Informe sobre la Implementación del Esquema de Firma Digital LUOV

Resumen

Este informe explica la estructura y funcionalidad del esquema de firma digital LUOV (Lifted Unbalanced Oil Vinegar). LUOV es un protocolo criptográfico post-cuántico que garantiza la seguridad en la generación y verificación de firmas resolviendo sistemas de ecuaciones específicos sobre campos finitos.

El proyecto consta de varios módulos en Python, cada uno con un propósito específico dentro del esquema. Este informe incluye explicaciones detalladas de los componentes clave e integra fragmentos de código relevantes para mayor claridad.

1. Constantes (constants.py)

```
r = 7   # Grado de extensión del campo F2 -> F2^r
m = 57  # Número de variables "oil"
v = 197 # Número de variables "vinegar"
n = m + v  # Total de variables
SEED_SIZE = 32
SECURITY_LEVEL = 1  # Nivel de seguridad: 1, 3 o 5
```

Propósito: Estos parámetros dictan el tamaño y la complejidad de las estructuras matemáticas utilizadas en el esquema LUOV, como matrices y vectores.

2. Generación de Claves (keygen.py)

Este módulo genera las claves privada y pública:

Funciones Clave

- **generate_private_seed**: Genera un seed privado aleatorio.

```
def generate_private_seed():
    return os.urandom(max(SEED_SIZE, 57))  # Asegura un tamaño mínimo de 57 bytes
```

- **generate_keys**: Deriva el seed público y otros parámetros a partir del seed privado.

```
def generate_keys(private_seed):
    public_seed = SqueezePublicSeed(private_seed)
```

C, L, Q1 = SqueezePublicMap(public_seed)

Q2 = np.random.randint(0, 256, size=(57, ((v * (v + 1)) // 2) + (v * m)), dtype=np.uint8)

return (public_seed, Q2), private_seed

```
PS C:\Users\User\Downloads\LUOV_Project_1-master\luov> python validate.py
Starting validation process...

Private Seed: 75d5b762d45b8bfca30f20978c8ec7086b5ca05376160daa06534f3105ceece8ba9b733fe0bf9a6838713a8832517b9cfae1b6564484212076
Generated Q2 Shape: (57, 30732)
Public Seed (Hex): 75d5b762d45b8bfca30f20978c8ec7086b5ca05376160daa06534f3105ceece8
Q2 (Shape): (57, 30732)
Validating key generation...
Key generation validation passed.
Message: This is the test message for LUOV validation.
```

Propósito: Estas funciones aseguran una generación de claves segura mientras mantienen compatibilidad con la estructura del esquema.

3. Generación de Firmas (sign.py)

Este módulo implementa el proceso de firmar mensajes usando la clave privada.

Funciones Clave

- **Sign**: Genera una firma para un mensaje dado:

```python
def Sign(private_seed, message):
    public_seed, T = generate_public_seed_and_T(private_seed)
    C, L, Q1 = SqueezePublicMap(public_seed)
    salt = os.urandom(16)
    h = Hash(message + b"\x00" + salt, m)
    while True:
        vinegar_vector = np.random.randint(0, 256, size=v, dtype=np.uint8)
        try:
            A = BuildAugmentedMatrix(C, L, Q1, T, h, vinegar_vector)
            solution = GaussianElimination(A)
            if solution is not None:
                oil_vector = solution
                break
        except ValueError:
            continue
    s = np.concatenate((vinegar_vector, oil_vector))
    return s, salt
```

```
Signing the message...
Initial T shape: (197, 57), L shape: (57, 254)
C shape: (57,)
L shape: (57, 254)
Q1 shape: (57, 30732)
vinegar_vector shape before adjustment: (197,)
Adjusting vinegar_vector from 197 to 254
Computed RHS shape: (57,)
Adjusting T rows from 197 to 57
Computed LHS shape: (57, 311)
Final Augmented Matrix shape: (57, 58)
Signature (length 254): [135.  42. 244.  74.  15. 120.  95. 121.   5. 237.]... (truncated)
Salt (Hex): 69abc93d430eedb5adfe7a99514e76b7
Converting signature from float to uint8...
Combined Signature + Salt (Array): [135  42 244  74  15 120  95 121   5 237]... (truncated)
```

Propósito: Esta función asegura que el proceso de firma cumpla con los requisitos matemáticos de LUOV, incluyendo la resolución de sistemas de ecuaciones para derivar el oil_vector.

