

Sistemas de Acceso y Transmisión de Datos

Sistemas de Acceso y Transmisión de Datos	1
Página 19. Ejemplo Python CRC	2
Página 21. Ejemplo Python SHA256	2
Página. 22 Protocolo Diffie-Hellman	3
Página 32. Propiedades de la Seguridad	4
Ejercicio 1.1. Confidencialidad	4
Ejercicio 1.2. Integridad	7
Ejercicio 1.3. Autenticidad	8
Ejercicio 1.5. Aplicación práctica	13
Página 44. Clave privada y pública	15
Ejercicio 1.6: Cifrado y Descifrado Simétrico (Cifrado César)	15
Ejercicio 1.7: Comparación entre Criptografía Simétrica y Asimétrica	19
Ejercicio 1.8: Aplicaciones Prácticas de Criptografía	19
Ejercicio 1.9: Resolviendo un Problema de Seguridad	20
Página 58. Ejemplo Clave Pública y Privada	21
Página 63. Ejemplo Digital Signature Algorithm	22
Página 65. Protocolos de intercambio de claves	23
Ejercicio 1.10 Protocolo Diffie-Hellman	23
Ejercicio 1.11. RSA y su papel en el intercambio de claves	26
Ejercicio 1.12. ECDH (Elliptic-Curve Diffie-Hellman)	28
Ejercicio 1.13. DSA (Digital Signature Algorithm)	29
Ejercicio 1.14. Comparación de protocolos	32
Ejercicio 1.15. Aplicaciones y casos de uso	34
Página 74. Ejercicio Python RSA	37
Página 87. Ejercicio Python SHA-256-1	37
Página 90. Funciones Hash	38
Ejercicio 1.16. Comprensión de Conceptos Básicos	38
Ejercicio 1.17. Aplicación Práctica de Funciones Hash	39
Ejercicio 1.18. Evaluación de la Seguridad de Algoritmos Hash.	40
Ejercicio 1.19. Caso de Estudio Integrador de Datos	42
Ejercicio 1.20. Creación de una Función Hash Personalizada.	43
Página 94. Requerimientos Legales de firma electrónica (Ley 59/2003)	48
Ejercicio 1.21. Debate sobre Firmas Electrónicas vs. Manuscritas	48
Página 104. La Firma digital	50
Ejercicio 1.22. Concepto de Firma Digital	50
Ejercicio 1.23. Tipos de Firma Digital	51
Ejercicio 1.24. Uso y Validación de Firmas Digitales	53
Ejercicio 1.25. Ventajas y Desventajas de las Firmas Digitales	56
Página 114. Protocolos de Intercambio de Claves	57

Ejercicio 1.26: Diferencias entre Cifrado de Flujo y Cifrado de Bloque	57
Ejercicio 1.27: Análisis de Casos de Uso (Video en Tiempo Real)	58
Ejercicio 1.28: Comparación de Protocolos de Intercambio de Claves	59
Ejercicio 1.29: Aplicación práctica de ECDH	60
Prueba Práctica de ECDH (Intercambio de Claves)	62
Ejercicio 1.30: Implementación y seguridad en protocolos de intercambio de claves	66
Página 117. Ejercicio Herramientas GPG Online (Windows)	72
Página 131. Herramientas de cifrado	78
Ejemplo Práctico: Uso de Luks en Linux	78
Ejercicio 1.31: Cifrado y firma de archivos con GPG	81
Ejercicio 1.32: Configuración de GPG en un cliente de correo electrónico	85

Página 19. Ejemplo Python CRC

```

EjercicioCRC.py x EjercicioSHA256.py 1 EjercicioDiffieHellman.py
Pruebas > EjercicioCRC.py > ...
56 # =====
57
58 message = "Hola"
59 generator_polynomial = "1101" # Polinomio generador de ejemplo (CRC-3)
60
61 # Codificamos el mensaje
62 encoded_message, remainder = crc_encode(message, generator_polynomial)
63 print(f"Mensaje original: {message}")
64 print(f"Polinomio generador: {generator_polynomial}")
65 print(f"Mensaje codificado CRC: {encoded_message}")
66 print(f"Residuo (CRC): {remainder}")
67
68 # Simulación de recepción del mensaje (correcto)
69 received_message = encoded_message
70 is_valid = crc_verify(received_message, generator_polynomial)
71 print(f"\nMensaje recibido (sin errores): {received_message}")
72 print(f"¿Válido?: {is_valid}")
73
74 # Simulación de recepción del mensaje con error
75 # Introducimos un error: invertimos un bit
76 received_message_with_error = list(encoded_message)

```

PROBLEMS 30 OUTPUT DEBUG CONSOLE TERMINAL PORTS Filter

```

[Running] python -u "c:\Users\2-DAW\Desktop\Proyectos\Pruebas\EjercicioCRC.py"
Mensaje original: Hola
Polinomio generador: 1101
Mensaje codificado CRC: 01001000011011110110110001100001110
Residuo (CRC): 110

Mensaje recibido (sin errores): 01001000011011110110110001100001110
❖❖lido?: True

Mensaje recibido con error: 01011000011011110110110001100001110
❖❖lido?: False

[Done] exited with code=0 in 0.075 seconds

```

Página 21. Ejemplo Python SHA256

```
EjercicioCRC.py 9 EjercicioSHA256.py 1 X EjercicioDiffieHellman.py
Pruebas > EjercicioSHA256.py > ...
1 # Cómo funciona SHA-256
2
3 import hashlib # Importamos la librería para usar funciones hash
4
5 # 1. Función para obtener el hash de un mensaje usando SHA-256
6 def calculate_sha256_hash(message):
7     # Crear un objeto sha256
8     sha256 = hashlib.sha256()
9     # Convertir el mensaje a bytes y actualizar el objeto sha256
10    sha256.update(message.encode('utf-8'))
11    # Devolver el hash en formato hexadecimal
12    return sha256.hexdigest()
13
14 # 2. Demostración con un mensaje
15 message = "Hola, este es un ejemplo de hash con SHA-256."
16 hash_result = calculate_sha256_hash(message)
17
18 # 3. Mostramos el resultado
19 print("Mensaje original:", message)
20 print("Hash con SHA-256:", hash_result)
21
PROBLEMS 30 OUTPUT DEBUG CONSOLE TERMINAL PORTS Filter
❖❖❖lido?: True

Mensaje recibido con error: 010110000110111101101100011000011110
❖❖❖lido?: False

[Done] exited with code=0 in 0.075 seconds

[Running] python -u "c:\Users\2-DAW\Desktop\Proyectos\Pruebas\EjercicioSHA256.py"
Mensaje original: Hola, este es un ejemplo de hash con SHA-256.
Hash con SHA-256: 584cb55df37685b0e4ccfa765b9b4a4eebc6874d5f4d3a8299a964878eb8a010

Mensaje modificado: Hola, este es un ejemplo de has con SHA-256.
Hash con SHA-256 (modificado): 9221e7e950cdef1e0dc4c916730709493d32337f9ceb628a7132da3429736f8e
|
[Done] exited with code=0 in 0.081 seconds
```

Página. 22 Protocolo Diffie-Hellman

```
EjercicioDiffieHellman.py X
Pruebas > EjercicioDiffieHellman.py > ...
1  # Protocolo Diffie-Hellman
2
3  import random
4
5  # Paso 1: Selección de parámetros públicos (p y g)
6  p = 23 # Un número primo público
7  g = 5  # Un generador público (base)
8
9  print(f"Parámetros públicos:\n p = {p}\n g = {g}\n")
10
11 # Paso 2: Cada usuario selecciona su clave privada
12 # Usuario A elige su clave privada (secreto)
13 a = random.randint(1, p-1)
14 print(f"Usuario A selecciona su clave privada (secreto) a = {a}")
15
16 # Usuario B elige su clave privada (secreto)
17 b = random.randint(1, p-1)
18 print(f"Usuario B selecciona su clave privada (secreto) b = {b}")
19
20 # Paso 3: Cálculo de claves públicas
21 # Usuario A calcula su clave pública
22 A_pub = (g ** a) % p
23
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS Filter
[Running] python -u "c:\Users\2-DAW\Desktop\Proyectos\Pruebas\EjercicioDiffieHellman.py"
Parámetros públicos:
p = 23
g = 5

Usuario A selecciona su clave privada (secreto) a = 12
Usuario B selecciona su clave privada (secreto) b = 8
Usuario A calcula su clave pública: A_pub = 18
Usuario B calcula su clave pública: B_pub = 16

Usuario A calcula la clave compartida: K_A = 16
Usuario B calcula la clave compartida: K_B = 16

El intercambio de claves ha sido exitoso! Clave compartida: 16

[Done] exited with code=0 in 0.08 seconds
```

Página 32. Propiedades de la Seguridad

Ejercicio 1.1. Confidencialidad

Supongamos que trabajas en una empresa que se dedica al desarrollo de software. se te ha encomendado la tarea de asegurar la confidencialidad de datos personales que se almacenan en la bd de la empresa. Explica como implementarías un sistema para proteger esos datos utilizando técnicas criptográficas. ¿qué tipo de cifrado recomendarías y por qué?

1. Objetivo: Garantizar la Confidencialidad

La confidencialidad significa que **solo las personas y sistemas autorizados** puedan acceder a los datos personales almacenados en la base de datos (BD).

Para lograrlo, debemos **cifrar la información sensible**, tanto **en reposo** (dentro de la BD o backups) como **en tránsito** (entre aplicaciones y servidores).

2. Tipos de cifrado aplicables

a) Cifrado simétrico (clave única)

- Utiliza la **misma clave** para cifrar y descifrar los datos.
- Algoritmos recomendados:
 - **AES** con clave de 256 bits (AES-256).
 - Modo de operación segura: **GCM** (AES-GCM, que utiliza Initialization Vector). Aun con una misma clave, usando IV diferentes el resultado de la encriptación es del mismo modo diferente.
- Ventajas: rápido y eficiente para grandes volúmenes de datos.
- Desventaja: la **gestión de la clave** (almacenamiento, rotación, acceso) es crítica.

b) Cifrado asimétrico (par de claves pública/privada)

- Ejemplo: RSA, ECC (Elliptic Curve Cryptography).
- Ideal para **intercambio seguro de claves, firmas digitales o autenticación**.
- No es eficiente para cifrar grandes volúmenes de datos directamente.

c) Hashing (para datos no reversibles)

- Para contraseñas u otros datos que **no deben descifrarse nunca**.
 - Algoritmos recomendados: **bcrypt**, **Argon2** o **PBKDF2** (con sal aleatoria y múltiples iteraciones, por ejemplo: PBKDF2(password="123456", salt="A7d9x2", iterations=100000)).
-

3. Diseño de un sistema criptográfico en la empresa

a. Identificar los datos personales

Ejemplo: nombre, dirección, correo electrónico, número de documento, teléfono.

b. Definir el nivel de protección

No todos los campos requieren el mismo nivel de cifrado.

Por ejemplo:

- Contraseñas → **hash + sal**
- DNI, teléfono, correo → **cifrado AES-256**
- Datos estadísticos → quizás podría aplicarse la **anonimización**, pero hay que tener en cuenta que este proceso es irreversible y no se ajusta al cumplimiento normativo (GPDR, etc).

c. Implementación técnica

1. **Cifrado en reposo (en la base de datos):**
 - Crear una capa de cifrado en la aplicación (por ejemplo, en la API que escribe en la BD). Hay que revisar todas las aplicaciones y aplicar esta capa a todas las APIs que escriban/lean a/de BD.
 - Usar **AES-256-GCM** para cifrar antes de guardar los datos, y para descifrar después de leer los datos de BD.
 - Guardar el IV (vector de inicialización) junto con los datos cifrados.
 2. **Gestión segura de claves:**
 - Utilizar un **KMS (Key Management Service)** como AWS KMS o Azure Key Vault.
 - Rotar las claves periódicamente.
 - Limitar el acceso a las claves solo al servicio autorizado.
 3. **Cifrado en tránsito:**
 - Toda comunicación entre cliente-servidor o servicio-BD debe ir sobre **TLS 1.3** o superior.
 4. **Registro y auditoría:**
 - Registrar cada acceso o intento de descifrado.
 - Monitorizar accesos inusuales.
-

4. Recomendación técnica final

Recomendaría implementar un **sistema híbrido**:

Componente	Técnica recomendada	Justificación
Datos personales en BD	AES-256-GCM (cifrado simétrico)	Alta seguridad, rendimiento excelente, autenticación de integridad.
Contraseñas	bcrypt / Argon2	Evita descifrado, resistente a ataques de fuerza bruta.
Claves de cifrado	Gestión en KMS	Previene exposición de claves en código o BD.
Transmisión de datos	TLS 1.3	Protege datos en tránsito entre sistemas.

