Université Paris 8 J. Lavauzelle

Master 1 Mathématiques et applications — année 2022–23

Cryptographie à clé publique – Devoir 3

07/04/2023

Consignes :

1. à rendre par email avant le vendredi 28/04/2023 ;
2. le code doit être commenté ;
3. il est conseillé d'utiliser `python` et sa bibliothèque externe `cryptodome` si besoin, mais tout autre langage (standard) est accepté. Pour l'installation de `cryptodome`, suivre : `https://pycryptodome.readthedocs.io/en/latest/src/installation.html`

Exercice 1. Implantation de l'IBE de Cocks.

Un chiffrement basé sur l'identité (*identity-based encryption*, IBE) est un chiffrement dans lequel on souhaite utiliser l'**identité d'un utilisateur comme clé publique** (par exemple, son adresse email).

Les avantages sont que l'expéditeur du message peut s'implement utiliser l'identité publique de son destinataire. Cela évite également la prédistribution des clefs publiques (pas de PKI). En revanche, l'inconvénient est qu'une autorité centrale, possédant des **clés maîtresses**, doit délivrer les clés privées utiles au déchiffrement.

En 2001, Cocks propose un IBE fondé sur le problème de la résiduosité quadratique ; c'est le sujet de cet exercice. Avant cela, donnons les définitions formelles d'un IBE :

Définition. Un schéma de **chiffrement basé sur l'identité** (*identity-based encryption*, IBE) est constitué de 4 algorithmes :

1. Un algorithme de génération de **clés maîtresses** $\texttt{MasterKeyGen}()$, qui engendre $\texttt{mpk}/\texttt{msk}$, où la clé maîtresse privée $\texttt{msk}$ est détenue par un tiers de confiance.
2. Un algorithme de génération de **clés personnelles** $\texttt{KeyGen}(\texttt{msk}, \texttt{id}_U)$ qui produit une paire de clés $\texttt{pk}_U/\texttt{sk}_U$. À chaque utilisateur U vont être associées une clé publique $\texttt{pk}_U$ dérivée de son identité $\texttt{id}_U$, et une privée $\texttt{sk}_U$.
3. Un algorithme de chiffrement $\mathsf{Encrypt}(\texttt{pk}_U, \boldsymbol{m})$, avec un fonctionnement similaire au chiffrement à clef publique « classique »,
4. Un algorithme de dechiffrement $\mathsf{Decrypt}(\texttt{sk}_U, \boldsymbol{c})$, avec un fonctionnement similaire au déchiffrement à clef publique « classique ».

Notations. Pour rappel, $\mathrm{QR}(n)$ représente l'ensemble des résidus quadratiques modulo n, et $\mathrm{QR}^*(n)$ l'ensemble des pseudo-résidus modulo n. On note $\mathcal{Q}_n = \mathrm{QR}(n) \cup \mathrm{QR}^*(n)$. On se donne également $H_n : \{0,1\}^* \to \mathcal{Q}_n$ une fonction de hachage, dont la description est donnée plus loin dans l'énoncé. On note enfin $\left(\frac{a}{n}\right)$ le symbole de Jacobi entre a et n.

Pour une taille d'entier $t \geq 1$ fixée, on rappelle que l'IBE de Cocks est décrit comme suit.

Clés maîtresses. La clé publique maîtresse est $\mathtt{mpk} = n$ et la clé privée maîtresse est $\mathtt{msk} = (p,q)$, où p et q sont deux grands nombres premiers distincts de t bits, congrus à 3 modulo 4.

Génération de clés pour l'utilisateur U d'identité $\mathtt{id}_U$.

1. La clé publique $\mathtt{pk}_U$ est $H_n(\mathtt{id}_U)$,
2. La clé privée $\mathtt{sk}_U$ est une racine carrée de $\begin{cases} \mathtt{pk}_U & \text{si } \mathtt{pk}_U \in \mathrm{QR}(n)\,, \\ -\mathtt{pk}_U & \text{si } \mathtt{pk}_U \in \mathrm{QR}^*(n)\,. \end{cases}$

L'espace des clairs est $\mathcal{M} = \{-1,1\}$, et celui des chiffrés est $\mathcal{C} = (\mathbb{Z}/n\mathbb{Z})^2$.

Chiffrement. Si l'on souhaite chiffrer un élément $m \in \{-1,1\}$:

1. Choisir aléatoirement $t_1, t_2 \in \mathbb{Z}/n\mathbb{Z}$, tels que $\left(\frac{t_1}{n}\right) = \left(\frac{t_2}{n}\right) = m$.
2. Calculer
$$\begin{cases} y_1 &= t_1 + \mathtt{pk}_U \cdot t_1^{-1} \mod n \\ y_2 &= t_2 - \mathtt{pk}_U \cdot t_2^{-1} \mod n \end{cases}$$
3. Le chiffré est $y = (y_1, y_2)$.

Déchiffrement. On veut déchiffrer $y = (y_1,y_2) \in (\mathbb{Z}/n\mathbb{Z})^2$:

1. Si $\mathtt{pk}_U = (\mathtt{sk}_U)^2$, alors définir $s = y_1$. Sinon définit $s = y_2$.
2. Calculer et retourner
$$m' = \left(\frac{s + 2\mathtt{sk}_U}{n}\right)$$

Pour vous aider dans cet exercice, vous trouverez dans la correction du TD3, exercice 5, une fonction calculant le symbole de Jacobi $\left(\frac{a}{n}\right)$, où n est un nombre composé dont la factorisation est inconnue, et où a est un entier quelconque.

