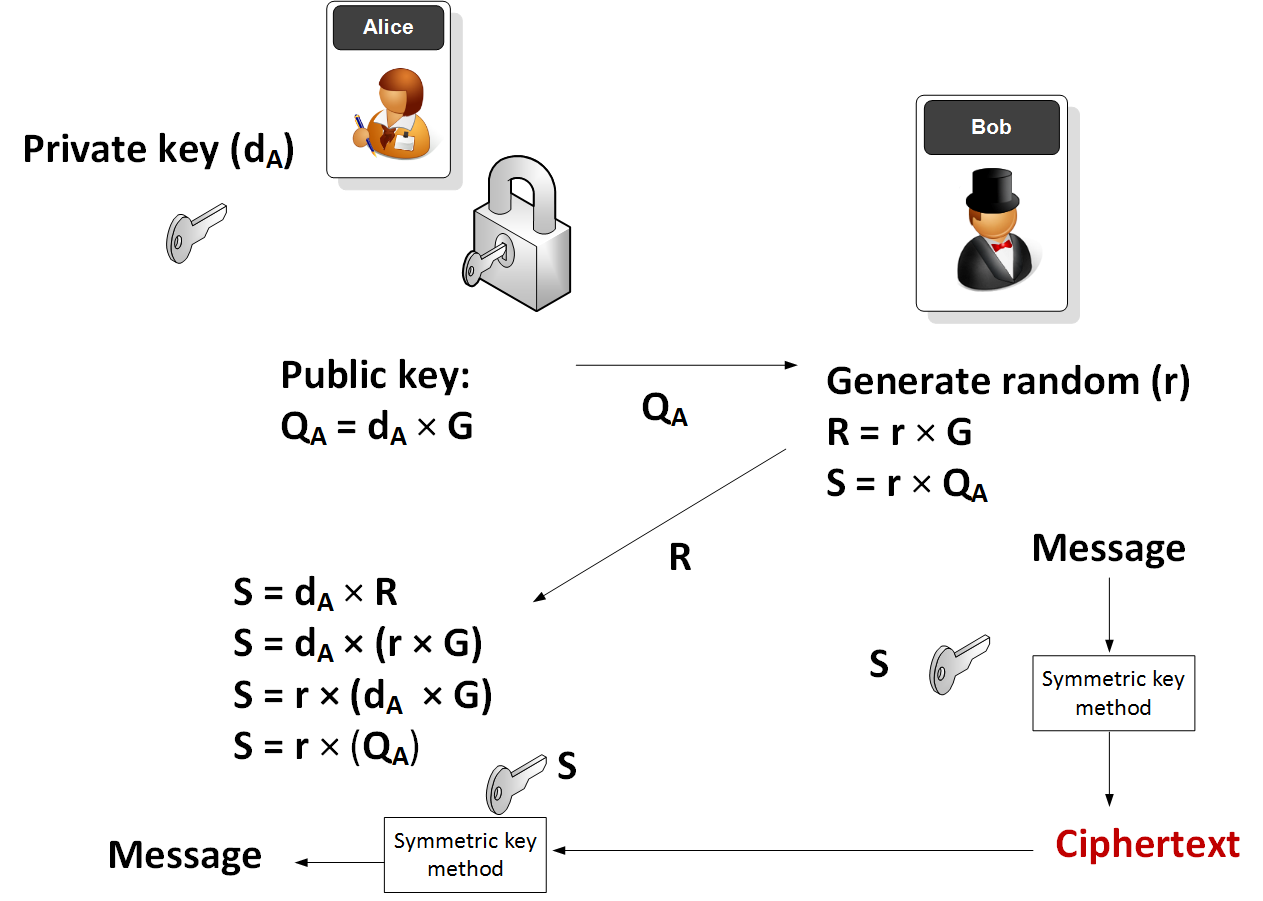
**Documentation Chiffrage\_component**



Étudiant : Floryan KIEFFER

Enseignant : Jose LUU

Université de Paris Dauphine



**Introduction :**

Le composant chiffrage\_component permet d’encrypter un message en utilisant ECIES. C’est un composant codé en C++ utilisable en python. Elliptic Curve Integrated Encryption Scheme, ou ECIES, est un système de chiffrement hybride proposé par Victor Shoup en 2001. La cryptographie à courbe elliptique (ECC) peut être utilisée comme un outil pour chiffrer des données, créer des signatures numériques ou effectuer des échanges de clés. En ce qui concerne la procédure de chiffrement, les schémas actuellement utilisés sont appelés cryptosystèmes hybrides, car ils utilisent à la fois des techniques symétriques et asymétriques. Parmi ces cryptosystèmes hybrides basés sur ECC, le schéma de chiffrement intégré à courbe elliptique (ECIES) est le plus connu et, à ce titre, on le retrouve dans plusieurs normes cryptographiques.

**Spécification :**

1. Le composant chiffrage\_component utilise deux submodules :
   * CryptoPP qui est une librairie de cryptographie
   * Pybind11 qui permet l’interprétation du code C++ en python
2. Le composant chiffrage\_component est composé :
   * D’un fichier Makefile qui construit le composant
   * D’un fichier chiffrage\_component.cpp qui contient le code c++ du composant
   * D’un fichier chiffrage\_component.h qui contient les en-têtes des méthodes du composant
   * Des submodules CryptoPP et pybind11
3. La classe Chiffrage est composé de deux attributs de type String :
   * String plaintext 🡺message non crypté ou message décrypté
   * String encryptedtext 🡺Message encrypté
4. La classe Chiffrage est composé de 8 Méthodes :
   * Chiffrage() 🡺Constructeur
   * ~Chiffrage() 🡺Desctructeur
   * const std::string &getPlaintext() const 🡺 retourne la référence vers le message non crypté/décrypté
   * const std::string &getEncryptedText() const 🡺 retourne la reference vers le message encrypté
   * void SavePrivateKey(const CryptoPP::PrivateKey& key, const std::string& file) 🡺 Sauve la clé privé passé en paramètre dans un fichier donc le chemin est également passé en paramètre
