Nouvelle Calédonie – 2025 – sujet2 - Correction

Exercice 1 (6 points)

Partie A – Cryptographie symétrique

1) Instruction pour Bob

Alice a envoyé :

```
m1 = code(m0, cle)
```

Pour déchiffrer, Bob écrit :

```
m2 = code(m1, cle)
```

👉 Commentaire :

> On utilise la propriété donnée :
>
> ```
> code(code(m,cle),cle)=mcode(code(m, cle), cle) =
> mcode(code(m,cle),cle)=m
> ```
>
> Donc la même fonction sert à chiffrer et déchiffrer.
>
> ⚠️ **Problème fondamental :**
> Alice a envoyé **la clé sur le réseau** → Eve peut faire exactement la même chose.

Conclusion : le chiffrement symétrique nécessite un **échange sécurisé préalable de la clé**.

Partie B – Cryptographie asymétrique

2) Instruction pour Bob

Alice a exécuté : `m1 = code(m0, cle1_b)`

Réponse : `m2 = code(m1, cle2_b)`

👉*Commentaire :*

On utilise la propriété :

`code(code(m,cle1),cle2)=mcode(code(m, cle1), cle2) = mcode(code(m,cle1),cle2)=m`

Bob doit utiliser **sa clé privée**.

---

3) Pourquoi Eve ne peut pas lire le message ?

Eve connaît :

- le message chiffré `m1`
- la clé publique `cle1_b`

Mais elle ne connaît pas la clé privée `cle2_b`.

Or il est supposé impossible de déchiffrer un message sans la clé associée.

Donc seul Bob peut lire le message.

---

4) Quelle est la clé publique / privée ?

Pour Bob :

- 🔓 Clé publique : `cle1_b`
- 🔐 Clé privée : `cle2_b`

Même principe pour Alice :

- 🔓 Clé publique : `cle1_a`
- 🔐 Clé privée : `cle2_a`

---

5) Réponse de Bob à Alice

Bob veut envoyer un message secret à Alice.

Bob : `m4 = code(`"Bien reçu. Rendez-vous à 16h donc.", `cle1_a)`

Alice : `m5 = code(m4, cle2_a)`

---

6) Pourquoi deux clés ?

Si une seule clé était utilisée :

- Elle devrait être transmise → elle ne serait plus secrète.
- Tout le monde pourrait déchiffrer les messages.

Le système à deux clés permet :

- de diffuser la clé publique librement,
- de garder la clé privée secrète.

Partie C – Signature

7) Pourquoi Bob est sûr que le message vient d'Alice ?

Alice calcule :

```
m0_s = code(m0, cle2_a)
```

Bob peut vérifier en faisant :

```
code(m0_s, cle1_a)
```

S'il obtient `m0`, alors :

- Le message a été signé avec la clé privée d'Alice.
- Seule Alice possède cette clé privée.

Donc :

- Le message provient bien d'Alice.
- Mallory ne peut pas fabriquer une fausse signature.

👉*Commentaire:*

Signer avec la clé privée permet de **prouver son identité**.

Ici on ne cherche pas à cacher le message, mais à garantir son auteur.

Partie D – Signature par empreinte

8) Réduction de "bac"

Indices :

```
0 1 2
b a c
```

Calcul :

$1\times|97-98|+2\times|99-97|$ $1\times1+2\times2=51 \times 1 + 2 \times 2 = 51\times1+2\times2=5$

Réduction = **5**

---

9) Fonction Python

```
def reduction(chaine):
    total = 0
    for i in range(1, len(chaine)):
        diff = abs(ord(chaine[i]) - ord(chaine[i-1]))
        total += i * diff
    return total
```

---

10) Vérification par Bob

Bob doit :

1. Calculer l'empreinte du message reçu : `empreinte = str(reduction(m0))`
2. Déchiffrer la signature : `verification = code(m0_s, cle1_a)`
3. Comparer : Si `verification == empreinte`, le message est authentique.

---

11) Limiter aux 10 premiers caractères

Cela diminue la taille de la signature mais :

- augmente fortement le risque de collision,
- deux messages différents peuvent produire la même réduction partielle.

Cela réduit donc la sécurité.

Partie E – Chiffrement + Signature

12) Protocole assurant confidentialité et authenticité

Côté Alice:

- Calcul de l'empreinte : `m0_r = str(reduction(m0))`

- Signature : `signature = code(m0_r, cle2_a)`

- Assemblage : (paquet contenant message + signature)

- Chiffrement avec la clé publique de Bob : `paque_chiffre = code((m0, signature), cle1_b)`

Côté Bob:

- Déchiffrement : `paquet = code(paque_chiffre, cle2_b)`

- Séparation message + signature

- Vérification :

```
empreinte = str(reduction(m0))
verification = code(signature, cle1_a)
```

Comparer les deux valeurs.