InLab:

```
from ecdsa import SigningKey, SECP256k1
from cryptography.hazmat.primitives import hashes
from cryptography.hazmat.primitives.kdf.pbkdf2 import PBKDF2HMAC
from cryptography.hazmat.primitives.kdf.hkdf import HKDF
from cryptography.hazmat.backends import default_backend
from cryptography.hazmat.primitives.ciphers import Cipher, algorithms, modes
import os

def generate_keys():
    private_key = SigningKey.generate(curve=SECP256k1)
    public_key = private_key.get_verifying_key()
    return private_key, public_key

def encrypt(public_key, plaintext):
    shared_secret = public_key.to_string()  # In a real scenario, this would be generated through ECDH
    salt = os.urandom(16)

    kdf = PBKDF2HMAC(
        algorithm=hashes.SHA256(),
        length=32,
        salt=salt,
        iterations=100000,
        backend=default_backend()
    )
    key = kdf.derive(shared_secret)
    iv = os.urandom(12)
    cipher = Cipher(algorithms.AES(key), modes.GCM(iv), backend=default_backend())
    encryptor = cipher.encryptor()
    ciphertext = encryptor.update(plaintext.encode()) + encryptor.finalize()
    return salt, iv, ciphertext, encryptor.tag

def decrypt(private_key, salt, iv, ciphertext, tag):
    shared_secret = private_key.verifying_key.to_string()
        kdf = PBKDF2HMAC(
        algorithm=hashes.SHA256(),
        length=32,
        salt=salt,
        iterations=100000,
        backend=default_backend()
    )
    key = kdf.derive(shared_secret)
    cipher = Cipher(algorithms.AES(key), modes.GCM(iv, tag), backend=default_backend())
    decryptor = cipher.decryptor()
    plaintext = decryptor.update(ciphertext) + decryptor.finalize()

    return plaintext.decode()

private_key, public_key = generate_keys()
print("Private Key:", private_key.to_string().hex())
print("Public Key:", public_key.to_string().hex())
message = "Hello, this is a secret message."
salt, iv, ciphertext, tag = encrypt(public_key, message)
print("Ciphertext:", ciphertext.hex())
decrypted_message = decrypt(private_key, salt, iv, ciphertext, tag)
print("Decrypted Message:", decrypted_message)
```