Con esta combinación, garantizas **confidencialidad, integridad y control de acceso**, cumpliendo además con normativas como **GDPR o la LOPDGDD**.

Ejercicio 1.2. Integridad

Imagina que envías un archivo importante a un amigo a través de un email. ¿cómo podrías asegurarte de que el archivo no ha sido alterado durante la transmisión? describe un método criptográfico para garantizar la integridad del archivo y explica cómo funciona.

1. Objetivo: Garantizar la integridad del archivo

Cuando envías un archivo por correo electrónico, puede existir riesgo de:

- Corrupción del archivo durante la transmisión.
- Alteración maliciosa por un atacante (ataque "man-in-the-middle").
- Modificación accidental en el servidor intermedio.

Por tanto, necesitamos un método para **detectar cualquier cambio**, aunque sea de un solo bit.

2. Método criptográfico recomendado: Funciones hash + Firma digital

Existen dos enfoques, según el nivel de seguridad que se necesite:

A. Hash criptográfico (básico)

- Generas un **hash (recomendado SHA-256)** del archivo antes de enviarlo.
- Tu amigo genera el **hash del archivo recibido**.
- Si ambos hashes coinciden, el archivo **no ha sido alterado**.

Limitación:

Si alguien modifica el archivo **y también el hash**, el receptor no detectará el cambio. Por eso, en entornos sensibles, se añade **autenticación criptográfica** con una firma digital.

B. Firma digital (integridad + autenticación)

Este método usa **criptografía asimétrica (par de claves pública/privada)**.

Funcionamiento:

1. **Tú (emisor)** tienes una clave privada (solo tuya).
2. Calculas el **hash** del archivo.
3. Cifras ese hash con tu **clave privada** → esto produce la **firma digital**.
4. Envías el archivo + la firma digital a tu amigo.
5. **Tu amigo (receptor):**
 - Calcula el hash del archivo recibido.

- Usa tu **clave pública** para descifrar la firma digital.
- Si ambos hashes coinciden → el archivo **no fue alterado** y proviene de ti.

Beneficios:

- Garantiza **integridad** (el archivo no se modificó).
- Garantiza **autenticidad** (proviene realmente del emisor).
- Evita falsificación o manipulación intermedia.

Comparación de métodos

Método	Garantiza integridad	Garantiza autenticidad	Recomendado para
Hash (SHA-256)	Sí	No	Archivos personales o entornos de baja sensibilidad
Firma digital (RSA/ECC + SHA-256)	Sí	Sí	Comunicaciones seguras, contratos, documentos oficiales

Conclusión

Para asegurar que el archivo **no ha sido alterado durante la transmisión**:

Usa una firma digital basada en hash criptográfico (como SHA-256) y cifrado asimétrico (RSA o ECC).

Este método garantiza tanto la **integridad** del archivo como la **autenticidad del remitente**. En entornos menos críticos, un simple **hash SHA-256** compartido por canal seguro también puede ser suficiente.

Ejercicio 1.3. Autenticidad

Supongamos que trabajas en un equipo de desarrollo que crea software para la banca en línea. ¿Cómo podrías asegurarte de que las transacciones en línea realizadas por los usuarios son auténticas y no han sido alteradas por un atacante? Explica el papel de la autenticidad en este contexto y cómo podrías implementarla utilizando técnicas criptográficas.

1. ¿Qué significa “autenticidad” en este contexto?

Autenticidad significa poder **verificar que algo (una transacción, usuario o mensaje)** proviene **realmente de quien dice ser**, y que **no ha sido modificado** durante la transmisión.

En el caso de la **banca en línea**, esto implica:

- Que la **transacción fue efectivamente enviada por el usuario legítimo** (no por un atacante).
 - Que **nadie la alteró** en el camino (por ejemplo, cambiar el monto o la cuenta destino).
 - Que el **servidor y el cliente se reconocen mutuamente** (autenticación mutua).
-

2. Cómo garantizar la autenticidad con técnicas criptográficas

En caso de un hipotético “**Man In the Middle**”, si el sistema **no verifica la autenticidad ni la integridad**, el banco podría procesar una transacción **falsa**.

Para evitar esto se usan varias capas de seguridad basadas en criptografía, a nivel de usuario, del servidor, del mensaje y de la comunicación:

A. Autenticación del usuario

Asegura que el usuario es quien dice ser.

Técnicas:

1. **Certificados digitales (PKI)** → el cliente tiene un certificado emitido por una autoridad confiable.
2. **Firma digital** → el usuario firma su transacción con su **clave privada**; el banco verifica con la **clave pública**.
3. **Autenticación multifactor (MFA)** → combinación de:
 - Algo que sabes (contraseña o PIN),
 - Algo que tienes (token, app móvil),
 - Algo que eres (huella o rostro).

Resultado: solo el verdadero titular puede generar una transacción válida.

B. Autenticación del servidor

El usuario también necesita saber que **se está conectando al banco real** y no a una Página falsa.

Técnica:

- Uso de **certificados SSL/TLS** emitidos por una **autoridad de certificación confiable** (CA).
→ Por ejemplo, el candado verde del navegador indica que la conexión es segura y auténtica.
-

C. Integridad y autenticidad del mensaje

Una vez que se ha autenticado al usuario y al servidor, hay que garantizar que la transacción **no se altere durante la transmisión**.

Técnicas criptográficas:

1. Firma digital de la transacción

- El usuario firma el contenido (monto, cuenta destino, fecha) con su clave privada.
- El servidor verifica la firma con su clave pública.
- Si algo cambia en el mensaje, la firma ya no es válida.

Ejemplo conceptual:

Firma = Cifrar(Hash(transacción), clave_privada_usuario)

2. Códigos MAC (Message Authentication Code)

- Si el sistema usa claves simétricas, se genera un **HMAC** (por ejemplo, HMAC-SHA256).
 - El servidor recalcula el HMAC y compara. Si difiere, el mensaje fue alterado.
-

D. Cifrado de la comunicación (TLS 1.3)

Incluso si el mensaje está firmado, **debe viajar cifrado** para evitar espionaje o manipulación. Por eso toda comunicación entre el navegador o app móvil y el servidor bancario usa **HTTPS (TLS 1.3)**.

TLS proporciona:

- **Autenticación del servidor y/o cliente**
 - **Confidencialidad** (cifrado)
 - **Integridad y autenticidad** (mediante MAC o AEAD como AES-GCM)
-

3. Flujo simplificado:

1. El usuario inicia sesión con MFA y obtiene su token digital.
2. Realiza la transacción (ej. transferencia).

3. La aplicación bancaria crea un **hash del mensaje** y lo **firma digitalmente** con la clave privada del usuario.
4. La transacción firmada se envía cifrada por **TLS 1.3** al servidor.
5. El servidor verifica:
 - La firma digital (**autenticidad e integridad**).
 - El certificado del cliente (**autenticación del usuario**).
6. Si todo es correcto, procesa la transacción.

Elemento protegido	Técnica criptográfica usada	Propósito
Usuario legítimo	Certificado digital / Firma / MFA	Autenticación
Transacción no alterada	Firma digital o HMAC	Integridad + Autenticidad
Canal de comunicación	TLS 1.3 (AES-GCM + certificados)	Confidencialidad + autenticidad del servidor
Claves privadas	Almacenadas en HSM o token seguro (DNI electrónico, token..)	Protección contra robo

Conclusión

El papel de la **autenticidad** en la banca en línea es **garantizar que las transacciones provienen del usuario real y no fueron manipuladas**.

Esto se logra combinando varias capas de criptografía:

Certificados digitales + Firmas digitales + Cifrado TLS + HMAC o AES-GCM

Con este esquema, incluso si un atacante intercepta el tráfico, **no puede modificar ni falsificar una transacción válida**.

Ejercicio 1.4. No Repudio, imputabilidad y sellado de tiempos

Considera un escenario donde se firma digitalmente un contrato entre dos partes. ¿Qué papel juegan el no repudio, la imputabilidad y el sellado de tiempos en este proceso? Explica como cada una de estas propiedades protege la validez del contrato firmado.

1. Contexto

Cuando dos partes firman digitalmente un contrato, queremos garantizar que:

1. La firma es **válida y auténtica** (proviene de cada parte).
2. Nadie pueda **negar que firmó el contrato**.
3. El momento de la firma quede **registrado y verificable**.
4. Se pueda **imputar responsabilidades legales** si hay conflictos.

Para esto entran en juego:

- **No repudio**
 - **Imputabilidad**
 - **Sellado de tiempos (timestamping)**
-

2. No repudio

El no repudio garantiza que **el firmante no pueda negar haber firmado un documento digital**.

Cómo se logra criptográficamente:

- Cada parte firma el documento con su **clave privada**.
- La firma se puede verificar con su **clave pública**, garantizando la autenticidad.
- Como solo el titular posee la clave privada, no puede decir:

“Yo no firmé esto”.

Ejemplo: Firma = cifrar(hash(documento), clave_privada_emisor)

Si alguien verifica la firma con la clave pública del firmante y coincide, queda **probado que la persona firmó el documento**.

Protección del contrato: Evita que una parte **niegue haber firmado** o tratado de invalidar la firma.

3. Imputabilidad

La imputabilidad es la capacidad de **asignar claramente una acción o firma a una persona concreta**, vinculándola legalmente a la acción realizada.

Cómo se logra:

1. Las firmas digitales están asociadas a **identidades verificadas** mediante certificados digitales emitidos por una Autoridad de Certificación (CA).
2. Cada certificado contiene información del titular: nombre, correo, empresa, etc.
3. Esto permite que cualquier firma pueda **asignarse inequívocamente a un individuo**.

Protección del contrato:

- En caso de disputa, se puede **determinar qué persona firmó el contrato**.
 - La responsabilidad legal de cada acción queda clara.
-

4. Sellado de tiempos (timestamping)

El sellado de tiempos agrega un **registro confiable de la fecha y hora exacta** en que se firmó un documento.

Cómo se logra:

- Se usa un **Servicio de Sellado de Tiempos (TSA, Time Stamping Authority)** confiable.
- La TSA aplica su propia firma digital sobre el **hash del documento + la marca de tiempo**.
- Esto garantiza que el documento **existía en ese momento exacto** y que la firma se realizó antes de un cierto instante.

Protección del contrato:

- Permite probar que la firma se realizó **en un momento específico**.
 - Evita disputas sobre cuándo se firmó el contrato (por ejemplo, para cumplir plazos legales o regulatorios).
-

5. Resultado: Flujo de firma digital con estas propiedades:

1. El firmante calcula el **hash del contrato**.
2. Firma digitalmente el hash con su **clave privada** → garantiza **autenticidad y no repudio**.
3. Se envía el documento al **servicio de sellado de tiempo (TSA)** → añade un **timestamp firmado** → garantiza el **momento exacto de la firma**.
4. El certificado digital del firmante vincula la firma a **su identidad** → garantiza **imputabilidad**.

Estas tres propiedades combinadas **protegen la validez legal del contrato**, asegurando que sea **auténtico, verificable y vinculante** ante cualquier disputa.

Ejercicio 1.5. Aplicación práctica

Desarrolla un breve caso práctico donde se combinen las propiedades de confidencialidad, integridad, autenticidad, no repudio, imputabilidad y sellado de tiempos. Describe cómo implementarías un sistema que utilice todas estas propiedades en un entorno de comercio electrónico.

