Advanced Cryptography

Lab 1 : Finite field Cryptography

3A informatique majeure MSI

Notation : la note de TP prend essentiellement en compte l'efficacité et la quantité de travail réalisé pendant la séance, de sorte qu'une note de zero est attribuée à la séance en cas d'absence injustifiée et que la note baisse proportionnellement en cas de retard. Néanmoins la fonctionnalité et la qualité du code rendu sont aussi pris en compte dans la note.

Rendu attendu : les étudiants doivent développer (en python) de manière individuelle le travail demandé. Celui-ci doit être une archive au format zip contenant au minimum quatres fichiers : un fichier contenant les classes, un fichier contenant les fonctions de tests, un fichier contenant les autres fonctions (si elles existent) et un fichier readme.md. L'archive devra aussi contenir les fichiers binaires générés par openssl s'il y en a.

Le projet se prête bien à l'utilisation de classe abstraite, portant sur un groupe générique que l'on instanciera plus tard avec une loi de groupe donnée. Néanmoins le sujet ne prévoit pas cette direction qui risque d'augmenter la charge de travail. A réserver aux étudiants les plus motivés.

Consignes sur le plagiat : regarder le code source d'une autre personne, chercher du code sur internet ou sur des outils tels que chatgpt, ou travailler à plusieurs n'est pas interdit. Cela ne doit toutefois pas servir d'excuse pour du plagiat. Récupérer du code source extérieur en le modifiant à la marge (nom de variable, commentaires, etc...) est du plagiat. Au moindre doute, écrire l'origine des sources extérieures en commentaires (code récupéré de tel site internet, auprès de telle personne, ...). Le cas échéant, laisser votre propre code source en commentaire, en expliquant que celui-ci ne fonctionnant pas, vous avez du vous tourner vers un autre code.

Part 1. The class Group.

We consider the following class `Group` and the constructor `__init__` :

```
class Group(object):
    def __init__(self, l, e, N, p):
        self.l = l
        self.e = e
        self.N = N
        self.p = p
        if self.checkParameters() != True:
            raise Exception("Problem with parameters")
```

In this class, the parameter l is a string which indicates the type of group. In this lab, l is equal to "ZpAdditive" or "ZpMultiplicative", e is the identity element, N is the order of the group and p is a prime integer defining $\mathbb{Z}_p$ or $(\mathbb{Z}_p)^\times$.

Write the method `checkParameters` which returns `True` if $l =$ "ZpAdditive" and $e = 0$ or if $l =$ "ZpMultiplicative" and $e = 1$ and `False` otherwise. Don't forget the `self` parameter in the method definition. Remark : the parameters l and e are recovered in the method with `self.l` and `self.e`. The method `checkParameters` should raise an exception if l is unknown.

Write a method `law` with two parameters g_1 and g_2 (and `self` as first parameter) which returns $g_1 * g_2$ where $*$ is the law group depending to l (and p).

Write a method `exp` with two parameters g and k, where g is a group element and k an integer, which returns the group element g^k, using the MontgomeryLadder algorithm. Complete this method by returning e if $k = 0$ and the inverse of g if $k = -1$, using $g^N = g * g^{N-1} = e$ (Lagrange Theorem).

Write a function `testLab1_part1` (outside the class) which verifies that `exp(5, 7)` returns 12 in the additive group $\mathbb{Z}_{23}$ and 17 in the multiplicative group $(\mathbb{Z}_{23})^\times$. In the first case, it corresponds to something like :

```
monGroupe = Group("ZpAdditive", 0, 23, 23)
print("In Z23 : exp(5,7) = 12 ?", monGroupe.exp(5,7) == 12).
```

Verify that the inverse of 5 modulo 23 is 18 and 14 with the `exp` method respectively in $\mathbb{Z}_{23}$ and $(\mathbb{Z}_{23})^\times$.

Part 2. The class SubGroup.

We consider the following class `SubGroup` which inherits from the class `Group` and the constructor `__init__` :

```
class SubGroup(Group):
    def __init__(self, l, e, N, p, g):
        Group.__init__(self, l, e, N, p)
        self.g = g
```

In this class, the parameter g is the generator of the SubGroup. Remark : this lab don't uses the order of g, which corresponds to the order of the subgroup.

Write a method `DLbyTrialMultiplication` in this subclass, with one parameter : a group element h, which returns the integer i between 0 and $N - 1$ such that $g^i = h$. This method computes this integer by trials multiplications.

Complete the function `testLab1_part1`, by considering the subgroup of $\mathbb{Z}_p$, with $p = 809$, generated by $g = 3$ (which is $\mathbb{Z}_p$ itself because $\gcd(3, 809) = 1$). Let i be a random integer between 0 and p and `h = exp(g, i)`, recover i with `ComputeDLbyTrialMultiplication`. Test also with $p = 809$ and $g = 3$ in the group $(\mathbb{Z}_p)^\times$ (where g is a primitive root modulo p).

