import hashlib  
import secrets  
  
# Получение хэша от переданного значения (строки)  
def generate\_hash(value):  
 # Преобразуем значение в байты, если оно еще не в байтах  
 if isinstance(value, str):  
 value = value.encode('utf-8')  
 # Создаем хэш-объект SHA-256 и возвращаем хэш  
 return hashlib.sha256(value).hexdigest()  
  
# Генерация случайных пар для секретного числа (256 пар, так как 256 битный хэш)  
def generate\_secret\_key():  
 secret\_key = [] # Array of secret pairs  
 for i in range(256):  
 a = secrets.token\_bytes(16) #128 bites  
 b = secrets.token\_bytes(16)  
 secret\_key.append((a, b))  
 return secret\_key  
  
# Создание публичного ключа на основе секретного  
def generate\_public\_key(secret\_key):  
 public\_key = [[generate\_hash(x), generate\_hash(y)] for x,y in secret\_key]  
 return public\_key  
  
# Создание подписи Лемпорта  
def generate\_lamport\_signature(document, secret\_key):  
 signature = []  
 document\_hash = hashlib.sha256(document.encode()).hexdigest()  
 bit\_string = bin(int(document\_hash, 16))[2:].zfill(256)  
  
 for i, bit in enumerate(bit\_string):  
 a, b = secret\_key[i]  
 signature.append(a if bit == "0" else b)  
  
 return signature  
  
# Генерация хэша публичного ключа  
def get\_public\_key\_hash(public\_key):  
 serialized\_key = ""  
 for a, b in public\_key:  
 serialized\_key += a + b  
 key\_hash = hashlib.sha256(serialized\_key.encode()).hexdigest()  
 return key\_hash  
  
  
def create\_signature\_lamport(text):  
 secret\_key = generate\_secret\_key()  
 public\_key = generate\_public\_key(secret\_key)  
 signature = generate\_lamport\_signature(text, secret\_key)  
 public\_key\_hash = get\_public\_key\_hash(public\_key)  
 return (signature, public\_key\_hash)  
  
# Функция для создания дерева Меркла  
def build\_merkle\_tree(leaves):  
 # Начальный уровень — листья, сохраняем их как первый уровень дерева  
 tree\_levels = [leaves]  
 current\_level = leaves  
  
 # Пока не достигнем корня  
 while len(current\_level) > 1:  
 next\_level = []  
  
 # Проверка на четное количество элементов  
 if len(current\_level) % 2 != 0:  
 current\_level.append(current\_level[-1]) # Дублируем последний элемент, если нечетное количество  
  
 # Создаем следующий уровень  
 for i in range(0, len(current\_level), 2):  
 combined\_hash = generate\_hash(current\_level[i] + current\_level[i + 1])  
 next\_level.append(combined\_hash)  
  
 # Добавляем следующий уровень в список уровней дерева  
 tree\_levels.append(next\_level)  
 current\_level = next\_level  
  
 # Возвращаем дерево и корневой хэш  
 return tree\_levels, current\_level[0] if current\_level else None  
  
def create\_public\_hash(list\_of\_signatures):  
 return [y for x, y in list\_of\_signatures]  
  
  
def get\_proof(mercle\_tree, index):  
  
 trace = []  
ы  
 for i in range(len(mercle\_tree) - 1 ):  
 if index % 2 == 0:  
 trace.append((mercle\_tree[i][index], mercle\_tree[i][index + 1]))  
 else:  
 trace.append((mercle\_tree[i][index - 1], mercle\_tree[i][index]))  
 index = int(index / 2 )  
 trace.append(mercle\_tree[-1])  
 return trace  
  
  
def create\_signatures(documents):  
 list\_of\_signatures = []  
  
 for i in documents:  
 list\_of\_signatures.append(create\_signature\_lamport(i))  
  
 return list\_of\_signatures  
  
  
# -------------------------  
Documents = ["Hello, world!", "Hi!", "Lamport signature", "Mercle Tree","123", " 345", "fdg", "df"]# Список докуметов  
list\_of\_signatures = create\_signatures(Documents)  
list\_of\_pbc\_hash = create\_public\_hash(list\_of\_signatures) # список публичных ключей подписей  
  
mercle\_tree = build\_merkle\_tree(list\_of\_pbc\_hash)[0] # Дерево Меркла  
root\_hash = build\_merkle\_tree(list\_of\_pbc\_hash)[1] # Root-hash  
proof = get\_proof(mercle\_tree, 3)  
  
