Informe de Avance - LUOV Digital Signature Scheme

Curso: Criptografía
Departamento: Ingeniería de Sistemas y Computación
Universidad: Universidad del Norte
Repositorio: GitHub
Entregado por:

- Cristian Cubillos.
- David Escorcia.

Introducción

Este proyecto tiene como objetivo implementar el esquema de firma digital **LUOV (Lifted Unbalanced Oil Vinegar)**. La meta es comprender y desarrollar el **Algoritmo 6 (Sign)**, adaptándolo para los parámetros obligatorios y opcionales indicados en el enunciado.

Este avance muestra el progreso en la implementación del código y explica los métodos utilizados para resolver cada uno de los pasos del algoritmo de generación de firmas.

Esquema LUOV y Funcionamiento del Algoritmo de Firma

LUOV se basa en la estructura Oil-Vinegar, dividiendo las variables en dos grupos y combinándolas para crear sistemas de ecuaciones polinómicas complejas, difíciles de resolver sin la clave privada. El **Algoritmo 6 (Sign)** genera una firma única para un mensaje dado, resolviendo un sistema de ecuaciones y aplicando un proceso iterativo hasta obtener una solución válida.

La implementación se diseñó para soportar los conjuntos de parámetros requeridos:

- **Obligatorios:** LUOV-7-57-197, LUOV-7-83-283, LUOV-7-110-374
- **Opcionales:** LUOV-47-42-182, LUOV-61-60-261, LUOV-79-76-341

Código Implementado en `sign.py`

La implementación se realizó en Python, usando librerías como `NumPy` y `hashlib` para realizar operaciones matemáticas y de hashing. A continuación, se describe el flujo principal del código de generación de firmas:

Función Principal: `Sign`

La función `Sign` toma como entrada una `private_seed` y un `message`, y realiza los siguientes pasos:

1. **Generación de Semilla Pública y Matriz T:** Se obtiene una semilla pública y una matriz (T) a partir de la `private_seed` utilizando la función `generate_public_seed_and_T`.
2. **Generación de C, L y Q1:** A partir de la semilla pública, se calculan las matrices necesarias para el esquema Oil-Vinegar con la función `G`, que corresponde a la función comprimida pública.
3. **Generación de Salt Aleatorio y Hash:** Se genera un `salt` aleatorio de 16 bytes y se calcula el hash `h` del mensaje concatenado con un byte nulo y el salt. Esto asegura que cada firma generada sea única, incluso si el mensaje es el mismo.
4. **Bucle de Resolución:**
   - En un bucle, se generan valores aleatorios para las variables de Vinegar.
   - Se construye la matriz aumentada `A` mediante la función `BuildAugmentedMatrix`.
   - Aplicamos **eliminación gaussiana** con `GaussianElimination` para intentar resolver el sistema. Si se obtiene una solución, el bucle se rompe y se almacena en `s0`.
5. **Cálculo de la Firma Final `s`:** La firma final `s` se calcula combinando los vectores de Oil y Vinegar, y aplicando transformaciones con la matriz (T).
6. **Retorno de la Firma y Salt:** La función devuelve la firma `s` y el `salt` generado, necesarios para la verificación posterior.

Fragmento del Código Principal de Firma

```
def Sign(private_seed, message):
    public_seed, T = generate_public_seed_and_T(private_seed)
    C, L, Q1 = G(public_seed)
    salt = RandomBytes(16)
    h = Hash(message + b"\x00" + salt, salt)

    while True:
        V = RandomBytes(r * v // 8)
        A = BuildAugmentedMatrix(C, L, Q1, T, h, V)
        solution = GaussianElimination(A)

        if solution is not None:
            s0 = np.concatenate((V, solution))
            break
```

```
    s = np.dot(np.hstack((np.eye(v.shape[0]), -T)), s0)
    return s, salt
```

Descripción de las Funciones de Apoyo

- **`generate_public_seed_and_T`** : Genera la semilla pública y la matriz (T) a partir de la semilla privada mediante un hash SHA-256.
- **`BuildAugmentedMatrix`** : Crea la matriz aumentada (A) usando (C), (L), (Q1), (T), y el vector hash (h).
- **`GaussianElimination`** : Resuelve el sistema lineal utilizando eliminación gaussiana.

Librerías Utilizadas

Para esta implementación, se usaron las siguientes librerías:

- **NumPy**: Para operaciones de matriz y álgebra lineal.
- **Hashlib**: Para realizar hashing SHA-256, crucial para generar semilla y hashes seguros.

Estas herramientas nos han permitido cumplir con los requisitos criptográficos y matemáticos del algoritmo LUOV.

Conclusión y Próximos Pasos

El código cumple con la generación de firmas utilizando LUOV, adaptado para los parámetros requeridos. Aún se realizarán pruebas para asegurar la integridad de las firmas y se documentarán más a fondo los métodos.

Fecha de entrega: 2 de noviembre de 2024.