Caso práctico: Compra de un producto en línea

Escenario:

Un cliente realiza una compra de un artículo caro (por ejemplo, un ordenador) en una tienda de comercio electrónico. Queremos garantizar que:

1. Los datos del cliente y la transacción no puedan ser vistos por terceros (**confidencialidad**).
 2. La información no se modifique durante la transmisión (**integridad**).
 3. El cliente y la tienda sean quienes dicen ser (**autenticidad**).
 4. El cliente no pueda negar que realizó la compra (**no repudio**).
 5. La transacción se pueda atribuir inequívocamente al cliente (**imputabilidad**).
 6. La fecha y hora de la compra quede registrada (**sellado de tiempos**).
-

Implementación del sistema

1. Confidencialidad

- Toda la comunicación entre cliente y servidor se realiza mediante **TLS 1.3** (HTTPS), usando **AES-GCM** para cifrado simétrico autenticado.
 - Los datos sensibles almacenados en la base de datos (número de tarjeta, dirección, etc.) se cifran en reposo usando **AES-256-GCM** con claves gestionadas por un **KMS** (**Key Management Service**).
-

2. Integridad

- Cada transacción se convierte en un **hash criptográfico (SHA-256)** antes de ser almacenada.
 - Además, se puede generar un **HMAC** (con una clave secreta compartida por el servidor) que asegura que el contenido de la transacción **no fue alterado**.
-

3. Autenticidad

- **El cliente se autentica** mediante **certificado digital o MFA** (usuario + token/app).
 - **El servidor de la tienda** se autentica mediante certificado **SSL/TLS** para que el cliente se conecte al sitio legítimo.
-

4. No repudio

- El cliente **firma digitalmente la transacción** con su **clave privada** (incluyendo monto, producto, fecha y hora).
- Esta firma garantiza que **no puede negar haber realizado la compra**.

5. Imputabilidad

- La firma digital está vinculada a un **certificado digital emitido por una autoridad confiable**, que incluye la identidad del cliente.
 - Esto permite **atribuir legalmente la transacción a un cliente específico**.
-

6. Sellado de tiempos

- La transacción firmada se envía a un **servicio de sellado de tiempos (TSA)**.
 - La TSA genera un **timestamp firmado** que queda registrado en la transacción, asegurando que la compra se realizó en un **momento exacto** y verificable.
-

Flujo completo de la transacción

1. **Cliente selecciona el producto** y envía los datos a la tienda mediante **TLS 1.3** → **confidencialidad**.
 2. La tienda genera un **hash de la transacción** y un **HMAC** → **integridad**.
 3. El cliente firma digitalmente el hash con su **clave privada** → **autenticidad + no repudio + imputabilidad**.
 4. La transacción firmada se envía al **TSA** → se agrega un **timestamp firmado** → **sellado de tiempos**.
 5. El servidor almacena la transacción cifrada en la base de datos → **confidencialidad y trazabilidad**.
-

Conclusión

Con este sistema, una transacción de comercio electrónico queda **totalmente protegida** desde el punto de vista de seguridad:

- Nadie puede **leer** ni **modificar** los datos.
- Se puede **verificar que la transacción provino del cliente legítimo**.
- Se puede **atribuir legalmente la compra** a un individuo.
- Se registra **cuándo ocurrió la compra**, con validez legal.

Página 44. Clave privada y pública

Ejercicio 1.6: Cifrado y Descifrado Simétrico (Cifrado César)

Este ejercicio utiliza el Cifrado César, un ejemplo clásico de criptografía simétrica por sustitución.

Parte 1: Cifrar el Mensaje "SECRETO" con K=3

- Concepto de Clave (K)

En criptografía, la **clave** (K) es una pieza de información secreta (un número, una palabra o una secuencia de bits) que, combinada con un algoritmo de cifrado, transforma el mensaje original (texto plano) en el mensaje cifrado (criptograma).

Significado de K=3 en el Cifrado César

En el contexto cifrado César, K=3 significa:

1. **Tipo de Clave:** Es la clave secreta compartida entre el emisor y el receptor (criptografía **simétrica**).
2. **Valor de Desplazamiento:** El número **3** indica cuántas posiciones se debe **desplazar** cada letra del mensaje original hacia adelante en el alfabeto para cifrar, y hacia atrás para descifrar.

Ejemplo Práctico

- **Cifrado (Desplazar hacia adelante):**

Si la letra es **A** y usamos K=3, la letra cifrada será **D** (A -B - C - D).

- **Descifrado (Desplazar hacia atrás):**

Si la letra cifrada es **D** y K=3, la letra original es **A** (D - C - B - A).

1. Tabla de asignación de números al alfabeto:

Letra	Número (P)	Letra	Número (P)	Letra	Número (P)
A	1	J	10	S	19
B	2	K	11	T	20
C	3	L	12	U	21
D	4	M	13	V	22

E	5	N	14	W	23
F	6	O	15	X	24
G	7	P	16	Y	25
H	8	Q	17	Z	26
I	9	R	18		

2. Conversión "SECRETO" a números:

Letra	S	E	C	R	E	T	O
Posición (P)	19	5	3	18	5	20	16

3. Aplicación la Fórmula de Cifrado:

Fórmula de Cifrado César (desplazamiento):

$$C = (P + K) \bmod N$$

Donde N es el tamaño del alfabeto (usaremos $N=26$). Si el resultado es 0, lo consideramos 26 (Z).

Letra original (P)	19+3	5+3	3+3	18+3	5+3	20+3	16+3
Resultado	22	8	6	21	8	23	19

Letra cifrada (C)	V	H	F	U	H	W	S
-------------------	---	---	---	---	---	---	---

Respuesta a) ¿Cuál es el mensaje cifrado resultante?

- El mensaje cifrado es: **VHFUHWS**

Parte 2: Descifrar y Verificar el Mensaje con **K=3**

1. Aplicar la Fórmula de Descifrado

La fórmula para descifrar es:

$$P = (C - K) \bmod 26$$

Letra cifrada (C)	V (22)	H (8)	F (6)	U (21)	H (8)	W (23)	S (19)
C - 3	22-3	8-3	6-3	21-3	8-3	23-3	19-3
Resultado	19	5	3	18	5	20	16
Letra original (P)	S	E	C	R	E	T	O

Respuesta b) Descifra el mensaje cifrado y verifica si recuperas el mensaje original.

- El mensaje descifrado es **SECRETO**. Se recupera el mensaje original, verificando el proceso de cifrado y descifrado siendo estos correctos.

Ejercicio 1.7: Comparación entre Criptografía Simétrica y Asimétrica

Conceptos Clave:

- Simétrica (Clave Secreta Única): Usa la misma clave para cifrar y descifrar.
- Asimétrica (Par de Claves): Usa dos claves diferentes (una pública para cifrar y una privada para descifrar).

Característica	Criptografía Simétrica	Criptografía Asimétrica
Uso de claves	Se usa una sola clave (secreta) para cifrar y descifrar.	Se usa un par de claves (pública para cifrar, privada para descifrar).
Velocidad del cifrado/descifrado	Rápida (algoritmos más sencillos).	Lenta (algoritmos matemáticamente más complejos).
Seguridad frente a ataques	Alta, pero depende de la seguridad de la clave (si se filtra, la comunicación está comprometida).	Muy alta, protege la identidad y el no repudio (quien firma es quien dice ser).
Ejemplo de algoritmo	AES, DES, Twofish, Cifrado César.	RSA, ECC, ElGamal.

Ejercicio 1.8: Aplicaciones Prácticas de Criptografía

Este ejercicio evalúa cuándo es más apropiado usar la criptografía simétrica, asimétrica o una combinación.

Explicación:

- Simétrica: Se usa para confidencialidad y velocidad (cifrar muchos datos rápidamente).
- Asimétrica: Se usa para intercambio seguro de claves, firmas digitales, y autenticación (certificar la identidad).
- Combinación: La mayoría de sistemas modernos usan un híbrido (Asimétrica para intercambiar la clave secreta y Simétrica para cifrar los datos).

a) Una empresa quiere asegurar la transmisión de datos en tiempo real entre sus servidores, garantizando rapidez y seguridad.

- **Respuesta: Criptografía Simétrica (o un sistema Híbrido)**
 - Justificación: La rapidez es clave para la transmisión en tiempo real. La criptografía simétrica (como **AES**) es miles de veces más rápida que la asimétrica y es ideal para cifrar grandes volúmenes de datos de manera eficiente, garantizando la confidencialidad y la integridad. Un sistema híbrido usaría Asimétrica solo para el primer intercambio de la clave simétrica.

(**AES**) transforma bloques de datos de 128 bits (16 bytes) mediante una serie de sustituciones, permutaciones y transformaciones matemáticas a lo largo de varias "rondas", dependiendo del tamaño de la clave. El resultado es un criptograma que no tiene sentido sin la clave secreta correcta.

b) Un usuario quiere firmar digitalmente un contrato electrónico para asegurar que solo él pudo haber creado la firma.

- **Respuesta: Criptografía Asimétrica**
 - Justificación: La firma digital asegura el no repudio (no se puede negar haber firmado) y la autenticidad. Para esto, el usuario usa su clave privada (sólo él la tiene) para cifrar un **hash** del documento, y cualquier persona puede verificarlo usando su clave pública correspondiente.

c) Una aplicación de mensajería quiere asegurar que los mensajes enviados entre usuarios no puedan ser leídos por terceros y además verificar la identidad de los usuarios.

- **Respuesta: Una Combinación (Sistema Híbrido: Simétrico + Asimétrico)**
 - Justificación:
 - Simétrica (e.g., AES): Se usa para cifrar y descifrar el contenido de los mensajes (confidencialidad) debido a su alta velocidad.
 - Asimétrica (e.g., RSA/ECC): Se usa para verificar la identidad de los usuarios y para intercambiar de forma segura la clave simétrica que se usará para la conversación.

Ejercicio 1.9: Resolviendo un Problema de Seguridad

Este ejercicio aborda un problema de seguridad común en la criptografía simétrica: el uso de claves débiles o cortas.

a) ¿Qué problemas de seguridad puede causar el uso de claves cortas en criptografía simétrica?

Problema Principal: Vulnerabilidad al Ataque de Fuerza Bruta

- **Explicación:** Una clave corta tiene un **espacio de claves** (el número total de claves posibles) muy pequeño.

- Por ejemplo, una clave de 8 bits solo tiene $2^8 = 256$ combinaciones. Una computadora moderna podría probarlas todas en una fracción de segundo.
- El atacante simplemente prueba **todas las combinaciones posibles de claves** hasta que encuentra la correcta que descifra el mensaje. Cuanto más larga sea la clave (e.g., 128 bits o 256 bits), el número de combinaciones es tan astronómico que el ataque de fuerza bruta se vuelve inviable.

b) Proporciona dos soluciones que podrías implementar para mejorar la seguridad sin cambiar completamente el sistema de cifrado.

Las soluciones deben mejorar la seguridad sin reemplazar completamente el algoritmo de cifrado simétrico (como pasar de AES a RSA, lo cual sería un cambio total).

1. Aumentar la Longitud de la Clave Secreta

- **Implementación:** Si el sistema actual usa una clave de 56 bits (como el antiguo DES), **migrar a una clave de 128 o 256 bits** (como AES) es el cambio más efectivo.
- **Impacto:** Esto incrementa el espacio de claves de forma exponencial, haciendo que un ataque de fuerza bruta sea **computacionalmente imposible** con la tecnología actual.