4. Utilidades (utils.py)

Proporciona funciones auxiliares para operaciones matriciales y derivación de parámetros.

Funciones Clave

- **BuildAugmentedMatrix**: Construye una matriz aumentada para resolver con eliminación gaussiana.

```
def BuildAugmentedMatrix(C, L, Q1, T, h, vinegar_vector):
  # Ajusta vinegar_vector para que coincida con las columnas de L
  if vinegar_vector.shape[0] != L.shape[1]:
    vinegar_vector = np.pad(vinegar_vector, (0, L.shape[1] - vinegar_vector.shape[0]))
  RHS = h - C - np.dot(L, vinegar_vector)
  LHS = L
  if T is not None:
    LHS = np.hstack((LHS, -T))
  augmented_matrix = np.hstack((LHS, RHS[:, np.newaxis]))
  return augmented_matrix
```

GaussianElimination: Resuelve sistemas de ecuaciones usando eliminación gaussiana.

```
def GaussianElimination(A):
    LHS = A[:, :-1]
    RHS = A[:, -1]
    solution = np.linalg.solve(LHS, RHS)
    return solution
```

Propósito: Estas funciones facilitan las operaciones matemáticas clave requeridas para la generación y verificación de firmas.

5. Verificación (utils_for_verify.py)

Este módulo valida la autenticidad de las firmas.

Funciones Clave

- **verify_signature**: Valida una firma dada usando la clave pública.

  def verify_signature(public_key, message, signature):

  public_seed, Q2 = public_key

  C, L, Q1 = SqueezePublicMap(public_seed)

  salt = signature[-16:]

  s = signature[:-16]

  vinegar_vector = s[:v]

  h = hashlib.shake_256(np.concatenate((np.array(list(message), dtype=np.uint8), salt)).tobytes()).digest(m)

  A = BuildAugmentedMatrix(C, L, Q1, None, h, vinegar_vector)

  solution = GaussianElimination(A)

  return solution is not None

Propósito: Asegura que la firma corresponda al mensaje y la clave pública validando el oil_vector derivado.

**** Validación** (validate.py)**

Actúa como el punto de entrada principal para probar la implementación.

Flujo de Trabajo

1. Genera un seed privado y deriva claves.
2. Firma un mensaje de prueba.
3. Verifica la firma.

Fragmento de Código

```
f __name__ == "__main__":
   private_seed = generate_private_seed()
   public_key, private_key = generate_keys(private_seed)
   message = "Test message for validation.".encode("utf-8")
   signature, salt = Sign(private_key, message)
   combined_signature = np.concatenate([signature, np.frombuffer(salt, dtype=np.uint8)]
   is_valid = verify_signature(public_key, message, combined_signature)
   print(f"Verification Result: {'Valid' if is_valid else 'Invalid'}")
```

```
Verifying signature...
Message type: <class 'numpy.ndarray'>, Message shape: (45,)
Salt type: <class 'numpy.ndarray'>, Salt shape: (16,)
Q2 shape: (57, 30732)
C shape: (57,), L shape: (57, 254), Q1 shape: (57, 30732), h length: 57
Vinegar vector shape: (197,)
C shape: (57,)
L shape: (57, 254)
Q1 shape: (57, 30732)
vinegar_vector shape before adjustment: (197,)
Adjusting vinegar_vector from 197 to 254
Computed RHS shape: (57,)
Computed LHS shape: (57, 254)
Final Augmented Matrix shape: (57, 58)
Augmented matrix shape: (57, 58)
Verification Result: Valid

Validation process completed successfully.
```

Propósito: Asegura que toda la tubería LUOV funcione como se espera, desde la generación de claves hasta la verificación de firmas.

Conclusión

La implementación de LUOV cumple con los estándares criptográficos post-cuánticos, aprovechando operaciones matemáticas robustas y técnicas de aleatorización seguras. Cada módulo está diseñado con modularidad y claridad, facilitando futuras adaptaciones o optimizaciones.