#print("List of signatures:", list\_of\_signatures)  
print("List of public hashes:", list\_of\_pbc\_hash)  
print("\nMercle Tree:")  
for i in mercle\_tree:  
 print(i)  
print("\nRoot Hash: ", root\_hash)  
print("Proof for document №3: ", proof)  
print(generate\_hash(proof[-2][0] + proof[-2][1]))

Результат:

**List of signatures**: [([b'z\xba?\x91\x8e\xe95m\xcd#\x1a\x1e\x06\xb9\xc2\_', b'\xa8\x83\x9a=\xe4\xed`\xf8$\xcc)\xb1\x13\x8fh\x89', b'\xee\x94l\x0b=\xf33#\x9d\x880\xf6vW\x8bB', b'v\x15#\xaf.\x04\xf4\xcc\t\xd7?\xd9\x94\xff\x95\xd3', b'\xfa%Z\xc6&\xc5\x08\x96\xcc\x8c]\xd4\xec\xb7\x18\xae', b'dK\xd4.\xfc\xd5\xc7N"\x07\xbc\xb5J\xcd`\xab',………………………………….. '333940c336e2be997061afabb8cf6cd7424960e0830ed6c2d3ec84493f8c91b8')]

**List of public hashes:** ['98df099859e936feb087ac5f99fd68bc98795c7528cf33d2638f313ec1b61fe2', '3f07770831c50d95380bae100355eb492622c75c4639728d4dba97692aa8a6e8', '6ee57f43bd81451373ff5c3639b59cfc5becef8f247885415aefe544c619e3da', '175a60e51d080758e2da2a1d7658e1400c61c866afa98a45451e1cf07f144167', '82800fdfc95443bf073b3cdfb4fe1075f7236e41edc7ee4fe2469a06ed677736', '7910c8d85187e44bf71314d4c038d3147cb60f19000ff36602a4e7be35461515', '3b304e93b56b353011eb97645a980d5411ed135280e8ed11451209ec258d98fd', '333940c336e2be997061afabb8cf6cd7424960e0830ed6c2d3ec84493f8c91b8']

**Mercle Tree:**

['98df099859e936feb087ac5f99fd68bc98795c7528cf33d2638f313ec1b61fe2', '3f07770831c50d95380bae100355eb492622c75c4639728d4dba97692aa8a6e8', '6ee57f43bd81451373ff5c3639b59cfc5becef8f247885415aefe544c619e3da', '175a60e51d080758e2da2a1d7658e1400c61c866afa98a45451e1cf07f144167', '82800fdfc95443bf073b3cdfb4fe1075f7236e41edc7ee4fe2469a06ed677736', '7910c8d85187e44bf71314d4c038d3147cb60f19000ff36602a4e7be35461515', '3b304e93b56b353011eb97645a980d5411ed135280e8ed11451209ec258d98fd', '333940c336e2be997061afabb8cf6cd7424960e0830ed6c2d3ec84493f8c91b8']

['46d325ee4a63e69059ae32ed3fa1e20e9cfbdf5d8cefd0201ebd2c54c3bcc83d', '4b78197ad084cd6a724a63736652a95f91e205269fbe39dd155e8cd3025c7a00', '1383ef7a2e928bfabffb7ef98259c29dcf60a7d198006cefa62b723f55b8659f', 'd849a24ba25f34303faf444d5dda64850bd509836ce1a281e9d50835e3f5c1f9']

['104b22f9493750b3b5f178819e903e6019ba4ab68e9260144dba27fa8c2ceae4', '8c601d83775e0b1f2a7d344b80b83777136381f554d5fb148dd1f7fa0eb5f4cd']

['0554a162cff2a29a3d85395ffc174fc978377bd92c377e802bf3c9193e959132']

**Root Hash**: 0554a162cff2a29a3d85395ffc174fc978377bd92c377e802bf3c9193e959132

**Proof for document №3:** [('6ee57f43bd81451373ff5c3639b59cfc5becef8f247885415aefe544c619e3da', '175a60e51d080758e2da2a1d7658e1400c61c866afa98a45451e1cf07f144167'),

('46d325ee4a63e69059ae32ed3fa1e20e9cfbdf5d8cefd0201ebd2c54c3bcc83d', '4b78197ad084cd6a724a63736652a95f91e205269fbe39dd155e8cd3025c7a00'), ('104b22f9493750b3b5f178819e903e6019ba4ab68e9260144dba27fa8c2ceae4', '8c601d83775e0b1f2a7d344b80b83777136381f554d5fb148dd1f7fa0eb5f4cd'),

['0554a162cff2a29a3d85395ffc174fc978377bd92c377e802bf3c9193e959132']]

Результат хэширвания 2 хэшей с предпоследнего уровня: 0554a162cff2a29a3d85395ffc174fc978377bd92c377e802bf3c9193e959132