Donnons maintenant la description de la fonctionde hachage $H_n : \{0,1\}^* \to \mathcal{Q}_n$. Pour cela, on considère la fonction de hachage standardisée `SHA3-224`, que l'on trouvera par exemple dans la bibliothèque `cryptodome` pour le langage `python`. Étant donné un texte $\boldsymbol{m}$, on calcule alors $H_n(\boldsymbol{m}) \in \mathcal{Q}_n$ par l'algorithme suivant :

1. Calculer $r = \lfloor \log_2(n)/224 \rfloor$.
2. Calculer
$$\boldsymbol{h} = \mathtt{SHA3\_224}(\boldsymbol{m} \,\|\, 0) \,\|\, \mathtt{SHA3\_224}(\boldsymbol{m} \,\|\, 1) \,\|\, \cdots\cdots \,\|\, \mathtt{SHA3\_224}(\boldsymbol{m} \,\|\, r)$$
où « $\boldsymbol{m} \,\|\, 0$ » représente la concaténation du texte $\boldsymbol{m}$ avec l'entier 0.
3. Calculer x l'entier associé à $\boldsymbol{h}$, où les bits de poids faible de x sont les bits à droite de $\boldsymbol{h}$.
4. Réduire x modulo n.
5. Si $\left(\frac{x}{n}\right) \neq 1$, revenir à l'étape 2 avec $r \leftarrow r + 1$.
6. Retourner $\boldsymbol{h}$.

Aide (en python).

— On pourra utiliser la bibliothèque `pycryptodome` pour accéder à la fonction de hachage `SHA3_224`, voir la documentation ici : `[lien]`. Cette fonction prend en entrée une séquence d'octets.

— On peut également convertir une chaine de caractères m (encodée en `utf-8`) en une chaîne d'octets par la fonction `bytes(m, 'utf-8')`. Ainsi, pour transformer une chaîne `m` en haché `h` sous forme hexadécimale, grâce à la fonction de hachage `SHA3_224`, on aura le code python suivant :

```
from Cryptodome.Hash import SHA3_224
h = SHA3_224.new(bytes(m, "utf-8")).hexdigest()
```

— Il est également possible d'utiliser la bibliothèque `hashlib`.

— Pour lire le contenu d'un fichier, il existe les fonction `open()` et `read()`. Exemple d'utilisation :

```
# pour ouvrir l'objet fichier
f = open("nom_du_fichier.txt", 'r')
# pour lire le contenu  et le stocker dans une chaine de caracteres
s = f.read()
# pour fermer le fichier (important)
f.close()
```

— Pour transformer un nombre `a` en entier sous forme hexadécimale, on peut utiliser `int(a, 16)`.

Question 1.– Implanter la fonction H_n. On pourra vérifier que pour le message $\boldsymbol{m} =$ `alice@mail.com` et l'entier $n = 473821$, on a :

$$\boldsymbol{h} = \texttt{8800fb998c8eb49a863c3fc46c511c960010375fbe8cfd2c1a6558d8}$$

puis

$$H_n(\boldsymbol{m}) = 154387\,.$$

Question 2.– Implanter la fonction de génération des clefs d'un utilisateur. Cette fonction prendra notamment en entrée un texte (par exemple, une adresse email), ainsi que la clé privée maîtresse. On rappelle que lorsque p est congru à 3 modulo 4, une racine carrée de x modulo p est donnée par $x^{(p+1)/4} \mod p$.

Question 3.– Implanter les fonctions de chiffrement et de déchiffrement du schéma de Cocks.

Question 4.– Chiffrer le message $m = -1$ pour l'utilisatrice d'identité `alice@mail.com`, lorsque la clé maîtresse est

$$n = 473821$$

On souhaite maintenant chiffrer et déchiffrer des textes. Pour cela, on considère que chaque caractère est encodé en un octet via le code ASCII étendu (norme ISO-8859-1). Cet octet correspond à une suite d'exactement 8 bits, le bit de poids faible étant situé à droite.

Par exemple, pour le caractère `A` de code ASCII étendu 65, les 8 bits correspondants sont 01000001. Pour le caractère `é` de code ASCII étendu 233, les bits correspondants sont 11101001.

Question 5.–

1. Écrire une fonction qui convertit un caractère c codé en ASCII étendu, en une suite de 8 bits correspondant à son code ASCII étendu en binaire. Avec le langage `python`, on pourra s'intéresser aux fonctions `bin()`, `ord()` et `chr()`.
2. En déduire une fonction qui convertit un texte constitué de N caractères, en une suite de $8N$ bits.
3. Écrire également la fonction inverse, c'est-à-dire la fonction qui convertit une suite de $8N$ bits en un texte de N caractères.

On vérifiera que la suite de bits associée à `Hello!` est :

$$010010000110010101101100011011000110111100100001$$

Dans le dossier compressé

`www.math.univ-paris13.fr/~lavauzelle/teaching/2022-23/docs/CP/devoir/devoir3-aux.zip`

vous trouverez :

1. un fichier `mpk.txt` qui contient la clé publique maîtresse,
2. un fichier `sku.txt` qui contient la clé privée de l'utilisateur,
3. un fichier `pku.txt` qui contient la clé publique du même utilisateur,
4. un fichier `ciphertext.txt`, formé de $16 \times 15 = 240$ lignes, qui stocke le chiffré $\boldsymbol{y} = ((y_1^{(1)}, y_2^{(1)}), \ldots, (y_1^{(8N)}, y_2^{(8N)})$ d'un message $\boldsymbol{m} = (m_1, \ldots, m_N)$ formé de $N = 15$ caractères. Le fichier `ciphertext.txt` est écrit de sorte que l'entier $y_i^{(j)}$ est stocké à la $(i + 2j)$-ème ligne.

Question 6.– Déchiffrer le chiffré contenu dans `ciphertext.txt`.