   * void SavePublicKey(const CryptoPP::PublicKey& key, const std::string& file) 🡺 Sauve la clé public passé en paramètre dans un fichier donc le chemin est également passé en paramètre
   * void LoadPrivateKey(CryptoPP::PrivateKey& key, const std::string& file) 🡺 Récupère une clé privée dans un fichier dont le chemin est passé en paramètre et la charge dans la clé privée dont la paramètre est passé par référence
   * void LoadPublicKey(CryptoPP::PublicKey& key, const std::string& file) 🡺 Récupère une clé publique dans un fichier dont le chemin est passé en paramètre et la charge dans la clé publique dont la paramètre est passé par référence
   * void encrypt\_decrypt(std::string em) 🡺 Encrypte le message passé en paramètre et le stock dans l’attribut encryptedtext.

**Comment fonctionne la méthode principale qui va encrypter le message ?**

Le composant chiffrage\_component a pour objectif d’encrypter un message à partir d’une clé publique fournit par le décrypteur. Le décrypteur est un objet qui se construit sur la base d’une courbe ECIES et qui va générer une clé privée. Sur la base de cette clé privée il va générer une clé publique qu’il va stocker dans un fichier (par exemple : ECIES\_publicKey.key). C’est à partir de ce moment que le composant chiffrage\_component intervient avec la fonction encrypt\_decrypt qui prend en paramètre le message à encrypter. Cette fonction va créer un objet Encryptor de la libraire CryptoPP en récupérant la clé publique chargé en amont par le Decryptor (ou décrypteur) en utilisant la fonction LoadPublicKey qui prend en paramètre la référence d’une clé publique et le chemin vers le fichier contenant la clé publique sauvegarder par le Decryptor. L’objet Encryptor va ensuite vérifier que la clé publique récupérer dans le fichier est valide en utilisant les fonctions GetPublicKey().ThrowIfInvalid(prng, 3) appelé depuis l’objet Encryptor. Si la clé publique n’est pas valide une exception est retournée sinon le message est encrypté avec l’aide de l’objet StringSource de la libraire CryptoPP. Une fois le message encrypté il est afficher à l’écran sous 3 formes différentes : le message encrypté UTF\_8, le message encrypté en hexadécimale et le message encrypter par HexEncoder de la libraire CryptoPP.

