**Fundamentos de RSA**

RSA es un algoritmo de cifrado **asimétrico**, lo cual significa que utiliza dos claves distintas para cifrar y descifrar:

1. **Clave pública**: Usada para cifrar datos. Es segura para compartirse públicamente.
2. **Clave privada**: Usada para descifrar datos. Debe mantenerse en secreto para evitar que otros puedan descifrar la información.

**¿Cómo Funciona el Cifrado y Descifrado RSA?**

1. Se generan un par de llaves (pública y privada).
2. Para cifrar un mensaje, se utiliza la clave pública.
3. Para descifrar el mensaje, se utiliza la clave privada.

**Paso 1: Generación de Llaves (Primer Archivo)**

Primero, necesitamos un par de llaves que permitan el cifrado y descifrado.

1. **Generación del Par de Llaves:** Usamos RSA.generate(2048) para crear un par de llaves de 2048 bits (tamaño estándar recomendado para seguridad actual).
2. **Exportación de la Llave Privada:** Guardamos la llave privada en un archivo seguro llamado llave\_privada.pem. La extensión .pem indica un archivo que contiene datos encriptados o claves criptográficas.
3. **Exportación de la Llave Pública:** Guardamos la llave pública en un archivo llamado llave\_publica.pem, que se puede compartir públicamente para que otros cifren mensajes.

Primer archivo generadorDeLlaves.py

Captura de pantalla con letras

Descripción generada automáticamente

**Paso 2: Cifrado del Mensaje (Segundo Archivo)**

Aquí es donde el mensaje se cifra utilizando la clave pública. Este proceso asegura que solo quien tenga la clave privada podrá descifrar el mensaje.

1. **Leer el Mensaje Original:** El mensaje se obtiene del archivo mensaje.txt.
2. **Cargar la Clave Pública:** Cargamos la clave pública desde el archivo llave\_publica.pem usando RSA.import\_key.
3. **Inicializar el Cifrador RSA:** Utilizamos el cifrador PKCS1\_OAEP, un esquema de cifrado seguro que añade relleno al mensaje para prevenir ataques de criptoanálisis.
4. **Cifrar el Mensaje:** El mensaje en texto plano se cifra con la clave pública.
5. **Guardar el Mensaje Cifrado:** El mensaje cifrado se almacena en un archivo binario llamado mensaje\_cifrado.bin.

Segundo Archivo (cifradoDeMensaje.py)

Texto

Descripción generada automáticamente

**Paso 3: Descifrado del Mensaje (Tercer Archivo)**

En este paso, desciframos el mensaje cifrado usando la clave privada para recuperar el texto original.

1. **Cargar la Clave Privada:** Cargamos la clave privada desde el archivo llave\_privada.pem.
2. **Leer el Mensaje Cifrado:** Se obtiene el mensaje cifrado desde el archivo mensaje\_cifrado.bin.
3. **Inicializar el Descifrador RSA:** Igual que en el cifrado, utilizamos PKCS1\_OAEP para descifrar el mensaje.
4. **Descifrar el Mensaje:** Aplicamos la clave privada para transformar el mensaje cifrado de vuelta a texto plano.
5. **Guardar el Mensaje Descifrado:** Finalmente, guardamos el mensaje descifrado en un archivo llamado mensaje\_descifrado.txt.

Texto

Descripción generada automáticamente

**Comprobación de uso de archivos**

1. Primero ejecutamos **Python generadorDeLLaves.py** para generar las claves publicas y privadad
2. Segundo ejecutamos **Python cifradoDeMensaje.py** para cifrar el mensaje en un archivo.bin
3. Tercero ejecutamos **Python descifradoMensaje.py** para descifrar el mensaje y copiarlo a un nuevo archivo TXT

Texto

Descripción generada automáticamente

Empecemos por el principio. RSA es uno de los primeros sistemas de criptografía de clave pública. Se usa ampliamente para asegurar la transmisión de datos, como en el caso que vimos del archivo que me mostraste.

**¿Cómo funciona RSA?**

RSA se basa en la dificultad de factorizar números grandes. En términos simples, se utilizan dos claves: una pública para cifrar y una privada para descifrar.

* **Clave pública:** Se puede compartir con cualquiera.
* **Clave privada:** Debe mantenerse secreta.

Si alguien quiere enviarte un mensaje secreto, usa tu clave pública para cifrarlo. Solo tú, con tu clave privada, puedes descifrarlo.

**¿Por qué 2048?**

El "2048" en RSA.generate(2048) se refiere a la longitud de la clave en bits. Esto determina el tamaño de los números primos que se utilizan para generar las claves. Cuanto mayor sea la longitud de la clave, más difícil será factorizarla y, por lo tanto, más seguro será el cifrado. 2048 bits es un tamaño de clave comúnmente utilizado que ofrece un buen equilibrio entre seguridad y rendimiento.

**¿Y qué es PKCS1\_OAEP?**

PKCS1\_OAEP (Optimal Asymmetric Encryption Padding) es un esquema de relleno que se utiliza junto con RSA para mejorar la seguridad.

Imagina que quieres enviar un mensaje corto. Si simplemente lo ciframos con RSA, podría ser vulnerable a ciertos ataques. PKCS1\_OAEP añade un relleno aleatorio al mensaje antes de cifrarlo, lo que hace que sea mucho más difícil de descifrar sin la clave privada.

En resumen, RSA es un algoritmo de cifrado que utiliza claves públicas y privadas para asegurar la información. 2048 bits es una longitud de clave común que ofrece un buen nivel de seguridad. PKCS1\_OAEP es un esquema de relleno que se utiliza para mejorar la seguridad de RSA.

Espero que esto te ayude a comprender mejor el archivo que me mostraste. Si tienes más preguntas, no dudes en preguntar.