TP : Mecanismes Cryptographiques

Olivier Blazy

Janvier 2017

Un compte-rendu clair est à remettre sur moodle au plus tard le 11 février à minuit. Vous rendrez un pdf avec votre code ainsi que les réponses aux questions (n'hésitez pas à commenter votre code, # en sage), ce qui permet d'avoir le code et les réponses aux questions.

1 Introduction

Sage (Math) est un logiciel libre de calul formel et de mathématique sous la licence publique GNU version 2. (Ce qui en fait une alternative open source aux systèmes comme Magma, Maple et Mathematica ou même Matlab). Il combine de nombreux programmes libres dans une interface basée sur Python. Sage a deux modes d'utilisation, un mode bloc-note accessible via un navigateur web, et un mode en ligne de commande.

Vous pouvez vous contenter de l'interface web pour les besoins de ce TP : `https://sagecell.sagemath.org/`. Bien entendu une installation locale marche aussi. Une documentation est disponible à l'adresse `http://doc.sagemath.org/html/fr/tutorial/index.html`.

2 Tutoriel

2.1 Courbes elliptiques sur $\mathbb{F}_p$ avec p premier

On rappelle qu'une courbe elliptique $\mathcal{E}$ définie sur $\mathbb{F}_p$, avec p premier est donnée par une équation de la forme : $\mathcal{E} : y^2 = x^3 + ax + b$, avec a, b deux éléments de $\mathbb{F}_p$ tels que $a^3 + 27b^2 \neq 0 \bmod p$.

Par exemple, on va étudier la courbe $\mathbb{E}$ définie pour $p = 101$ et $y^2 = x^3 + x + 1$.

Exercice 1. Echauffement

1. Définissez une variable p, et un corps avec la commande FiniteField.
2. Définissez les coefficients a, b.
3. Vérifiez que la courbe vérifie les critères sur a, b.
4. Construisez la courbe elliptique E sur $\mathbb{F}_p$ avec a, b avec la commande EllipticCurve.
5. Trouvez le nombre de points de la courbe avec E.cardinality(), et factorisez-le avec factor. Affichez les points avec E.points()

6. Pour visualiser la courbe, invoquez la commande plot(E). Expliquez ce que vous voyez.
7. Générez un point aléatoire avec E.random_element() et comparez son ordonnée et son abscisse par rapport à son opposé.

   Pour chercher l'ordre d'un point, on utilise P.order(), et on teste la primalité d'un nombre avec is_prime, pour générer un point de la courbe on utilise E.gens()[0]
8. Mettez maintenant en place un Diffie Hellman.

Quand l'ordre du corps de base n'est pas premier, la commande EllipticCurve prend $[1, a, 0, 0, b]$ en argument au lieu de $[a, b]$.

2.2 Fonction de Hashage

Afin de signer vous aurez besoin d'une fonction de hashage (sha256 par exemple). Vous aurez besoin d'importez la librairie hashlib.

Exercice 2. Hash the cheerleader, hash the world Importez la bibliothèque hashlib avec import hashlib, définissez le message à hasher, la fonction à utiliser : message = 'cheerleader", f = hashlib.sha256()

— Donnez les sorties de : f.update(message), f.hexdigest(), f.digest().

Il existe d'autres librairies au besoin.

3 Ressources pour le TP

```
def couplage(A, B):
    l = A.order()
    k = 2
    Fp = (A.curve()).base_field()
    p = Fp.characteristic()
    Fq.<w> = FiniteField(p^k)
    EE = E.base_extend(Fq)
    AA = EE(A); BB = EE(B);
    PP.<x> = PolynomialRing(Fq)
    f = x^2+1
    i = f.roots()[0][0]
    BBt = EE(-BB.xy()[0], i*BB.xy()[1])
    return AA.weil_pairing(BBt, l)

def hash_curve(E, P, mot):
    import hashlib
    m = hashlib.sha256()
    m.update(mot)
    a = int(m.hexdigest(), 16) % p
```

```
    return a*P



def hash_couplage(res):
    import hashlib
    m = hashlib.md5()
    mot = str(res._matrix_()[0][0]) + str(res._matrix_()[0][1])
    m.update(mot)
    m.hexdigest()
    return m.hexdigest()



def XOR(a, b):
    aa = bin(int(a, 16))[2:]
    bb =  bin(int(b, 16))[2:]
    complete = ''.join('0' for i in [1..abs(len(aa)-len(bb))])
    if len(aa) > len(bb):
        bb = complete + bb
    else:
        aa = complete + aa
    res = ''.join('0' if i == j else '1' for i, j in zip(aa[::-1],bb[::-1])[::-1])
    res = aa[len(res):] + bb[len(res):] + res
    resul = hex(int(res, 2))[2:]
    if resul[len(resul)-1] == 'L':
        resul = resul[:-1]
    return resul
```

