CRY 2020

Laboratoire #3

15-12-2020

Préambule

- Ce laboratoire demande de résoudre différents challenges basés sur la cryptographie asymétrique. Chaque étudiant doit rendre un rapport **individuel** avec ses propres solutions ainsi que son code. Les façons d'arriver à la solution sont libres mais doivent être expliquées dans le rapport.
- Vous pouvez par contre discuter ensemble pendant que vous résolvez les problèmes. Essayez de ne pas spoiler !
- Voici une liste de fonctions qui pourraient vous être utiles en python :
  - dans le module pycryptodome [1] : `Crypto.Util.number` : `long_to_bytes` et `bytes_to_long`. Ces fonctions existent aussi nativement depuis Python 3.2 (`from_bytes` et `to_bytes`).
- Les exercices contiennent des mots tirés aléatoirement. Ne vous en offensez pas et googlez-les à vos risques et périls.
- Vous trouverez vos entrées personnalisées dans le zip `nom_prenom.zip` sur cyberlearn (dans le devoir où vous devez rendre le rapport).
- Le **rapport** et votre **code** doivent être rendus dans un zip sur cyberlearn avant le **10.01.2021** à 23h55.
- Il se peut que nous annoncions des erreurs sur cyberlearn/teams.

1 El Gamal (1 pt)

Depuis le dernier labo, vous avez décidé d'ignorer toutes les propositions de votre ami informaticien. Malheureusement, il se lance maintenant dans la création d'une startup proposant des solutions cryptographiques "extrêmement efficaces". Vu l'argent qu'il est en train d'investir dans cette entreprise, vous décidez d'arrêter de l'ignorer et de lui donner un coup de main.

Nous avons vu en classe que le protocole de Diffie-Hellman pouvait être défini sur n'importe quel groupe (par exemple $\mathbb{Z}_p^*$ ou sur une courbe elliptique). Il en va de même pour le chiffrement d'El Gamal. Votre ami décide, pour optimiser les performances, d'implémenter El Gamal sur le groupe $\mathbb{Z}_p$, avec p un grand nombre premier. Vu que ce groupe a un nombre premier d'élément, il estime ne pas avoir besoin du nombre q. Dans son cas, g va simplement générer tout $\mathbb{Z}_p$.

1. Implémentez le chiffrement d'El Gamal sur $\mathbb{Z}_p$. Pour cela, vous pouvez compléter le template `elgamal_template.sage` fourni. Dans votre rapport, expliquez mathématiquement comment le chiffrement et le déchiffrement fonctionnent. Vous pouvez utiliser les fonctions de pycryptodome présentées dans l'intro pour convertir votre message en un entier. Si vous le souhaitez, vous pouvez ignorer le template afin de faire votre implémentation dans un autre langage.
2. Malheureusement, cette construction est complètement cassée. Expliquez comment vous pouvez la casser dans votre rapport. Dans votre fichier de paramètres, vous trouverez une clef publique `A1`, des paramètres `p1` et `g1` et un texte chiffré `c1`. Récupérez le texte clair correspondant et donnez-le dans votre rapport.

1. https://pycryptodome.readthedocs.io

3. Etant donné cette attaque, on peut voir qu'El Gamal n'est pas toujours sûr. Que pouvez-vous en déduire sur la sécurité d'El Gamal ? A quoi faut-il faire attention pour qu'il soit sûr ?

2 Chiffrement avec racines carrées (2.5 pts)

Il est possible de chiffrer un message de la manière suivante :
— Soit $n = pq$ un module RSA.
— Le texte chiffré est calculé de la manière suivante $c = m^2 \bmod n$.
— Pour déchiffrer, il suffit de calculer les racines carrées de c.
— Vu qu'un nombre peut avoir 4 racines carrées dans $\mathbb{Z}_n$, il est nécessaire d'ajouter de la redondance dans le texte clair afin de savoir laquelle des racines est la bonne. Dans notre implémentation, cette redondance consiste en 50 bytes nuls à la fin du message.

1. A quoi ressemble la clef privée dans cette construction ? Complétez le code du template `sqrt.sage`.
2. A quoi ressemble le déchiffrement dans cette construction ? Complétez le code du template et testez votre code.
3. Dans votre fichier de paramètres, votre trouverez une clef publique `n2`. En plus de cela, vous avez réussi, en dumpant la RAM juste après un déchiffrement, à récupérer la valeur des quatre racines carrées correspondant au texte chiffré `c2`. Le texte clair correspondant est le message `m2`. A l'aide de ces informations, décryptez le texte chiffré `chall2`. Justifiez et expliquez pourquoi votre attaque fonctionne.

   Indice : pour réussir cette attaque, il faut bien comprendre le théorème des restes chinois. Visualisez les racines dans $\mathbb{Z}_p \times \mathbb{Z}_q$ et tentez d'obtenir un multiple de p qui ne soit pas un multiple de q.

3 RSA (1.5 pts)

Votre ami vous propose 3 solutions sensées améliorer soit les performances, soit la sécurité d'RSA. Cassez-les toutes ![2]

1. Pour la première solution, votre ami trouve le fait de générer deux nombres premiers trop coûteux. A la place, il décide que le module vaut p^4, pour un nombre premier p. Dans vos paramètres, vous avez le module `n31`, l'exposant publique `e31` et un texte chiffré `c31`. Récupérez le message, expliquez dans votre rapport comment vous l'avez récupéré et pourquoi cette solution n'est pas sûre.
2. Pour la deuxième solution, votre ami utilise le code suivant pour générer un module RSA :

```
def keygen2():
    phi = 65537
    e = 65537
    num_primes = 10
    n = 1
    while gcd(phi, e) != 1:
        n = 1
        phi = 1
        for i in range(num_primes):
            p = random_prime(2**60)
            n *= p
            phi *= (p-1)
    return (e, n, inverse_mod(e, phi)) #e, n, d
```

   Dans vos paramètres, vous avez le module `n32`, l'exposant publique `e32` et un texte chiffré `c32`. Récupérez le message, expliquez dans votre rapport comment vous l'avez récupéré et pourquoi cette solution n'est pas sûre.

2. Une fois le message obtenu, obtenez le texte correspondant avec la fonction `long_to_bytes`.

3. Pour la troisième solution, votre ami estime que la sécurité est maximisée lorsque les deux nombres premiers p et q du module RSA sont très proches. En effet, en tirant 1048 bits aléatoire pour chaque nombre premier, il pense qu'il existe un risque que les bits de poids forts soient égals à 0, ce qui réduit la sécurité. C'est pour ça, qu'il s'assure pendant la génération de clef d'avoir des nombres p et q pas trop éloignés. Dans vos paramètres, vous avez le module `n33`, l'exposant publique `e33` et un texte chiffré `c33`. Récupérez le message, expliquez dans votre rapport comment vous l'avez récupéré et pourquoi cette solution n'est pas sûre.