2. Usar Salt (Sal) e Iteraciones en la Derivación de Claves (Key Stretching)

- **Implementación:** Si la clave secreta se genera a partir de una contraseña de usuario:
 - **Salt:** Añadir una cadena de datos aleatoria y única (el "salt") a la contraseña antes de aplicar la función de *hashing* para crear la clave de cifrado.
 - **Iteraciones:** Repetir el proceso de *hashing* (iterar) cientos o miles de veces para ralentizar intencionalmente el proceso.
- **Impacto:** Esto dificulta enormemente los ataques que usan **tablas precalculadas** (como *Rainbow Tables*) y los ataques de **fuerza bruta en GPU**, ya que cada intento de adivinar la contraseña se vuelve miles de veces más lento.

```
EjercicioDiffieHellman.py  EjercicioClavePublicaPrivada.py X
Pruebas > EjercicioClavePublicaPrivada.py > ...
1 # Parámetros públicos
2 p = 23 # Número primo
3 g = 5 # Generador
4
5 # Claves privadas elegidas por Alicia y Benito
6 a = 6 # Clave privada de Alicia
7 b = 15 # Clave privada de Benito
8
9 # Cálculo de las claves públicas
10 # Alicia envía A a Benito
11 A = (g ** a) % p
12 print(f"Clave pública de Alicia (A): {A}")
13
14 # Benito envía B a Alicia
15 B = (g ** b) % p
16 print(f"Clave pública de Benito (B): {B}")
17
18 # Cálculo de la clave compartida
19 # Alicia calcula la clave compartida S usando la clave pública de Benito
20 S_Alicia = (B ** a) % p
21 print(f"Clave compartida calculada por Alicia (S_Alicia): {S_Alicia}")
22
23 # Benito calcula la clave compartida S usando la clave pública de Alicia
24 S_Benito = (A ** b) % p
25 print(f"Clave compartida calculada por Benito (S_Benito): {S_Benito}")

PROBLEMS 11 OUTPUT DEBUG CONSOLE TERMINAL PORTS Filter
[Done] exited with code=0 in 0.083 seconds

[Running] python -u "c:\Users\2-DAM\Desktop\Proyectos\Pruebas\EjercicioClavePublicaPrivada.py"
Clave pública de Alicia (A): 8
Clave pública de Benito (B): 19
Clave compartida calculada por Alicia (S_Alicia): 2
Clave compartida calculada por Benito (S_Benito): 2
¡Éxito, la clave compartida es: 2

[Done] exited with code=0 in 0.061 seconds
```

Página 63. Ejemplo Digital Signature Algorithm

```
EjercicioDigitalSignatureAlgorithm.py X
Pruebas > EjercicioDigitalSignatureAlgorithm.py > ...
1 # Importamos las Bibliotecas necesarias
2 from Crypto.PublicKey import DSA
3 from Crypto.Signature import DSS
4 from Crypto.Hash import SHA256
5
6 # 1. Generación de claves
7
8 # Alicia genera su par de claves
9 key_alicia = DSA.generate(2048)
10
11 # Benito genera su par de claves
12 key_benito = DSA.generate(2048)
13
14 # 2. Alicia firma un mensaje
15
16 # Definimos el mensaje que Alicia quiere enviar
17 mensaje = b"Hola Benito, este mensaje esta firmado por Alicia."
18
19 # Alicia genera el hash del mensaje
20 hash_mensaje = SHA256.new(mensaje)
21
22 # Alicia firma el hash del mensaje usando su clave privada
23 firmante = DSS.new(key_alicia, 'Fips-186-3')
24 firma = firmante.sign(hash_mensaje)
25
[Done] exited with code=0 in 2.952 seconds

[Running] python -u "c:\Users\Z-DMA\Desktop\Proyectos\Pruebas\EjercicioDigitalSignatureAlgorithm.py"
Mensaje enviado por Alicia: b'Hola Benito, este mensaje esta firmado por Alicia.'
Firma generada por Alicia: 9e76f51c58115352da66569064305f37f5cd2e5ca1f542ae82a8ad21d4665da2f137dd13da0669b67283f46e7fa889688a7c9574475a8b58
La firma es válida. El mensaje es auténtico y fue enviado por Alicia.
Mensaje enviado por Benito: b'Hola Alicia, confirmo que he recibido tu mensaje.'
Firma generada por Benito: 7e429d88a8875e2ea51e8fcd8252669583f14be61b3a8d4627498f76e57a373678fec7c5d434b2a24fdd2efa84fee264f52f8caf8c4b20f
La firma de Benito es válida. El mensaje es auténtico y fue enviado por Benito.
[Done] exited with code=0 in 3.255 seconds
```

Página 65. Protocolos de intercambio de claves

Ejercicio 1.10 Protocolo Diffie-Hellman

1. Explica con tus propias palabras cómo el protocolo Diffie-Hellman permite que dos partes generen una clave secreta compartida sin necesidad de transmitir la clave directamente.

El protocolo **Diffie-Hellman** permite a dos partes **generar de manera segura una clave compartida secreta sobre un canal público, sin que un tercero pueda descubrirla**. Se fundamenta en operaciones matemáticas de exponenciación modular que **son fáciles de calcular en un sentido, pero difíciles de invertir**.

2. Dado un grupo primo con $p=23$ y un generador $g=5$:

Ø Alicia elige una clave privada $a = 6$

Ø Benito elige una clave privada $b = 15$

Ø Calcula las claves públicas PA Y PB que ambos compartirán.

Ø Determina la clave secreta compartida que ambos calculan.

1 Calcular las claves públicas

La fórmula para la clave pública es:

$$P = g^{\text{privada}} \mod p$$

Alicia:

$$P_A = g^a \mod p = 5^6 \mod 23$$

Calculamos paso a paso:

1. $5^2 = 25 \equiv 2 \mod 23$
2. $5^4 = (5^2)^2 = 2^2 = 4 \mod 23$
3. $5^6 = 5^4 \cdot 5^2 = 4 \cdot 2 = 8 \mod 23$

✓ Entonces:

$$P_A = 8$$

Benito:

$$P_B = g^b \mod p = 5^{15} \mod 23$$

Usando potencias reducidas:

1. $5^1 = 5$
2. $5^2 = 25 \equiv 2$
3. $5^4 = 4$
4. $5^8 = (5^4)^2 = 16 \mod 23$
5. $5^{15} = 5^8 \cdot 5^4 \cdot 5^2 \cdot 5^1 = 16 \cdot 4 \cdot 2 \cdot 5$

Calculamos paso a paso módulo 23:

- $16 \cdot 4 = 64 \equiv 18 \mod 23$
- $18 \cdot 2 = 36 \equiv 13 \mod 23$
- $13 \cdot 5 = 65 \equiv 19 \mod 23$



Calculamos paso a paso módulo 23:

- $16 \cdot 4 = 64 \equiv 18 \pmod{23}$
- $18 \cdot 2 = 36 \equiv 13 \pmod{23}$
- $13 \cdot 5 = 65 \equiv 19 \pmod{23}$

✓ Entonces:

$$P_B = 19$$

2 Calcular la clave secreta compartida

La clave compartida se calcula así:

$$K = P_{\text{otro}}^{\text{propia}} \pmod{p}$$

Clave de Alicia:

$$K = P_B^a \pmod{p} = 19^6 \pmod{23}$$

Calculamos usando exponentes sucesivos:

- $19^2 = 361 \equiv 16 \pmod{23}$
- $19^4 = (19^2)^2 = 16^2 = 256 \equiv 3 \pmod{23}$
- $19^6 = 19^4 \cdot 19^2 = 3 \cdot 16 = 48 \equiv 2 \pmod{23}$

Clave de Benito:

$$K = P_A^b \pmod{p} = 8^{15} \pmod{23}$$

Usando potencias sucesivas:

- $8^2 = 64 \equiv 18 \pmod{23}$
- $8^4 = 18^2 = 324 \equiv 2 \pmod{23}$
- $8^8 = 2^2 = 4 \pmod{23}$
- $8^{15} = 8^8 \cdot 8^4 \cdot 8^2 \cdot 8^1 = 4 \cdot 2 \cdot 18 \cdot 8$

Calculamos paso a paso módulo 23:

- $4 \cdot 2 = 8$
- $8 \cdot 18 = 144 \equiv 6 \pmod{23}$
- $6 \cdot 8 = 48 \equiv 2 \pmod{23}$



✓ Coincide con Alicia.

3 Resultado final

- Claves públicas:

$$P_A = 8, \quad P_B = 19$$

- Clave secreta compartida:

$$K = 2$$

Ejercicio 1.11. RSA y su papel en el intercambio de claves

- 1.Cuál es la diferencia principal entre cómo se intercambian claves en RSA en comparación con Diffie-Hellman?

La **diferencia principal** es cómo se gestionan las claves:

- **RSA:** utiliza **clave pública y privada**; la clave pública se comparte abiertamente y cualquiera puede cifrar mensajes para el propietario de la clave privada, quien es el único que puede descifrarlos. El intercambio de claves no requiere que ambas partes contribuyan a la generación de la clave.
- **Diffie-Hellman:** **no envía directamente la clave secreta**; ambas partes contribuyen con secretos privados y usan información pública para **generar conjuntamente una clave compartida**, que nunca se transmite tal cual por el canal.

En resumen: **RSA distribuye una clave pública, Diffie-Hellman genera una clave secreta compartida entre ambos sin enviarla.**

2. Considera el siguiente par de claves RSA:

1. Clave pública: $e=7$, $n=55$

2. Clave privada: $d = 23$

3. Supón que Alicia quiere enviar un mensaje privado a Benito para que él puede descifrarlo usando su clave privada.

4. Paso 1: Alicia cifra el mensaje ($M=8$) usando la clave pública de Benito.

5. Paso 2: Benito, usando su propia clave privada, descifra el mensaje cifrado por Alicia.

Datos

- Clave pública de Benito: $e = 7, n = 55$
 - Clave privada de Benito: $d = 23$
 - Mensaje de Alicia: $M = 8$
-

Paso 1: Cifrar el mensaje (Alicia → Benito)

La fórmula de cifrado RSA es:

$$C = M^e \mod n$$

Sustituimos los valores:

$$C = 8^7 \mod 55$$

Calculamos paso a paso módulo 55:

1. $8^2 = 64 \equiv 9 \mod 55$
2. $8^4 = 9^2 = 81 \equiv 26 \mod 55$
3. $8^7 = 8^4 \cdot 8^2 \cdot 8^1 = 26 \cdot 9 \cdot 8$

Calculamos módulo 55 paso a paso:

- $26 \cdot 9 = 234 \equiv 234 - 4 \cdot 55 = 234 - 220 = 14 \mod 55$
- $14 \cdot 8 = 112 \equiv 112 - 2 \cdot 55 = 112 - 110 = 2 \mod 55$

✅ Mensaje cifrado:

$$C = 2$$

Paso 2: Descifrar el mensaje (Benito usando su clave privada)

La fórmula de descifrado RSA es:

$$M = C^d \pmod n$$

Sustituimos:

$$M = 2^{23} \pmod{55}$$

Calculamos usando exponenciación rápida:

1. $2^1 = 2$
2. $2^2 = 4$
3. $2^4 = 16$
4. $2^8 = 16^2 = 256 \equiv 36 \pmod{55}$
5. $2^{16} = 36^2 = 1296 \equiv 31 \pmod{55}$

Ahora descomponemos 23 en potencias de 2: $23 = 16 + 4 + 2 + 1$

$$2^{23} = 2^{16} \cdot 2^4 \cdot 2^2 \cdot 2^1 \equiv 31 \cdot 16 \cdot 4 \cdot 2 \pmod{55}$$

Multiplicamos paso a paso módulo 55:

1. $31 \cdot 16 = 496 \equiv 496 - 9 \cdot 55 = 496 - 495 = 1 \pmod{55}$
2. $1 \cdot 4 = 4 \pmod{55}$
3. $4 \cdot 2 = 8 \pmod{55}$

✅ Mensaje descifrado:

$$M = 8$$

Resultado final

- Mensaje cifrado enviado por Alicia: $C = 2$
- Mensaje descifrado por Benito: $M = 8$

El mensaje original se recupera correctamente usando la clave privada de Benito.

Ejercicio 1.12. ECDH (Elliptic-Curve Diffie-Hellman)

¿Por qué se considera que ECDH es más eficiente que Diffie-Hellman en términos de la seguridad que ofrece con claves más cortas?

La razón principal es que **ECDH (Elliptic Curve Diffie-Hellman)** usa **curvas elípticas** en lugar de la aritmética modular sobre enteros grandes como en Diffie-Hellman clásico. Esto permite:

1. **Mayor seguridad con claves más cortas:**

- Por ejemplo, una clave de 256 bits en ECDH ofrece una seguridad equivalente a una clave de 3072 bits en DH clásico.
- Esto se debe a que el problema del **logaritmo discreto en curvas elípticas** es mucho más difícil de resolver que el logaritmo discreto en enteros grandes.

