TP 3 : Signature ElGamal

Rémy Cassini, Théo Pirkl, Jérôme Chételat
Noria Foukia, Eryk Schiller

Introduction

ElGamal est un protocole de cryptographie asymétrique qui porte le nom de son inventeur, Taher Elgamal. Il a été conçu en 1984 et, bien que rarement utilisé tel quel aujourd'hui, ses méthodes ont été appliquées à DSA qui est devenu ECDSA qui, lui, est largement utilisé (par exemple pour les Bitcoin).

Mathématiquement, ElGamal repose sur le problème des logarithmes discrets. Dans un groupe cyclique multiplicatif $\mathbb{Z}_p^*$, il est très facile de prendre un élément a et de l'élever à une puissance n. Toutefois, si l'on vous donne n et le groupe $\mathbb{Z}_p^*$, il est très difficile de retrouver a en général. C'est-à-dire, il est difficile de faire l'opération "logarithme" dans ce contexte, afin de retrouver la base a.

Enfin, ElGamal est un protocole qui comprend une méthode de chiffrement et une méthode de signature : c'est cette dernière qui va nous occuper pendant ce TP. Lorsque je signe un fichier, j'utilise ma clef privée et le contenu du fichier pour produire une signature (que j'ajoute généralement au fichier). Pour vérifier ma signature, le destinataire devra utiliser ma clef publique.

Environnement

Vous utiliserez Python ainsi que votre éditeur de texte ou IDE préféré. Nous vous fournissons dans une archive *.zip* deux fichiers nommés : `elgamal.py` et `elgamal_tests.py`.

`elgamal.py`, contient votre squelette (spooky). Ne changez pas le nom des fonctions déjà présentes !

`elgamal_tests.py`, est la batterie de tests, ne modifiez **rien** dedans !

La commande pour lancer les tests est :
`python3 -m unittest elgamal_tests.ElGamalTestCase --verbose`

Évaluation

Ce TP durera **deux séances**. Il est **évalué** et devra donc figurer dans votre journal de laboratoire. Le code compte pour **2/3** et le journal pour **1/3**.

Votre code doit être un minimum commenté et vous devez être capable de l'expliquer à l'oral si nous avons des doutes sur son origine.

Nous vous rappelons qu'il est **formellement interdit** d'utiliser une IA générative (ChatGPT et autres) pour générer votre code.

Exercices

Vous allez implémenter votre propre algorithme ElGamal en suivant les étapes de ce TP, ainsi que les instructions données à l'oral.

Avant toute chose, reprennez votre **TP1** et effectuez la partie 3 si ce n'est pas déjà fait. Revenez ici une fois que vous aurez terminé.

Assurez-vous que votre fichier `ssi_lib.py` (ou une copie de celui-ci) est bien dans le même répertoire que votre fichier `elgamal.py`, car vous devez importer les fonctions utiles du premier dans le second.

Le fonctionnement entier de l'algorithme vous est expliqué à l'intérieur même de votre fichier `elgamal.py`, dans la fonction `main()`. Lisez ces instructions attentivement, car elles décrivent réellement les étapes que vous devrez suivre, avec certains conseils d'implémentation que vous pourrez retrouver dans des fonctions plus bas dans le fichier.

Pour la "théorie" d'ElGamal lui-même, nous la divisons en trois parties :

1. Génération des clefs

Avant tout chiffrement ou toute signature, un protocole de cryptographie asymétrique doit générer deux clefs : une publique (distribuée à tout le monde, y compris vous) et une privée (gardée secrète à tout prix).

Dans ElGamal, on fait comme suit :

1. Choisir un nombre premier p (pour vous, entre 1 et 10000), ce nombre sert à choisir notre groupe cyclique multiplicatif $\mathbb{Z}_p^*$.
2. Trouver un nombre g, générateur de $\mathbb{Z}_p^*$.
3. Choisir un nombre a tel que $0 \leq a < p - 1$.
4. Calculer $A = g^a \mod p$

   Votre clef publique est le triplet de nombres (p, g, A) et votre clef privée est le nombre a.

2. Signature

Lorsque Alice utilise utilise un algorithme de signature, elle utilise **sa clef privée** pour signer son message.

Avec ElGamal, la signature est un couple de deux entiers (Y, S) qui sont définis comme suit (pour m, voir le paragraphe Précisions ci-dessous) :

$$Y \equiv g^k \mod p \qquad S \equiv (m - aY)k^{-1} \mod (p-1)$$

3. Vérification de la signature

Lorsque Bob reçoit un message signé de la part d'Alice, il utilise **la clef publique d'Alice** pour vérifier que le message est bien authentique.

Pour se faire, il doit vérifier l'égalité suivante (ici aussi, pour m, voir Précisions) :

$$A^Y \cdot Y^S \equiv g^m \mod p$$

Précisions

Vous vous posez peut-être la question de ce que fait m, c'est-à-dire le message, dans une équation mathématique. En réalité, il faut "transformer" votre message (qui contient donc du texte) en un nombre m qui pourra être utilisé mathématiquement.

Pour ce faire, on utilise normalement une **fonction de hachage**. Cependant, vous n'avez pas vu ce chapitre pour le moment. Nous avons pris la décision de faire l'impasse pour ce TP : vous utiliserez une fonction qui n'est **pas** une fonction de hachage, mais qui possède les propriétés qui nous intéressent pour faire fonctionner ElGamal (décrite dans le fichier `elgamal.py` directement).

Attention !

Lorsque vous lancez vos tests avec la commande qui vous a été fournie plus haut, cela va régénérer vos clefs publique et privée (les fichiers `key` et `key.pub`) afin de tester votre implémentation correctement, puis les supprimer. Ne soyez donc pas surpris si ces fichiers disparaissent : ce n'est pas grave, votre code est censé en régénérer s'ils sont absents de toute manière.

Questions théoriques

Les réponses aux questions suivantes devront apparaître dans votre journal de laboratoire :

1. Quel est l'ensemble généré par g ?
2. Expliquez pourquoi ces bornes pour le choix de a.
3. Expliquez le choix de k que vous avez dû implémenter dans votre code.
4. Avec une même paire de clefs (fichiers `key` et `key.pub`) ainsi qu'un même message (`message.txt`), est-ce que votre signature sera toujours la même si vous lancez plusieurs fois votre code ? Expliquez pourquoi.