**Comment utiliser le composant ?**

Dans un premier temps il vous faire un clone du projet dans un dossier sur votre machine à l’aide de la commande git clone suivi de l’URL du projet git disponible sur Github.  
  
Une fois le projet cloné :  
  
1) Réaliser la commande **cd projet\_blochain\_python**  
  
2) Pour exécuter ce projet vous devez dans un premier temps réaliser un **./add\_submodules.sh** pour ajouter les submodules nécessaires au projet

3) Ensuite vous devrez réaliser la commande **git submodule init** pour initialiser les submodule et pouvoir les utiliser par la suite

4) Réaliser la commande **git submodule update**

5) Rendez-vous dans le composant à l’aide de la commande **cd chiffrage\_component**

6) Une fois dans le dossier du composant vous pouvez y voir tous les submodules et il vous faut vous rendre dans le submodule cryptopp par le biais de la commande **cd cryptopp**

7) Nous allons créer la libraire cryptopp que nous utilisons pour réaliser le chiffrage. Taper la commande **make** et attendre la création du fichier .a

8) Une fois la librairie crée faite un **cd ..** pour retourner dans le dossier chiffrage\_component et taper la commande **make** afin de crée notre composant

9) maintenant tout est prêt il nous fait donc lancer python avec la commande **python3**

10) Un fois python lancé taper les commandes suivantes :  
>> import chiffrage\_component  
>>chiffrage = chiffrage\_component.Chiffrage()  
>>chiffrage.encrypt\_decrypt(« votreMessage »)

**Entrée et sortie**

**Entrée :** Message à encrypter

**Sortie :** Message encrypté

**Test**

Pour la partie test étant donné que chiffrage\_composant reçoit la clé publique du décrypteur nous avons réalisé les tests dans le composant du décrypteur. Les tests portent sur la clé publique généré par le décrypteur que nous allons comparer avec la clé publique généré par la librairie eciespy. Pour générer la clé publique avec la librairie eciespy nous utilisons l’exposant privée généré par la librairie CryptoPP. Si les clés puliques générées par la librairie eciespy et CryptoPP sont les mêmes les tests seront valides.

CryptoPP :

Une image contenant texte

Description générée automatiquement

Eciespy :



Une image contenant texte

Description générée automatiquement

Nous ne sommes pas parvenus à tester la génération d’un message encrypté avec CryptoPP et de le déchiffrer avec eciespy (test.py) et nous en concluons que la façon d’encrypter des deux librairies n’est pas la même car ecies n’accepte pas le message crypter de CryptoPP (en hex).

**Communication des composants chiffrage et déchiffrage**

Les deux composants chiffrage et déchiffrage communiquent correctement ensemble. Nous avons mis en place un code en python qui permet de les faire communiquer. En préambule il faut créer un dossier dans lequel il faudra cloner le projet déchiffrage. Voici l’enchainement des commandes à réaliser pour lancer fichier communication python des deux composants.

**git clone** [**https://github.com/jeremysellem/projet\_blockchain\_python.git**](https://github.com/jeremysellem/projet_blockchain_python.git)

**cd projet\_blockchain\_python/**

**chmod 777 add\_submodules.sh**

**sudo ./add\_submodules.sh**

**git submodule init**

**git submodule update**

**cd comp\_decrypt\_ecies/cryptopp/**

**make**

**cd ..**

**make**

**python3 test.py**

**cd ..**

**python3 compo\_test.py**