# Cryptohraphy project

## Part 1 :

The objective of the first part of this project is to create 2 servers.

The first server must manage everything related to the user, i.e. registration, connection. While the second server must encrypt the data before saving it in the database.

For this first part I decided to make the servers using python, and the database and a docker database.

To start the first part, you should do :

- Bash build_docker.sh
- Bash run_docker.sh
- Python server1.py
- Python server2.py

« Dockerfile.db » is for the database

« init.sql » is for the initialisation of the database

« Templates » is for the interface

« Question » is the answer of the first part

And finaly, we have the servers

Server1.py :

Connexion to the databse :

```
app = Flask(__name__)
app.secret_key = secrets.token_hex(16)
app.config['SQLALCHEMY_DATABASE_URI'] = 'postgresql://admin:mon_mot_de_passe@localhost/password_db'
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False
```

Register :

So we take the data from the form and we hashed the password and send it to the server2.

The server2 give back the encrypted password and we store it in the databse

```
@app.route('/create', methods=['POST', 'GET'])
def create():
    form = LoginForm()

    if form.validate_on_submit():
        username = form.username.data
        password = form.password.data

        # Generate a unique salt for each user
        salt = bcrypt.gensalt()
        print(salt)
        password_bytes = (password).encode()
        password = bcrypt.hashpw(password_bytes, salt)
        print(password)
        password = password.decode('utf-8')

        response = requests.post(SERVER2_URL, json={'password': password})
        if response.status_code == 200:

            encrypted_hash = response.json().get('encryptedHash')
            user = User(id=str(uuid.uuid4()), username=username, hashed_password=encrypted_hash,salt=salt)
            db.session.add(user)
            db.session.commit()

            return jsonify({'message': 'Success'}), 200
        else:
            # En cas d'échec de la requête au Serveur 2
            return jsonify({'error': 'Failed to encrypt password'}), 500
```

Server2 :

The server2 just encrypt the password. Unfortunately, i can not store the key in a file.txt because i have a bug :

```
app = Flask(__name__)
daead.register()
keyset_handle = tink.new_keyset_handle(daead.deterministic_aead_key_templates.AES256_SIV)

# Generate a single AEAD primitive for encryption
daead_primitive = keyset_handle.primitive(daead.DeterministicAead)
SERVER2_LOGIN_URL = "http://localhost:5001"

@app.route('/encrypt', methods=['POST'])
def encrypt():
    data = request.json
    hashed_password = data.get('password')
    hashed_password=hashed_password.encode()

    # Encrypt with Tink
    ciphertext = daead_primitive.encrypt_deterministically(hashed_password, b'')
    print(ciphertext)
    print(base64.b64encode(ciphertext).decode('utf-8'))

    return jsonify({'encryptedHash': base64.b64encode(ciphertext).decode('utf-8')}), 200
```

# Part 2 :

The part 2 there is an error when I send the data about the jsons format, and I can't resolve it but the other functions work.

To start the project you just need to do :

- Python Alice.py
- Python Bob.py

So we have two servers : Alice and Bob.

The first server (Alice):

So if we look the main the user is elies, and the password is abc. First we generate the oprf key.

```
def oprf(mdp, nom_utilisateur):
    mdp_hashe = hash(mdp)
    #we do this because we cannot do this because it takes too many time
    #r = int.from_bytes(os.urandom(16), byteorder='big')
    #C = int.from_bytes(mdp, byteorder='big') ** r
    r = secrets.randbits(128)
    C = pow(int.from_bytes(mdp_hashe, byteorder='big'), r, PARAMETRES_DH.parameter_numbers().p)
    print('salut')
    envoyer(json.dumps(["OPRF", nom_utilisateur, C]))
    print("Envoi des données OPRF...")
    R = attendre()
    print("Réception des données OPRF...")
    r_inverse = pow(r, -1, PARAMETRES_DH.parameter_numbers().q)
    K = R ** r_inverse
    print("OPRF terminé avec succès.")
    return K
```

After we send the user and C to Bob (because bob is waiting the message of Alice). Since Alice is connected to the port (8080), and bob too), Alice send the data

```python
def envoyer(donnees):
    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as sock
        try:
            sock.connect(('localhost', 8080))
            print("Connexion établie avec succès.")
            print("Envoi des données :", donnees)
            sock.sendall(json.dumps(donnees).encode())
            print("Données envoyées avec succès.")
        except Exception as e:
            print("Erreur lors de l'envoi des données :", e)
```

and bob answer to the message.

```python
def attendre():
    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as sock:
        sock.connect(('localhost', 8080))
        print("Connexion établie avec succès.")
        reponse = sock.recv(4096)
        print("Données reçues :", reponse.decode())
    return json.loads(reponse.decode())
```

Finaly we just register or login it depend that what we want.

```python
def register(nom_utilisateur, mdp):
    print("Début de l'enregistrement...")
    cle = oprf(mdp, nom_utilisateur)
    print("Clé obtenue :", cle)
    cle_derivee = deriver_cle(cle)
    print("Clé dérivée :", cle_derivee)
    cle_privee = PARAMETRES_DH.generate_private_key()
    print("Clé privée générée avec succès.")
    cle_publique = cle_privee.public_key()
    cle_publique_serialisee = cle_publique.public_bytes(
        encoding=serialization.Encoding.PEM,
        format=serialization.PublicFormat.SubjectPublicKeyInfo
    )
    print("Clé publique générée avec succès.")
    enveloppe = encrypt(cle_derivee, cle_privee.private_bytes(
        encoding=serialization.Encoding.PEM,
        format=serialization.PrivateFormat.PKCS8,
        encryption_algorithm=serialization.NoEncryption()
    ))
```

```
def login(nom_utilisateur, mdp):
    print("Début de la connexion...")
    cle = oprf(mdp, nom_utilisateur)
    print("Clé obtenue :", cle)
    cle_derivee = deriver_cle(cle)
    print("Clé dérivée :", cle_derivee)
    envoyer(json.dumps(["login", nom_utilisateur]))
    print("Demande de connexion envoyée...")
    reponse = attendre()
    print("Réponse reçue :", reponse)
    enveloppe = reponse["enveloppe"]
    donnees_cle_privee = decrypt(cle_derivee, enveloppe)
    cle_privee = serialization.load_pem_private_key(
        donnees_cle_privee,
        password=None,
        backend=default_backend()
    )
    print("Connexion terminée avec succès.")
```

Without forgetting the encryption :

```
def encrypt(cle, donnees):
    """Chiffre les données avec la clé fournie en utilisant AES en mode CTR."""
    chiffreur = Cipher(algorithms.AES(cle[:16]), modes.CTR(os.urandom(16)), backend=default_backend()).encryptor()
    return chiffreur.update(donnees) + chiffreur.finalize()

def decrypt(cle, enveloppe):
    """Déchiffre l'enveloppe avec la clé fournie en utilisant AES en mode CTR."""
    iv, donnees_chiffrees = enveloppe[:16], enveloppe[16:]
    dechiffreur = Cipher(algorithms.AES(cle[:16]), modes.CTR(iv), backend=default_backend()).decryptor()
    return dechiffreur.update(donnees_chiffrees) + dechiffreur.finalize()
```

The second server (Bob) :

The second is waiting the request from Alice and save, search users in the file : users.txt

# Conclusion :

To carry out this project, we easily did part 1, however for part 2, we got a lot of help from the internet because itw as very difficult for us. However thanks a lot, because we learn a lot of things we this cryptographic project.