2. Menor uso de recursos computacionales y memoria:

Al usar claves más cortas, se requieren menos operaciones aritméticas, menos ancho de banda y menos almacenamiento, lo que hace ECDH más eficiente especialmente en dispositivos con recursos limitados.

En resumen: **ECDH es más eficiente porque logra la misma seguridad con claves mucho más cortas**, lo que reduce tiempo de cálculo y consumo de recursos.

Ejercicio 1.13. DSA (Digital Signature Algorithm)

1. Explica cómo el DSA garantiza que un mensaje no ha sido modificado y proviene realmente del remitente indicado.

El **DSA (Digital Signature Algorithm)** garantiza autenticidad e integridad mediante **firmas digitales**:

1. Autenticidad del remitente:

- El remitente genera una firma usando su **clave privada** sobre un resumen (hash) del mensaje.
- Solo el propietario de la clave privada pudo crear esa firma.

2. Integridad del mensaje:

- El receptor verifica la firma usando la **clave pública** del remitente.
- Si el mensaje se modifica, aunque sea un bit, la verificación falla porque el hash del mensaje ya no coincide con la firma.

En resumen: DSA asegura que el mensaje **proviene del remitente correcto** y que **no ha sido alterado** durante la transmisión.

2. Alicia quiere enviar un mensaje firmado a Benito utilizando DSA:

- Alicia elige una clave privada $x=10$ y un valor $k=3$ aleatorio.
- Se tiene la base $g=2$, $p=23$ y $q=11$

- Paso 1: calcula la clave pública de Alicia
- Paso 2: Alicia genera una firma digital para el mensaje "Hola" cuyo hash es $H(m) = 5$. Calcula los valores de la firma (r,s) .
- Paso 3: Benito recibe el mensaje y la firma. Verifica la firma utilizando la clave pública de Alicia y los valores dados.

Ejemplo1 en Python:

```
Ejercicio DSA.py ×
Pruebas > Ejercicio DSA.py > ...
1  # Datos
2  p = 23
3  q = 11
4  g = 2
5  x = 10      # clave privada de Alicia
6  k = 3      # valor aleatorio
7  H_m = 5     # hash del mensaje
8
9  # Paso 1: clave pública de Alicia
10 y = pow(g, x, p)
11 print(f"Clave pública de Alicia: y = {y}")
12
13 # Paso 2: generar la firma digital (r,s)
14 r = pow(g, k, p) % q
15 # Inverso de k módulo q
16 k_inv = pow(k, -1, q)
17 s = (k_inv * (H_m + x * r)) % q
18 print(f"Firma digital: r = {r}, s = {s}")
19
20 # Paso 3: verificación por Benito
21 w = pow(s, -1, q)
22 u1 = (H_m * w) % q
23 u2 = (r * w) % q
24 v = (pow(g, u1, p) * pow(y, u2, p) % p) % q
25 print(f"Verificación: v = {v}")
26
27 if v == r:
28     print("La firma es válida")
29 else:
30     print("La firma NO es válida")

PROBLEMS 7 OUTPUT DEBUG CONSOLE TERMINAL PORTS

[Running] python -u "c:\Users\2-DAW\Desktop\Proyectos\Pruebas\Ejercicio DSA.py"
Clave pública de Alicia: y = 12
Firma digital: r = 8, s = 10
Verificación: v = 8
La firma es válida

[Done] exited with code=0 in 0.08 seconds
```

Ejemplo 2 en Python:

```

python > firma_digital.py > ...
33
34 s = (k_inv * (H + x * r)) % q
35 print(f"Valores de la firma digital: r = {r}, s = {s}")
36
37 # Paso 3: Verificación de la firma por parte de Benito
38 def verify_signature(H, r, s):
39     if not (0 < r < q and 0 < s < q):
40         return False
41     w = mod_inverse(s, q)
42     u1 = (H * w) % q
43     u2 = (r * w) % q
44     v = ((pow(g, u1, p) * pow(y, u2, p)) % p) % q
45     return v == r
46
47 # Verificamos la firma
48 if verify_signature(H, r, s):
49     print("La firma es válida.")
50 else:
51     print("La firma no es válida.")
52

```

```

[Running] python -u "c:\Dlore python\python\firma_digital.py"
Clave pública de Alicia (y): 6
Valor de r: 7
Valores de la firma digital: r = 7, s = 3
La firma es válida.

[Done] exited with code=0 in 0.214 seconds

```

Ejercicio 1.14. Comparación de protocolos

1. Compara y contrasta los protocolos Diffie-Hellman, RSA y ECDH en términos de eficiencia, seguridad y aplicabilidad en diferentes escenarios.

Protocolo	Eficiencia	Seguridad	Aplicabilidad
Diffie-Hellman clásico (DH)	Requiere operaciones de exponenciación modular con números grandes; lento con claves muy largas.	Seguro si se usan primos grandes; vulnerable a ataques de intermediario (MITM) si no se autentica.	Ideal para generar claves compartidas en canales inseguros; usado en protocolos como TLS.
RSA	Operaciones de potencias modulares con números grandes; más lento que DH para generar claves largas.	Seguridad basada en factorización de enteros grandes; requiere claves largas para mantener seguridad.	Cifrado de mensajes, firma digital, intercambio de claves; muy usado en certificados y correo seguro.
ECDH (Elliptic Curve DH)	Mucho más eficiente que DH clásico: claves más cortas y operaciones más rápidas.	Basado en logaritmo discreto en curvas elípticas; alta seguridad con claves cortas.	Ideal para dispositivos con recursos limitados (móviles, IoT); generación de claves compartidas en TLS y VPNs.

Resumen de diferencias clave

- Eficiencia:** ECDH > DH clásico > RSA (en operación de clave/ intercambio de claves).
- Seguridad por tamaño de clave:** ECDH ofrece seguridad equivalente con claves mucho más cortas que DH y RSA.
- Aplicabilidad:**
 - DH y ECDH → intercambio seguro de claves.
 - RSA → cifrado de mensajes, firma digital y autenticación.
- Vulnerabilidades:**
 - DH y ECDH → requieren autenticación para evitar MITM.
 - RSA → depende de factorización de enteros grandes; vulnerable si claves pequeñas.

2. Dado un escenario donde la potencia computacional es limitada (por ejemplo, en dispositivos móviles), justifica qué protocolo de intercambio de claves sería el más adecuado y por qué.

En un escenario con **potencia computacional limitada**, como dispositivos móviles, el protocolo más adecuado es **ECDH (Elliptic Curve Diffie-Hellman)**.

Justificación:

1. **Mayor eficiencia:**

- ECDH utiliza operaciones sobre curvas elípticas que requieren **mucho menos cálculo** que DH clásico o RSA para generar claves seguras.
- Esto reduce el **consumo de CPU y batería**, lo cual es crítico en dispositivos móviles.

2. **Seguridad con claves cortas:**

- Una clave ECDH de 256 bits ofrece seguridad equivalente a una clave DH de 3072 bits o RSA de 3072 bits.
- Esto permite mantener alta seguridad sin sobrecargar el dispositivo.

3. **Adecuado para intercambio de claves:**

- ECDH permite generar una **clave compartida secreta** sobre un canal inseguro de manera eficiente, que luego se puede usar para cifrado simétrico rápido (como AES).

Conclusión:

Para dispositivos con recursos limitados, **ECDH combina seguridad fuerte y eficiencia**, siendo mucho más práctico que DH clásico o RSA en términos de rendimiento y consumo energético.

Ejercicio 1.15. Aplicaciones y casos de uso

1. ¿En qué situaciones es más apropiado utilizar RSA en lugar de ECDH? ¿Y en qué casos es mejor usar ECDH?

Situaciones para usar RSA

1. **Cifrado de mensajes directamente:**

- RSA permite cifrar mensajes usando la **clave pública** del destinatario, de modo que solo él pueda descifrarlos con su clave privada.

2. **Firma digital y autenticación:**

- RSA se usa ampliamente para **firmas digitales**, certificados digitales y verificación de identidad.

3. **Infraestructura establecida:**

- Muy usado en **certificados X.509**, correos seguros (S/MIME) y protocolos antiguos que ya soportan RSA.
4. **Cuando la eficiencia no es crítica:**
- Ideal en sistemas con suficiente potencia computacional, porque las operaciones de RSA con claves grandes pueden ser lentas.

Situaciones para usar ECDH

1. **Intercambio de claves eficiente:**
 - ECDH genera **claves compartidas** de manera rápida y segura, incluso en dispositivos con recursos limitados.
2. **Dispositivos con baja potencia:**
 - Móviles, IoT y sistemas embebidos se benefician de claves más cortas con la misma seguridad.
3. **Protocolos modernos de cifrado de sesión:**
 - TLS 1.3, VPNs y aplicaciones de mensajería usan ECDH para establecer claves de sesión simétricas.
4. **Escenarios donde solo se necesita generar una clave secreta compartida:**
 - Cuando no se requiere cifrado directo de mensajes ni firma digital.

En resumen:

- **RSA:** mejor para cifrado directo, firma digital y autenticación.
- **ECDH:** mejor para intercambio seguro de claves y eficiencia en dispositivos con recursos limitados, y no se requiere cifrado directo de mensajes.

2. Identifica un caso real en la seguridad informática donde se use cada uno de los siguientes protocolos: Diffie-Hellman, RSA, ECDH y DSA. Describe brevemente cómo se aplican en esos casos.

Caso Diffie-Hellman (DH clásico)

Caso real: Establecimiento de claves en **VPNs** y **TLS** (antes de ECDH).

Aplicación:

- Cliente y servidor generan una **clave compartida secreta** mediante operaciones de exponenciación modular.
 - La clave nunca se transmite directamente; se usa luego para cifrado simétrico de la sesión.
-

Caso RSA

Caso real: Certificados SSL/TLS y correo electrónico seguro (S/MIME).

Aplicación:

- Se utiliza la **clave pública** para cifrar mensajes o datos de sesión, y la **clave privada** para descifrarlos.
 - También permite la **firma digital** de correos o documentos, garantizando autenticidad e integridad.
-

Caso ECDH (Elliptic Curve Diffie-Hellman)

Caso real: Mensajería segura (WhatsApp, Signal) y TLS 1.3.

Aplicación:

- Se genera una **clave compartida** usando operaciones sobre curvas elípticas, que es más eficiente que DH clásico.
 - La clave compartida se usa luego para cifrado simétrico de la sesión, protegiendo la comunicación.
-

