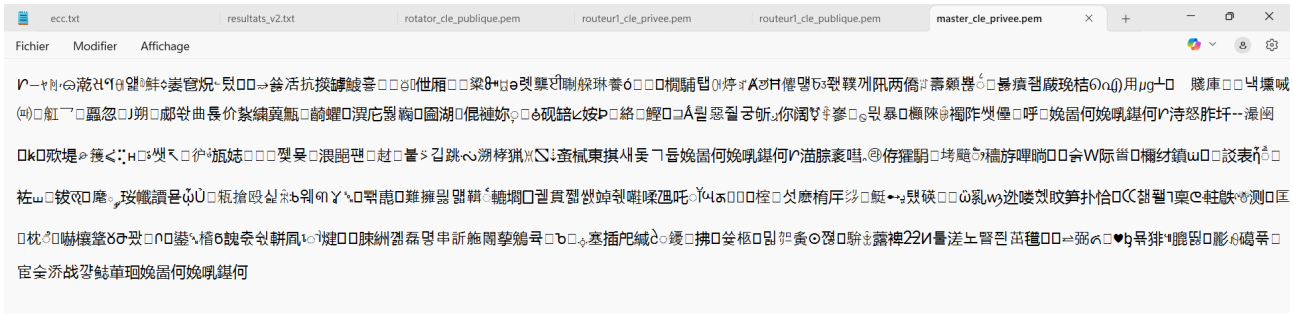


FIGURE 12: EXEMPLE D'ENREGISTREMENT DE CLÉ PRIVÉE RSA

4.4.8. Chargement Des Clés

Après avoir sauvegardé les clés, il faut nécessairement les charger. Cela se fait via la fonction **chargement_cles(nom_fichier, alfa, K)**.

4.5. ECDHE

La classe ECDHE() permet d'utiliser un chiffrement symétrique avec une clé partagée, comme avec DHE, sauf que la classe utilise les courbes elliptiques. Elle permet également de vérifier l'intégrité et l'authenticité d'un message. La classe utilise la courbe Secp256r1.

Les avantages de ECDHE par rapport à DHE sont :

- Les paramètres sont déjà prédéfinis. Donc, on ne perd pas de temps à s'échanger sur des paramètres à choisir. Ces paramètres ont été donnés grâce au SECG[1].
- Les clés sont de taille réduite.

Le paramètre premier pour Secp256r1 est $p = 2^{224}(2^{32}-1)+2^{192}+2^{96}-1$.

Soit **ecdhe = rotator.ECDHE()**.

4.5.1. Génération De La Clé Privée

Pour générer la clé privée Km, il faut utiliser la fonction **cle_privee()**. La clé privée est un entier Km choisis aléatoirement, grâce à **SystemRandom**, tel que $K_m \in [1\,;\,p-1]$.

4.5.2. Génération De La Clé Publique

Pour générer la clé publique Kp, il faut utiliser la fonction **cle_publique()**.

4.5.3. Génération De La Clé Partagée

Pour générer la clé partagée K, il faut utiliser la fonction **cle_partagee()**. Il faut d'abord que vous génériez une clé publique.

4.5.4. Vérifier L'intégrité

Pour vérifier l'intégrité et l'authenticité d'un message, les deux systèmes vont signer le message, en utilisant la fonction **signer(message)**.

La fonction prend l'argument « message », qui n'a pas de limite de taille, mais qui doit être au format « bytes ».

4.5.5. Chiffrement / Déchiffrement D'un Message

Pour chiffrer ou déchiffrer un message, il faut utiliser la fonction commune **chiffrer(message)**. Étant donné que la fonction n'utilise qu'une simple bascule xor, il est fortement déconseillé de chiffrer des données sensibles.

La fonction prend l'argument « message », qui doit être au format « bytes ». L'argument n'a pas de limite de taille, mais le résultat peut être trois à quatre fois plus grand que le message de départ.

La fonction utilise la fonction **flux_sha512(cle, longueur)**, qui génère un flux pseudo-aléatoire à partir de la clé pour le chiffrement XOR.

1 SECG : Standards for Efficient Cryptography Group

5. PHASES DE COMMUNICATION

5.1. PHASE D'INITIALISATION D'UN ROUTEUR

Le routeur désigne un routeur ou un client, puisqu'un client peut être un routeur. Celui-ci se déclare au maître via la procédure d'initialisation. Sur un port spécial, le maître écoute n'importe quel routeur voulant se déclarer.

Pour ce faire, le routeur envoie un certificat :

$$Certificat = [methode\,clé\,, taille\,clé\,, Kp\,, IP\,, port\,, H(Kp + IP + port)]$$

Avec les paramètres suivants :

- méthode clé : RSA, ECDSA
- taille clé (bits)
- Kp : clé publique
- IP : son adresse IP
- port : son port d'écoute
- H(Kp + IP + port).

5.1.1. Réponse Positive

Le Maître vérifie l'intégrité du message avec la fonction vérifier(). Si la signature est correcte, le Maître envoie ACK et sa clé publique au routeur, en chiffrant le message avec la clé publique du routeur. Ce dernier sera le seul à savoir si le certificat est correct ou non.

$$Réponse\ positive = chiff(ACK + Kp_{maître}, Kp_{routeur})$$

5.1.2. Réponse Négative

Si la réponse est NACK, alors il y a eu un problème. Le maître peut envoyer le problème qu'il a rencontré. La réponse sera $Réponse\,négative = chiff(NACK + message, Kp_{routeur})$, pour tous les problèmes non intrasèques à la clé privée du routeur. Sinon $Réponse\,négative = signer(NACK)$ et le maître ferme le port d'écoute.

Si toutefois l'erreur vient du fait que l'IP et le port du routeur sont déjà identiques à un autre, alors le message demandera de changer le port du routeur, avant de trouver un consensus.

Index des figures