HAI709I - TD 10

Rocco Mora

1 Décembre, 2025

Exercice 1 :

Montrez que la construction qui permet de transformer un schéma de chiffrement à clé publique CPA-sûr pour un message à un seul bit/longueur fixe en messages de longueur arbitraire ne fonctionne pas dans le cas de la sécurité CCA, ce qui signifie que même si Π est CCA-sûr, Π' ne l'est pas.

Exercice 2 :

Considérons le schéma de chiffrement à clé publique suivant. La clé publique est $(\mathbb{G}, q, g, h)$ et la clé privée est x, générée exactement comme dans le schéma de chiffrement El Gamal. Afin de chiffrer un bit b, l'expéditeur procède comme suit :

- Si $b = 0$, choisir $y \in \mathbb{Z}_q$ uniforme et calculer $c_1 := g^y$ et $c_2 := h^y$. Le texte chiffré est (c_1, c_2).
- Si $b = 1$, choisir des $y, z \in Z_q$ uniformes indépendants, calculer $c_1 := g^y$ et $c_2 := g^z$. Le texte chiffré est (c_1, c_2).

Montrer qu'il est possible de déchiffrer efficacement étant donné x. Prouver de manière informelle que ce schéma de chiffrement est CPA-sûr si le problème décisionnel de Diffie-Hellman est difficile par rapport à $\mathbb{G}$.

Exercice 3 :

GenRSA produit $(N, e, d) = (55, 3, 27)$. Vérifiez qu'il s'agit d'un triplet valide, écrivez la clé publique et la clé privée et chiffrez le message $m = 6 \in \mathbb{Z}^*_{55}$.

Exercice 4 :

Montrez que RSA est malléable, c'est-à-dire qu'un adversaire peut construire un autre texte chiffré c' à partir de c dont le déchiffrement m' est une transformation connue de m, sans connaître m. Par exemple, obtenez c' tel que $m' = 2m \mod N$.

Exercise 5 :

Considérons le schéma de chiffrement El Gamal pour le groupe $\mathbb{Z}^*_{23}$. Le générateur est $g = 5$ et la clé secrète $x = 6$. Déterminez la clé publique, chiffrez $m = 9$ avec $y = 2$, puis déchiffrez le texte chiffré obtenu et vérifiez la propriété de correction.

Exercise sur ordinateur :

Nous allons chiffrer un message avec RSA.

1. Étape 1 : installez les paquets `pycryptodome`.
2. Étape 2 : complétez le script suivant

```
from Crypto.PublicKey import RSA
from Crypto.Cipher import PKCS1_OAEP
from Crypto.Random import get_random_bytes


# ------------------------------------
# Étape 1 : Générer une paire de clés RSA avec un module de 2048 bits.
# ------------------------------------
private_key =
public_key =


# ------------------------------------
# Étape 2 : Chiffrer avec la clé publique à l'aide de PKCS1_OAEP
# ------------------------------------
message = b"Factorization is pretty hard"

ciphertext =

# ------------------------------------
# Étape 3 : Décrypter avec la clé privée et vérifier que message = plaintext.
# ------------------------------------

plaintext =
```