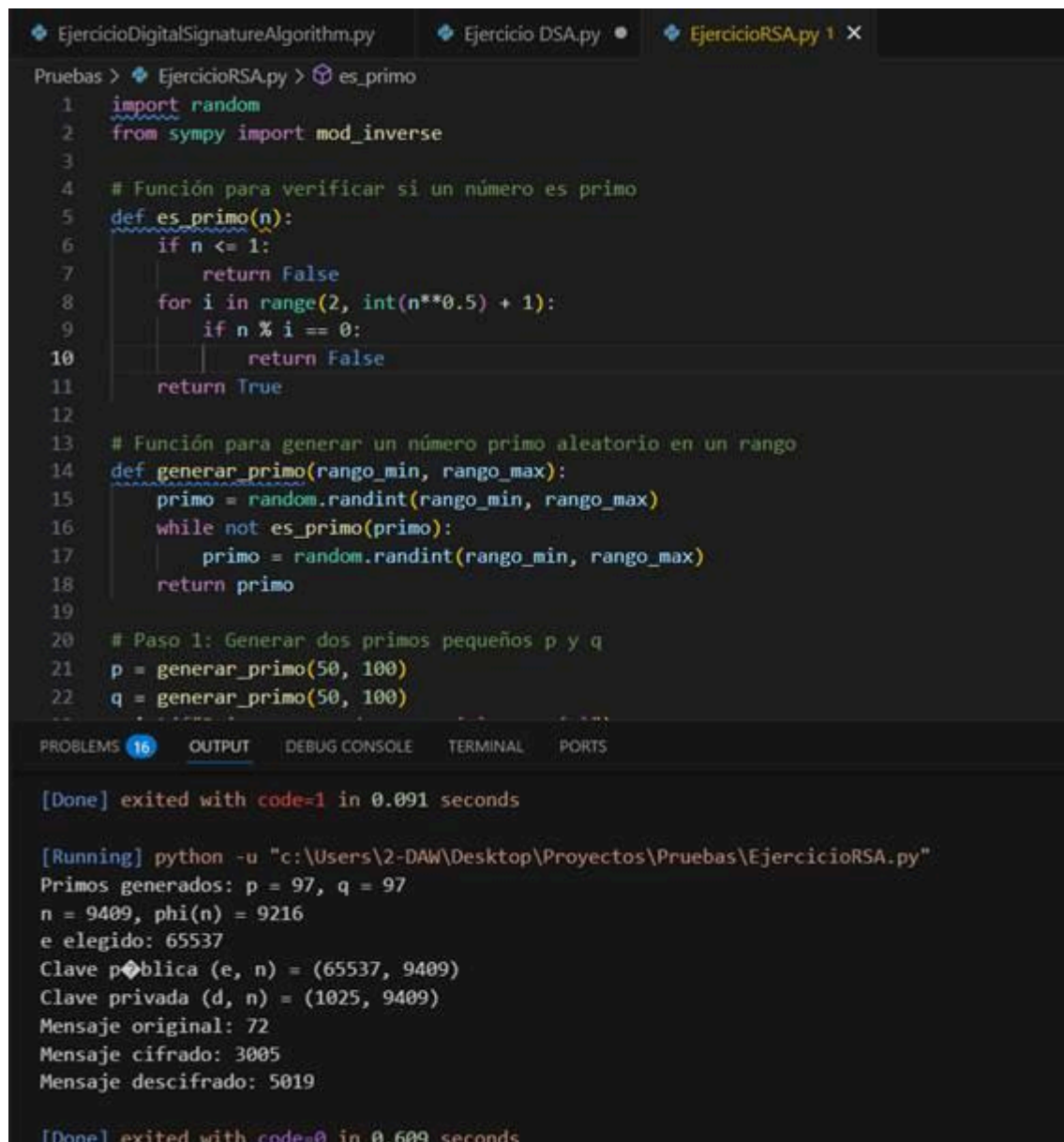
Caso DSA (Digital Signature Algorithm)

Caso real: Firmas digitales en software y documentos electrónicos.

Aplicación:

- El desarrollador firma software o documentos con su **clave privada DSA**.
- El usuario verifica la firma con la **clave pública**, confirmando que el contenido **no ha sido modificado** y proviene del remitente legítimo.
- Ejemplo: paquetes de Linux (.deb, .rpm) firmados digitalmente o documentos legales electrónicos.

Página 74. Ejercicio Python RSA



The screenshot shows a Python IDE with three tabs: 'EjercicioDigitalSignatureAlgorithm.py', 'Ejercicio DSA.py', and 'EjercicioRSA.py 1 X'. The active tab is 'EjercicioRSA.py'. The code defines two functions: `es_primo(n)` to check if a number is prime and `generar_primo(rango_min, rango_max)` to generate a random prime. It then generates two primes `p` and `q`, calculates `n` and `phi(n)`, and chooses an exponent `e`. Finally, it prints the public and private keys, the original message, the encrypted message, and the decrypted message.

```
Pruebas > EjercicioRSA.py > es_primo
1 import random
2 from sympy import mod_inverse
3
4 # Función para verificar si un número es primo
5 def es_primo(n):
6     if n <= 1:
7         return False
8     for i in range(2, int(n**0.5) + 1):
9         if n % i == 0:
10            return False
11    return True
12
13 # Función para generar un número primo aleatorio en un rango
14 def generar_primo(rango_min, rango_max):
15     primo = random.randint(rango_min, rango_max)
16     while not es_primo(primo):
17         primo = random.randint(rango_min, rango_max)
18     return primo
19
20 # Paso 1: Generar dos primos pequeños p y q
21 p = generar_primo(50, 100)
22 q = generar_primo(50, 100)
23
24 n = p * q
25 phi_n = (p - 1) * (q - 1)
26 e = 2
27 while not es_primo(e) or gcd(e, phi_n) != 1:
28     e += 1
29
30 d = mod_inverse(e, phi_n)
31
32 # Claves
33 clave_publica = (e, n)
34 clave_privada = (d, n)
35
36 # Mensaje original
37 mensaje = 72
38
39 # Mensaje cifrado
40 mensaje_cifrado = pow(mensaje, e, n)
41
42 # Mensaje descifrado
43 mensaje_descifrado = pow(mensaje_cifrado, d, n)
44
45 print(f"Primos generados: p = {p}, q = {q}")
46 print(f"n = {n}, phi(n) = {phi_n}")
47 print(f"e elegido: {e}")
48 print(f"Clave pública (e, n) = {clave_publica}")
49 print(f"Clave privada (d, n) = {clave_privada}")
50 print(f"Mensaje original: {mensaje}")
51 print(f"Mensaje cifrado: {mensaje_cifrado}")
52 print(f"Mensaje descifrado: {mensaje_descifrado}")
```

[Done] exited with code=1 in 0.091 seconds

[Running] python -u "c:\Users\2-DAW\Desktop\Proyectos\Pruebas\EjercicioRSA.py"

Primos generados: p = 97, q = 97
n = 9409, phi(n) = 9216
e elegido: 65537
Clave pública (e, n) = (65537, 9409)
Clave privada (d, n) = (1025, 9409)
Mensaje original: 72
Mensaje cifrado: 3005
Mensaje descifrado: 5019

[Done] exited with code=0 in 0.609 seconds

Página 87. Ejercicio Python SHA-256-1

completamente diferente. Si el hash original y el nuevo no coinciden, se confirma que la información no es íntegra (fue modificada).

2. **Almacenamiento Seguro de Contraseñas:** Las bases de datos guardan el **hash** de la contraseña, no la contraseña real. Esto es posible porque la función hash es **irreversible** (o de "sentido único"): es imposible obtener la contraseña original a partir del hash. Así, si la base de datos es robada, las contraseñas reales permanecen protegidas.

Ejercicio 1.17. Aplicación Práctica de Funciones Hash

Considera que tienes un archivo de texto llamado **"datos.txt"** que contiene la frase **"Seguridad Informática 101"**. Utiliza un algoritmo hash como **SHA-256** o **MD5** para generar el resumen criptográfico de este archivo. ¿Cuál es el resultado?

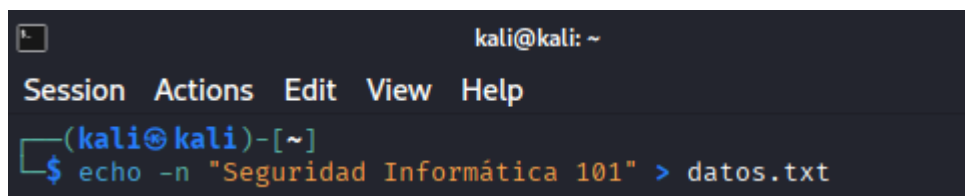
Instrucciones:

1. Usa una herramienta de línea de comandos para generar el hash.
2. Muestra los pasos seguidos para obtener el hash.
3. Compara los resultados obtenidos utilizando SHA-256 y MD5.

- Pasos para Ejecutar en Kali Linux

Paso 1: Crear el Archivo de Entrada

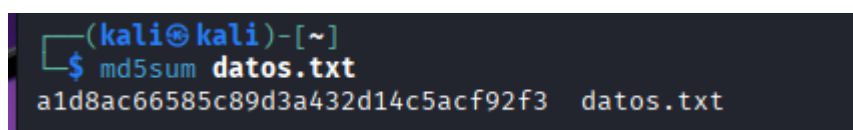
```
echo -n "Seguridad Informática 101" > datos.txt
```

A terminal window with a dark background. The prompt is 'kali@kali: ~'. Below the prompt, there is a menu bar with 'Session', 'Actions', 'Edit', 'View', and 'Help'. The terminal shows the command 'echo -n "Seguridad Informática 101" > datos.txt' being executed. The prompt is '(kali@kali)-[~]' and the command is preceded by a '\$' symbol.

```
kali@kali: ~  
Session Actions Edit View Help  
(kali@kali)-[~]  
$ echo -n "Seguridad Informática 101" > datos.txt
```

Paso 2: Generar el Hash MD5

```
md5sum datos.txt
```

A terminal window with a dark background. The prompt is '(kali@kali)-[~]'. The terminal shows the command 'md5sum datos.txt' being executed. The output is 'a1d8ac66585c89d3a432d14c5acf92f3 datos.txt'. The prompt is preceded by a '\$' symbol.

```
(kali@kali)-[~]  
$ md5sum datos.txt  
a1d8ac66585c89d3a432d14c5acf92f3 datos.txt
```

Paso 3: Generar el Hash SHA-256

```
(kali@kali)-[~]  
$ shasum -a 256 datos.txt  
d55bd8cfb3ae5a1ba55f7889858e251e669728d72551da1992a86c81fbda3577  da  
tos.txt
```

Paso 4: Comparación de Resultados

Algoritmo	Hash Generado	Longitud (Bits)
MD5	a1d8ac66585c89d3a432d14c5acf92f3	128
SHA-256	d55bd8cfb3ae5a1ba55f7889858e251e669728d72551da1992a86c81fbda3577	256

Conclusión y Comparación

La diferencia entre los dos conjuntos de hashes es una demostración práctica del Efecto Avalancha:

- MD5 y SHA-256 son funciones deterministas: Un cambio minúsculo en la entrada (un salto de línea) produce un hash final completamente diferente.
- Comparación: El hash SHA-256 (**d55bd8cfb3ae5a...**) es dos veces más largo que el hash MD5 (**a1d8ac665...**), lo que le confiere una seguridad mucho mayor y una resistencia superior a colisiones. Por ello, SHA-256 es el algoritmo preferido actualmente para la verificación de integridad.

Ejercicio 1.18. Evaluación de la Seguridad de Algoritmos Hash.

¿Por qué se desaconseja el uso de MD5 en aplicaciones críticas de seguridad?

Primeramente entramos en contexto:

- Conceptos Clave de Resistencia

Antes de comparar, definamos los dos tipos de ataques de resistencia:

Ataque	Objetivo	Resistencia (Propiedad)
Ataque de Colisión	Encontrar dos entradas diferentes (M_1 y M_2) que produzcan el mismo hash ($H(M_1) = H(M_2)$).	Resistencia a Colisión
Ataque de Preimagen	Dado un hash (H), encontrar la entrada original (M) que lo produjo, de forma que $H(M) = H$.	Resistencia a Preimagen (Irreversibilidad)

- Comparación de Fortalezas y Debilidades

Característica	SHA-256 (Secure Hash Algorithm 256)	MD5 (Message Digest Algorithm 5)
Longitud del Hash	256 bits (64 caracteres)	128 bits (32 caracteres)
Resistencia a Colisión	Muy Alta (Considerado criptográficamente seguro; es computacionalmente inviable encontrar colisiones).	Baja/Nula (Criptográficamente Roto; se han encontrado métodos rápidos para generar colisiones).

Resistencia a Preimagen	Muy Alta (Irreversible; la única forma práctica es mediante ataque de fuerza bruta).	Alta (Aún se considera difícil de revertir; sin embargo, las colisiones debilitan su uso en general).
Velocidad	Moderada (Más lento que MD5, pero más seguro).	Muy Rápida

En conclusión y teniendo en cuenta el anterior contexto:

MD5 se desaconseja totalmente para verificar la integridad de datos críticos (como firmware, certificados digitales, o para asegurar que un archivo no ha sido manipulado) debido a su falla total en la resistencia a colisiones.

- **El Problema:** La investigación ha demostrado que es factible (relativamente rápido y con hardware común) para un atacante generar dos archivos diferentes que resulten en el mismo hash MD5.
- **El Riesgo (**Suplantación**):** Un atacante podría crear un archivo malicioso (M_{Malo}) y un archivo inofensivo (M_{Bueno}) que comparten el mismo hash. Si el sistema de seguridad solo se verifica el hash MD5, el atacante puede hacer que la víctima descargue el archivo malicioso, y el sistema seguirá confirmando que el hash es correcto (porque coincide con el hash del archivo bueno).
- **Conclusión:** La integridad (la promesa de que los datos no han sido modificados) se rompe, haciendo que MD5 sea inutilizable para fines de seguridad crítica.

