Dans l'archive envoyé avec ce rapport, il y'a les éléments ci-dessous, listés dans l'ordre de leur utilisation :

- PGP_certificate_request.c :
  - **Description :** Ce programme permet de créer un certificat à partir du nom de la personne(Premier paramètre) et le chemin du fichier contenant la clé publique associée (Deuxième paramètre).
  - **Exemple :** ./request rokia ./client.pub où « rokia » est le nom du client, « ./client.pub » est le chemin du fichier contentant la clé publique du client, et « request » est le nom de l'exécutable. On suppose dans cet exemple que le fichier se trouve dans le même dossier que l'exécutable.
  - **Résultat :** L'archive contient la clé publique « client.pub », donc l'exécution de la commande donnée dans l'exemple produit un fichier nommé « rokia.cert » qui contient le certificat associée à cette clé publique. Le certificat contient l'identité du client et sa clé publique, ainsi que le haché de la clé publique, signé avec la clé privée de l'autorité qui se trouve dans le fichier « private_autority.key » donné également dans l'archive.
- PGP_certificate_verification.c :
  - **Description :** Ce programme permet de vérifier la validité d'un certificat à partir de l'identité du client (Premier paramètre) , du chemin du fichier contenant le certificat du client (Deuxième paramètre), et du chemin du fichier contenant la clé publique associée (Troisème paramètre).
  - **Exemple :** **./verify rokia ./rokia.cert ./client.pub** où « rokia » est le nom du client, « ./rokia.cert » est le chemin du certificat du client, « ./client.pub » est le chemin du fichier contentant la clé publique du client, et « verify » est le nom de l'exécutable. On suppose dans cet exemple que les fichiers se trouvent dans le même dossier que l'exécutable.
  - **Résultat :** L'archive contient la clé publique « client.pub » et le certificat « rokia.cert ». Ainsi, ce programme utilise la clé publique de l'autorité contenue dans le fichier « public_autority.key » pour vérifier la signature contenue dans le certificat « rokia.cert » en la comparant à la clé publique « client.pub ». Le programme répond dans cet exemple par la phrase «La clé publique que vous avez correspond bien à celle de rokia » si la vérification est réussie. Il retourne rien dans le cas échéant.
- PGP_AES_encryption.c :
  - **Description :** Ce programme permet, à l'instar du protocole PGP, de chiffrer un fichier en utilisant une clé AES. En effet, en plus du chemin du fichier à chiffrer (Premier paramètre), il faut préciser le chemin du fichier contenant la clé publique (Deuxième paramètre) du destinataire.
  - **Exemple :** **./encrypt ./test1.txt ./client.pub** où « encrypt » est le nom de l'exécutable, « ./test1.txt » est le chemin du fichier à chiffrer, et « ./client.pub » est le chemin du fichier contentant la clé publique du

destinataire. On suppose dans cet exemple que les fichiers se trouvent dans le même dossier que l'exécutable.

- **Résultat :** L'exécution de ce programme produit le fichier chiffré « test1.enc » ainsi que le fichier « aes.key ». En effet, le programme construit d'une manière aléatoire une clé AES et la chiffre avec la clé publique du destinataire « client.pub ». La clé AES chiffrée est mise dans le fichier « aes.key ». Par ailleurs, le fichier « ./test1.txt » est chiffré avec la clé AES avant qu'elle soit chiffrée avec la clé publique. Le résultat de chiffrement est mis dans le fichier « test1.enc ». La fonction openssl utilisée pour générer aléatoirement la clé AES est *RAND_file_name*. Pour que le programme fonctionne correctement il faut définir la variable d'environnement $HOME ,si elle n'existe pas déjà, pour pouvoir utiliser le fichier $HOME/.rnd. Si le système utilise un autre fichier qui produit des données aléatoires, il faut le renseigner dans la variable d'environnement $RANDFILE.

- PGP_AES_decryption.c :
  - **Description :** Ce programme permet, à l'instar du protocole PGP, de déchiffrer un fichier en utilisant une clé AES chiffrée avec une clé RSA. Le programme prend en entrée 1) le chemin du fichier à déchiffrer, 2) le chemin du fichier contenant la clé privée du client, et 3) le chemin du fichier contenant le chiffré de la clé AES.
  - **Exemple : ./decrypt ./test1.enc ./client.priv ./aes.key** où « decrypt » est le nom de l'exécutable, « ./test1.enc » est le chemin du fichier à déchiffrer, « ./client.priv » est le chemin du fichier contenant la clé privée du client, et « ./aes.key » est le chemin du fichier contenant le chiffré de la clé AES qui a été utilisée pour chiffrer le fichier. On suppose dans cet exemple aussi que les fichiers se trouvent dans le même dossier que l'exécutable.
  - **Résultat :** L'exécution de ce programme produit le fichier « test1_D.txt ». En effet, le programme utilise la clé privée «client.priv » pour déchiffrer le fichier « aes.key » et obtenir la clé AES. Ensuite, il utilise la clé AES pour déchiffrer le fichier « ./test1.enc » et il met le résultat dans le fichier « test1_D.txt » qui doit donc avoir le même contenu que le fichier original « ./test1.txt ».

Vous trouvez également dans l'archive des fichiers utiles pour le test des programmes :

- « client.priv » et « client.pub » sont les fichiers contenant respectivement la clé privée et la clé publique du client.
- « private_autority.key » et « public_autority.key » sont les fichiers contenant respectivement la clé privée et la clé publique de l'autorité de certification.
- le fichier « test1.txt » est un exemple de fichier utilisé pour tester le chiffrement.

Remarque : Pour obtenir les exécutables, il faudrait compiler les programmes en utilisant la commande : gcc prog.c -lssl -lcrypto -o prog; où prog désigne le nom du programme à compiler.