— couplage : Cette fonction prend deux points d'une courbe $\mathcal{E}$ et renvoie une valeur dans $\mathbb{F}_{p^k}$.
— hash_curve : Cette fonction prend en paramètre le triplet $(\mathcal{E}, P, id)$ où $\mathcal{E}$ est la courbe elliptique sur laquelle on chiffre, P un générateur et id l'identité à hacher.
— hash_couplage : Cette fonction prend en paramètre un élément du corps et le convertit en un hexadécimal de longueur fixe.
— XOR : cette fonction permet de faire le ou exclusif entre 2 héxadécimaux.

4 Les courbes elliptiques avec Sage

Exercice 3.

— Générez une courbe $\mathcal{E}$ avec 50 bits de sécurité sur $\mathbb{F}_p$ où p est un nombre premier. On n'oubliera de noter dans le compte rendu la valeur de p choisie, ainsi que l'équation de la courbe.

— Calculez un générateur de $\mathcal{E}$ que l'on notera P. On n'oubliera pas de noter les coordonnées de P ainsi que son ordre.
— Simulez Diffie-Hellman sur $\mathcal{E}$.
— Simulez ECDSA sur $\mathcal{E}$: Signez un message m puis vérifiez la signature.

Exercice 4.
— Générez une courbe $\mathcal{E}$ avec 50 bits de sécurité sur $\mathbb{F}_{2^n}$ où n est un nombre premier. On n'oubliera de noter dans le compte rendu la valeur de n choisie, ainsi que l'équation de la courbe.
— Calculez un générateur de $\mathcal{E}$ que l'on notera P. On n'oubliera pas de noter les coordonnées de P ainsi que son ordre.
— Simulez Diffie-Hellman sur $\mathcal{E}$.
— Simulez ECDSA sur $\mathcal{E}$: Signez un message m puis vérifiez la signature.

Exercice 5. Echange de clé tripartite
— Construire la courbe $\mathcal{E} : y^2 = x^3 + x$ définie sur $\mathbb{F}_p$ avec $p = 1125899906842679$.
— Quel est le plus grand facteur premier (noté ℓ) du cardinal de $\mathcal{E}$?
— Construire un point P d'ordre ℓ, notez ses coordonnées et son ordre.
— Simulez un échange de clés tripartite entre Alice, Bob et Charlie.
— Vérifiez l'exactitude de votre schéma en testant l'égalité des divers couplages

Exercice 6. Chiffrement basé sur l'identité Le chiffrement basé sur l'identité permet de se passer de clé publique pour chaque utilisateur, le premier (officiellement) proposé est le chiffrement dit de Boneh Franklin. Il suppose l'existence d'une fonction de hashage $\mathcal{H}$, d'un couplage e sur un groupe $\mathbb{G}$. Une autorité, publie une paire de clé $\mathsf{msk}, \mathsf{mpk}$: $x \in \mathbb{Z}_p, g^x$.

Chaque utilisateur a une clé secrête $\mathsf{usk}[\mathsf{id}] = \mathcal{H}(\mathsf{id})^x$.

Pour chiffrer un message M, pour une identité id : $C{=}g^r, \mathcal{H}'(e(\mathcal{H}(\mathsf{id}), g^x)^r) \oplus M$.

Pour déchiffrer, un utilisateur calcule $\mathcal{H}'(e(C_1, \mathsf{usk}[\mathsf{id}])) \oplus C_2$.

— Jouez le rôle de l'autorité, et générez msk et mpk.
— Construisez votre propre usk, en utilisant votre adresse email comme identité.
— Générez la clé publique correspondant à votre identité, et chiffrez un message. (Pensez à convertir votre message M en hexadécimal avec M.encode('hex') avant d'utiliser la fonction xor.)
— Déchiffrez le message précédent avec votre clé secrête.

5 Openssl

La commande openssl ecparam -list_curves donne la liste des courbes possibles pour chiffrer. On génère une clé basée sur une courbe elliptique avec la commande ec (et l'option keygen). Pour signer un message avec ECDSA, on utiliser la commande dgst avec l'option -ecdsa-with-SHA1.

Exercice 7. OpenSSL
— Générez une paire de clés pour ECDSA, avec la coubre secp192.

— Chiffrez avec votre clé privée.
— Etablissez une requête de certificat pour votre clé publique.
— Signez plusieurs messages et vérifiez-les.
— Chiffrez un email et signez avec ECDSA. Faites la vérification.