Por otro lado, SHA-256 pertenece a la familia SHA-2, que es el estándar actual por su robusta resistencia a ambos tipos de ataques.

Ejercicio 1.19. Caso de Estudio Integrador de Datos

Imagina que trabajas como **administrador de sistemas** en una empresa que debe garantizar la **integridad de los archivos descargados** por los usuarios.

¿Qué algoritmo hash recomendarías utilizar para verificar la integridad de los archivos? Justifica tu respuesta teniendo en cuenta aspectos como la **velocidad**, la **seguridad** y la **popularidad** del algoritmo.

Como administrador de sistemas encargado de la **integridad de archivos descargados**, se recomienda el uso del algoritmo **SHA-256** (Secure Hash Algorithm 256).

Justificación de la Elección

La elección de SHA-256 se justifica priorizando la **seguridad** y la **confianza** sobre cualquier pequeña ganancia de velocidad, garantizando la integridad de los datos de la empresa:

1. **Seguridad (Resistencia Criptográfica):** SHA-256 ofrece una **muy alta resistencia a colisiones** y a ataques de preimagen (irreversibilidad). Esto significa que es prácticamente imposible que un atacante cree un archivo malicioso que produzca la misma "huella digital" que un archivo legítimo. Esto es fundamental, ya que el objetivo principal es asegurar que los archivos **no han sido manipulados**.
2. **Popularidad y Estándar:** SHA-256 forma parte de la familia SHA-2, que es el **estándar actual** a nivel global. Está respaldado por gobiernos, navegadores y la mayoría de los protocolos de seguridad. Utilizarlo asegura **compatibilidad** y facilita la integración con otras herramientas de seguridad y verificación.
3. **Velocidad vs. Riesgo:** Aunque existen algoritmos más rápidos, la velocidad de SHA-256 es **eficiente** en el hardware moderno. La pequeña diferencia de tiempo que se gana con otros algoritmos no justifica el **riesgo inaceptable** de utilizar un algoritmo con vulnerabilidades conocidas en una aplicación de seguridad crítica.

Conclusión: SHA-256 es la opción más robusta y responsable para garantizar que cada archivo descargado sea **íntegro** y que provenga de la fuente esperada.

Ejercicio 1.20. Creación de una Función Hash Personalizada.

- Diseño Conceptual: Función Hash Simple de Dígitos (HSD)

Diseñaremos una función simple que toma un número entero (N) y lo transforma en otro valor de tamaño fijo.

Nombre: **F**unción Hash **S**imple de **D**ígitos (HSD)

Entrada (N): Cualquier número entero positivo (ejemplo: 987654)

Salida (H): Un valor de 3 dígitos (000 a 999).

Algoritmo Conceptual Paso a Paso:

1. **Cuadrado:** Se eleva el número de entrada al cuadrado: $S = N^2$
 - *Ejemplo:* $987654^2 = 975,460,904,016$
2. **Suma y Recorte:** Sumamos todos los dígitos del resultado S.
 - *Ejemplo:* $9+7+5+4+6+0+9+0+4+0+1+6 = 51$
3. **Compresión (Hashing):** Tomamos el resultado de la suma (51) y restamos el número 1. Luego, calcula el **Módulo 1000** (es decir, el resto de dividir entre 1000). Esto garantiza que la salida siempre tenga 3 dígitos.
 - *Ejemplo:* $(51 - 1) \bmod 1000 = 50 \bmod 1000 = **050**$

Paso Conceptual	Operación Matemática	Resultado Intermedio
1. Entrada (N)	Número de entrada	987654
2. Cuadrado (S)	$S = N^2$	$987654^2 = 975,460,904,016$
3. Suma y Recorte	Sumar todos los dígitos de S	$9 + 7 + 5 + 4 + 6 + 0 + 9 + 0 + 4 + 0 + 1 + 6 = 51$
4. Compresión (Hashing)	$(Suma - 1) \bmod 1000$	$(51 - 1) \bmod 1000 = 50 \bmod 1000$
5. Resultado Final (H)	Formato de 3 dígitos	050

Función Hash Simple de Dígitos (HSD) para la entrada 987654 es **050**

(Fácilmente reversible, muchas colisiones)

SHA-256 de Kali

Puesta en práctica:

- Paso 1: Generar la Entrada (N)

```
(kali㉿kali)-[~]  
$ echo "987654" > entrada.txt
```

- Paso 2: Generar el Hash Criptográfico (H)

```
(kali㉿kali)-[~]  
$ sha256sum entrada.txt  
07d8dcc0796a71fd4819c37af5d47ae0b00d335d8204b2e4334244c7d3cf98c4  en  
trada.txt
```

1. Demostración de Irreversibilidad

Una función hash como SHA-256 es una **función unidireccional** o de "calle de sentido único" .

1. **Transformación Destructiva:** El proceso de hashing mezcla y reduce la entrada (987654) aplicando transformaciones matemáticas tan complejas que **destruyen la información original** de una manera intencional.
2. **Imposible de Revertir:** Es trivial ir del número al hash, pero es **computacionalmente imposible** hacer el camino inverso (ir del hash al número). No existe un proceso matemático que pueda "deshacer" esa mezcla.
3. **La Única Opción:** La única manera de encontrar la entrada es adivinar y probar billones de combinaciones (fuerza bruta), lo cual tomaría un tiempo invariablemente largo (miles de millones de años).

Es decir que obtener el hash es fácil; obtener el número original a partir del hash es tan difícil que se considera **imposible** para cualquier computadora actual. Por eso se usa para proteger contraseñas.

2. Demostración de Resistencia a Colisiones

```
(kali㉿kali)-[~]  
$ echo "987655" > entrada_diferente.txt  
  
(kali㉿kali)-[~]  
$ sha256sum entrada_diferente.txt  
53566ead07dfdc448a55a28030309c7dabe38734dd9c13c30bf82312dae2dfed  en  
trada_diferente.txt
```

Comparación de Hashes

Entrada (N)	Hash SHA-256 (H)
Original (987654)	07d8dcc0796a71fd4819c37af5d47ae0b00d335d8204b2e4334244c7d3cf98c4
Nueva (987655)	53566ead07dfdc448a55a28030309c7dabe38734dd9c13c30bf82312dae2df

SHA-256 de Kali: Resultado 07d8dcc0796a71fd . . . (Imposible de revertir, sin colisiones conocidas).

Resultado (Resistencia a Colisiones)

A pesar de que las entradas (987654 y 987655) son casi idénticas, **sus hashes son completamente diferentes** y no comparten ninguna coincidencia visible al principio.

Esto demuestra la propiedad de **Resistencia a Colisiones** (y el **Efecto Avalancha**):

- **Resistencia a Colisiones:** Es increíblemente difícil, o mejor dicho, **computacionalmente imposible**, encontrar dos entradas distintas que generen el mismo hash.
- **Efecto Avalancha:** Un cambio mínimo en la entrada provoca un cambio masivo e impredecible en la salida. Esto asegura que nadie pueda "adivinar" cómo modificar un archivo para obtener un hash predecible.
- **Propiedades que debe tener una buena función Hash:**

Para ser considerada una **buena función hash criptográfica**, nuestro diseño debe aspirar a cumplir con tres propiedades clave, que son las que le dan valor para la seguridad digital:

Propiedad	Descripción

Uniformidad	Los resultados deben distribuirse equitativamente en todo el espacio de salida, sin amontonarse en unos pocos valores .
Irreversibilidad	Debe ser una " calle de sentido único ". Es fácil calcular el hash a partir de la entrada, pero computacionalmente es imposible encontrar la entrada original a partir del hash.
Resistencia a Colisiones	Debe ser imposible encontrar dos entradas diferentes que generen el mismo hash. Un cambio minúsculo en la entrada produce un hash completamente distinto (Efecto Avalancha).

Tabla de Contraste: HSD vs. SHA-256

Característica	HSD	SHA-256
Salida	Fijo (3 dígitos: 000-999)	Fijo (64 caracteres hexadecimales)
Uniformidad	Mala. Los resultados se agrupan en valores bajos.	Excelente. Cubre todo el espacio de salida de 2^{256} .
Irreversibilidad	Nula. Es trivial adivinar la entrada por la baja complejidad.	Fuerte. Computacionalmente imposible de revertir.



Resistencia a Colisiones	Nula. Es muy fácil encontrar entradas que sumen lo mismo.	Fuerte. Colisiones desconocidas y extremadamente improbables.
---------------------------------	------------------------------------------------------------------	----------------------------------------------------------------------

Página 94. Requerimientos Legales de firma electrónica (Ley 59/2003)

Ejercicio 1.21. Debate sobre Firmas Electrónicas vs. Manuscritas

Ventajas y Desventajas de las Firmas Electrónicas

Aspecto	Ventajas de la Firma Electrónica	Desventajas de la Firma Electrónica
Seguridad y Prueba	✓ Mayor Nivel de Prueba: Una firma digital avanzada ofrece integridad (se usa un hash para garantizar que el documento no cambió) e identidad (usa certificados para confirmar quién firmó). Es más difícil de falsificar que una firma a mano.	✗ Dependencia de la Tecnología: Requiere infraestructura (certificados, software, claves criptográficas). Si la tecnología falla o el certificado caduca, la firma puede invalidarse.
Eficiencia y Costo	✓ Velocidad y Comodidad: Permite firmar documentos al instante, sin importar la ubicación geográfica (lo que ahorra mucho tiempo y dinero en impresión y envío).	✗ Curva de Aprendizaje: Los usuarios menos técnicos pueden tener dificultades para entender cómo obtener, usar y proteger su certificado digital.

Integridad	 Garantía de Integridad: La firma digital bloquea el documento. Si se altera un solo carácter después de la firma, esta se invalida automáticamente.	 Riesgos de Clave Privada: Si la clave privada de un usuario es robada o comprometida, un atacante podría firmar documentos en su nombre.
-------------------	-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

¿Pueden las Firmas Electrónicas Reemplazar Completamente a las Manuscritas?

No, aún no pueden reemplazar completamente a las firmas manuscritas en **todos** los contextos legales, pero sí en la gran mayoría.

- Ejemplos y Consideraciones Legales

1. Contextos donde la firma electrónica domina:

En la mayoría de las situaciones cotidianas y comerciales, la firma electrónica (especialmente la **Firma Digital Avanzada** o **Cualificada**, que usa certificados) ya es el estándar y es legalmente superior:

- **Contratos Comerciales (B2B):** Es el método más rápido y seguro. Por ejemplo, una empresa de *software* firma un contrato de servicio con otra empresa a través de una plataforma digital. La ley (como la ley **eIDAS** en Europa o la ley **ESIGN** en EE. UU.) le da el mismo valor legal que una firma manuscrita.
- **Declaraciones de Impuestos:** El gobierno exige una forma de autenticación digital fuerte porque garantiza la identidad del declarante y la inalterabilidad de los datos enviados.

2. Contextos donde la firma manuscrita aún es relevante (las excepciones):

Existen áreas específicas donde las regulaciones o la tradición legal (y la necesidad de certeza física) aún exigen una firma manuscrita o, al menos, requieren una firma electrónica de la más alta categoría:

- **Testamentos y Herencias:** En muchos países, la ley requiere que el testamento sea firmado a mano por el testador y atestiguado por testigos **en presencia física** para evitar fraudes o presiones indebidas. La formalidad del papel es una barrera contra la manipulación digital.
- **Documentos Familiares Específicos:** Aunque esto cambia constantemente, ciertas actas matrimoniales o documentos de adopción pueden requerir la presencia física y la firma húmeda (manuscrita) por razones de derecho tradicional o para confirmar la voluntad en un acto solemne.

- **Documentos ante Notario (Actos de Fe Pública):** Aunque los notarios están adoptando la firma electrónica, el acto de firmar una escritura de propiedad o una hipoteca aún suele requerir la **firma física** o una **Firma Electrónica Cualificada** (el tipo de firma más seguro y con el máximo valor legal, equivalente a la firma manuscrita), y a menudo se hace en presencia del notario.