Part 3. The Diffie-Hellman protocol.

Complete the class `SubGroup` with a method `testDiffieHellman`, without parameters, which generates two random integers a and b between 0 and N, computes $A = g^a$ and $B = g^b$ and returns `True` if $A^b = B^a$ and `False` otherwise. Write a function `testLab1_part2` outside any class, by calling this method with the group $\mathbb{Z}_{23}$ and $g = 5$.

Complete the class `SubGroup` with a method `DiffieHellman`, with two integers a and b and three group elements A, B and K as parameters. The method should return `True` if $A = g^a$, $B = g^b$ and $K = A^b = B^a$ and `False` otherwise. Complete the function `testLab1_part2` by calling this method with the group $\mathbb{Z}_{23}$ and $g = 5$, and $a = 5$, $b = 6$, $A = 2$, $B = 7$ and $K = 12$.

Part 4. DLP on binary fields.

Let n be a positive integer. An element of the finite field $\mathbf{F}_{2^n}$ can be represented by $\sum_{i=0}^{n-1} p_i \alpha^i$, where $p_i \in \mathbf{F}_2$ and α is a symbolic root of an irreductible polynomial of degree n. In this lab, an element of $\mathbf{F}_{2^n}$ is represented by an integer x with the binary decomposition $x = \sum_{i=0}^{n-1} p_i 2^i$. Thus, the ith bit of x is obtained with `(x>>i) & 1`. For example, the element $\alpha^2 + \alpha$ of $\mathbf{F}_8$ is represented by the integer 6 (110 in binary or `(1<<2)^(1<<1)` in python). Reminder : the addition of two elements x and y of $\mathbf{F}_{2^n}$ is performed with a xor : `x^y`.

Multiplication of two elements in $\mathbf{F}_{2^n}$: the multiplication of two elements in $\mathbf{F}_8 = \{0, 1, \alpha, \alpha + 1, \alpha^2, \alpha^2 + 1, \alpha^2 + \alpha, \alpha^2 + \alpha + 1\}$ is performed using the relation $P(\alpha) = 0$, where $P(x) = x^3 + x + 1$ is an irreductible polynomial. Remark : the following algorithm is a variant of the Russian multiplication : `http://mathworld.wolfram.com/RussianMultiplication.html`.

Inputs : two elements x and $y \in \mathbf{F}_8$. Output : the product $p = xy \in \mathbf{F}_8$.

1. `p = 0`
2. While $y \neq 0$:
   (a) If `(y & 1) > 0` then `p = p ^ x` (addition in $\mathbf{F}_8$)
   (b) `x = x << 1`
   (c) If `(x & (1 << 3)) > 0` then :
       `x = x ^ (1 << 3) ^ (1 << 1) ^ 1`
   (d) `y = y >> 1`
3. Return p.

You can use the file `lab1_utils.py` for the end of this lab. Complete the method `law` with a new case `l = "F2^n"`, by generalizing the previous algorithm to any binary field. For this, we add a new parameter to the class `Group`, called `poly`, which is the irreductible polynomial used for the field construction and is reprensented by an integer as described above (use `deg` for the group order).

Let $\mathbf{F}_{256}$ be the field, defined with the irreductible polynomial $x^8+x^4+x^3+x+1$. Test the method `law` by verifying that $45 \times 72 = (1+\alpha^2+\alpha^3+\alpha^5)(\alpha^3+\alpha^6) = 198 = \alpha+\alpha^2+\alpha^6+\alpha^7$ in $\mathbf{F}_{256}$ in `testLab1_part2`.

Let $g = \alpha + 1$ be a primitive element of $\mathbf{F}_{256}$. Generate a random integer i between 1 and 255, compute $h = g^i$ with the previous method `exp` and verify that `DLbyTrialMultiplication` returns the good result. Call the method `testDiffieHellman`, with g on the group $(\mathbf{F}_{256})^*$.

Part 5. Baby Step Giant Step.

Write a method `ComputeDL`, with two parameters : an integer τ (by default $\tau = 1000$) and a group element h, which returns the integer i between 0 and $N-1$ such that $g^i = h$. This method only calls the method `DLbyTrialMultiplication` if $N \leq \tau$ or the method `DLbyBabyStepGiantStep` otherwise, and returns the result obtained by these methods.

Let $g = \alpha + 1$ be a primitive element of $\mathbf{F}_{256}$. Generate a random integer i between 1 and 255, compute $h = g^i$ and verify the result using a call of `ComputeDL` with $\tau = 100$ (it calls `DLbyBabyStepGiantStep`).