Conclusión Final:

Si bien las **firmas electrónicas son superiores** en términos de seguridad (*integridad criptográfica*) y eficiencia, no pueden reemplazar el componente **tradicional y presencial** que aún exigen ciertas leyes para proteger los derechos más sensibles de las personas (como la voluntad final o el estado civil). El futuro es electrónico, pero la transición total será lenta y estará marcada por las leyes de cada país.

Página 104. La Firma digital

Ejercicio 1.22. Concepto de Firma Digital

Define el concepto de firma digital y explica cómo se diferencia de una firma manuscrita tradicional. ¿Qué elementos garantiza una firma digital en un documento electrónico?

¿Qué es una Firma Digital?

Una **firma digital** es una técnica criptográfica que se utiliza para garantizar la autenticidad y la integridad de un documento electrónico. No es una imagen de una firma a mano, sino un **código matemático** único que se genera usando la **clave privada** del firmante y un **hash** del documento.

Diferencia con la Firma Manuscrita

Característica	Firma Manuscrita	Firma Digital
Garantía Principal	Autenticación del <i>acto</i> (prueba de que la persona estaba allí y lo aprobó).	Integridad del documento y Autenticación del firmante.

Vinculación	Está vinculada a la persona (rasgos caligráficos).	Está vinculada al contenido (el hash) y a la clave privada del firmante.
Falsificación	Puede ser falsificada por un experto en caligrafía.	Es criptográficamente imposible de falsificar sin la clave privada del firmante.
Detección de Cambios	No detecta si se cambió el texto después de firmar.	Si se cambia un solo carácter del documento, la firma se rompe .

Una firma digital fuerte garantiza tres elementos esenciales de seguridad:

1. **Autenticidad:** Confirma **quién** firmó el documento. Se verifica que la clave privada utilizada para firmar pertenece realmente al firmante declarado.
2. **Integridad:** Garantiza que el documento **no ha sido alterado** desde el momento en que se firmó. Si hay un cambio, el hash de verificación fallará.
3. **No Repudio:** El firmante **no puede negar** haber firmado el documento. Puesto que solo él posee la clave privada, la ley lo responsabiliza por la acción.

Ejercicio 1.23. Tipos de Firma Digital

Clasifica los siguientes escenarios según el tipo de firma digital más adecuado (simple, avanzada, cualificada):

- Un usuario firma un **contrato de arrendamiento en línea**, y se necesita que esta firma tenga la **misma validez legal** que una firma manuscrita.
- Un cliente aprueba una **transacción bancaria** a través de una aplicación móvil.
- Un empleado firma electrónicamente su **asistencia diaria** en el trabajo.

Para clasificar, usaremos la normativa europea **eIDAS** (que es un estándar internacional), donde la seguridad y validez legal crecen de simple a cualificada:

Tipo de Firma	Descripción para Principiantes	Valor Legal
Simple	Cualquier dato electrónico (Ej: escribir tu nombre al final de un email o un <i>checkbox</i> "Acepto").	Bajo (Fácil de impugnar).
Avanzada	Vinculada al firmante y puede detectar cambios. Requiere un certificado digital asociado a la identidad.	Alto (Tiene la misma validez legal que la manuscrita en muchos casos).
Cualificada	Es una Firma Avanzada creada con un dispositivo de creación de firma certificado (como un <i>hardware token</i>) y respaldada por un certificado cualificado.	Máximo (Legalmente equivalente a la firma manuscrita en toda la UE).

Clasificación de Escenarios

Escenario	Clasificación Recomendada	Justificación
Contrato de arrendamiento en línea que necesita la misma validez legal que una firma manuscrita.	Avanzada o Cualificada	Necesita el máximo nivel de no repudio y autenticidad para un documento con implicaciones legales serias (contratos, <i>pólizas</i>).
Un cliente aprueba una transacción bancaria a través de una aplicación móvil.	Avanzada	Requiere una identificación robusta (huella dactilar, clave OTP o <i>token</i> de seguridad), que detecte el fraude .

Un empleado firma electrónicamente su asistencia diaria en el trabajo.	Simple	La importancia legal es baja. Simplemente se requiere verificar la identidad del usuario al momento de registrar el evento, lo cual se logra con un nombre de usuario y contraseña junto a una marca de tiempo.
-------------------------------------------------------------------------------	---------------	------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Ejercicio 1.24. Uso y Validación de Firmas Digitales

Describe el proceso de validación de una firma digital. ¿Qué pasos se deben seguir para verificar que una firma digital es válida? Además, ¿qué rol juegan las autoridades de certificación (CA) en este proceso?

1. Proceso de Validación (Verificación)

La validación de una firma digital es un proceso de dos pasos que utiliza la clave pública del firmante para garantizar la integridad y la autenticidad del documento.

1. Paso 1: Verificar la Integridad del Documento:

- El verificador (el receptor) calcula el **hash** del documento **recibido**.
- Luego, utiliza la **clave pública** del firmante para **descifrar** el hash que está *incrustado* en la firma.
- Si el **hash calculado** es **igual** al **hash descifrado**, se confirma que el documento **no ha sido alterado**.

2. Paso 2: Verificar la Autenticidad del Firmante (y el Certificado):

- El verificador revisa el **certificado digital** del firmante (que contiene la clave pública).
- Se verifica que el certificado **no haya expirado** y que **no haya sido revocado** (cancelado).

Pasos de Verificación:

Fase 1: Creación de las Claves y el Archivo de Entrada

1. Generar una clave privada (la "llave secreta" del firmante)

La clave se guarda en 'private.key'

```
(kali㉿kali)-[~]
$ openssl genrsa -out private.key 2048
```

2. Extraer la clave pública correspondiente (la "llave de candado")

La clave se guarda en 'public.pem'

```
(kali㉿kali)-[~]  
$ openssl rsa -in private.key -outform PEM -pubout -out public.pem  
writing RSA key
```

3. Crear el documento original a firmar

```
(kali㉿kali)-[~]  
$ echo "El contrato de 1000 euros queda aceptado y sellado." > contrato.txt
```

Fase 2: Firmar el Documento (Acción del Firmante)

4. Firmar el documento (crear el hash cifrado)

```
(kali㉿kali)-[~]  
$ echo "— Documento firmado. Ahora verificaremos su validez. —"  
— Documento firmado. Ahora verificaremos su validez. —
```

Fase 3: Verificación de la Integridad (Acción del Verificador)

5. Verificar la firma usando la clave pública

```
(kali㉿kali)-[~]  
$ openssl dgst -sha256 -verify public.pem -signature contrato.sig contrato.txt  
Verified OK
```

PRUEBA 2: Verificación Fallida (Documento Alterado)

Simularemos un ataque de alteración del documento para demostrar la Resistencia a Colisión (Efecto Avalancha).

6. Alterar el documento original (cambiamos "euros" por "dólares")

```
(kali㉿kali)-[~]  
$ echo "El contrato de 1000 dolares queda aceptado y sellado." > contrato.txt
```

7. Intentar verificar la firma con el documento alterado

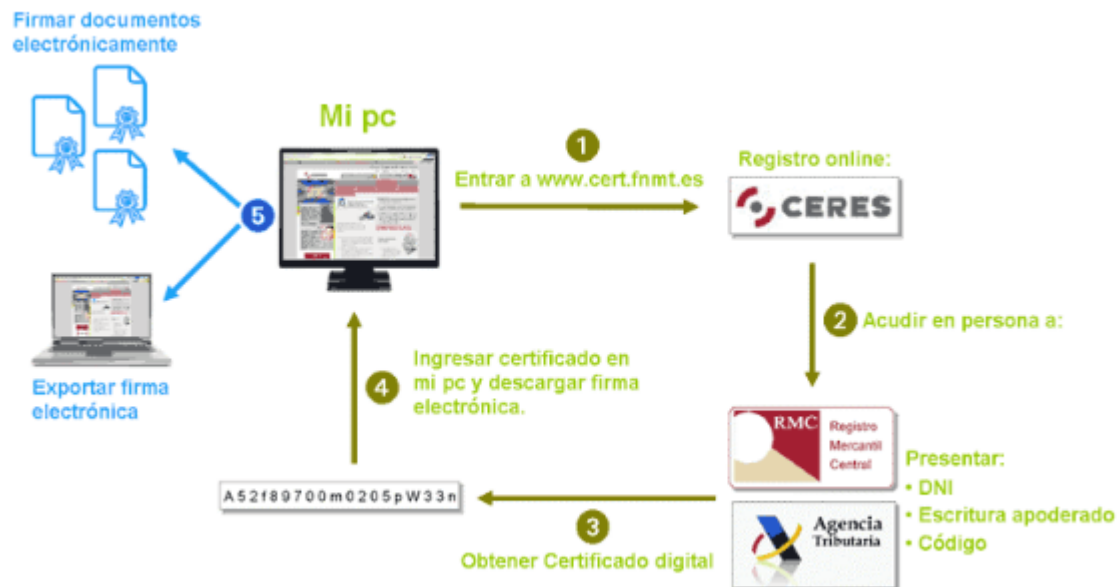
```
(kali㉿kali)-[~]  
$ openssl dgst -sha256 -verify public.pem -signature contrato.sig  
contrato.txt  
Verification failure  
4037A8B72F7F0000:error:02000068:rsa routines:ossl_rsa_verify:bad sig  
nature: ../crypto/rsa/rsa_sign.c:442:  
4037A8B72F7F0000:error:1C880004:Provider routines:rsa_verify_directl  
y:RSA lib: ../providers/implementations/signature/rsa_sig.c:1041:
```

- Un documento firmado exitosamente es validado (Verified OK).
- El más mínimo cambio en el documento rompe la integridad y anula la firma (Verification Failure).

2. El Rol de las Autoridades de Certificación (CA)

Las **Autoridades de Certificación (CA)** juegan un rol de **tercero de confianza** y son fundamentales para la autenticidad.

- **Emisión de Certificados:** La CA es una entidad de confianza (como el gobierno o una empresa acreditada) que verifica la **identidad real** de la persona u organización. Una vez confirmada, la CA emite el **Certificado Digital** (que incluye la clave pública) y lo firma con su propia clave privada.
- **Garantía de autenticidad:** La firma de la CA en el certificado garantiza al mundo que la clave pública efectivamente pertenece a la persona que dice ser. Esencialmente, la CA dice: "Yo, la Autoridad de Confianza, confirmó que esta persona es quien dice ser."
- **Listas de Revocación:** Las CA mantienen una lista pública de certificados que han sido revocados (por robo de clave, cese de actividad, etc.), asegurando que los certificados inválidos no puedan usarse.



Ejercicio 1.25. Ventajas y Desventajas de las Firmas Digitales

Discute las principales ventajas y desventajas de las firmas digitales en comparación con las firmas manuscritas tradicionales. ¿En qué casos podría ser preferible una firma digital?

1. Ventajas Clave (¿Por qué la preferimos?)

- **Seguridad Superior:** La firma digital ofrece una garantía de **integridad inigualable**. Es la única que puede demostrar criptográficamente que un documento no ha sido modificado.
- **Eficiencia:** Permite firmar contratos o documentos con personas al otro lado del mundo **instantáneamente**, eliminando los costos y el tiempo del correo físico.
- **Autenticación Fuerte:** Vincula la acción a una **clave privada única**, haciendo que el **no repudio** sea legalmente mucho más fácil de probar que la simple caligrafía.

2. Desventajas clave (limitaciones)

- **Dependencia Tecnológica:** Requiere **certificados**, *software* y acceso a internet. Un fallo en la infraestructura hace imposible firmar o verificar.
- **Costo Inicial y Mantenimiento:** Obtener y mantener certificados digitales cualificados puede tener un costo, a diferencia de la firma a mano.
- **Problemas de Almacenamiento:** El usuario es responsable de proteger su **clave privada**; si esta se pierde o es robada, el atacante puede firmar en su nombre.

3. Casos donde la firma digital es preferible

La firma digital es **siempre preferible** en contextos que requieren **velocidad, distancia y máxima garantía de integridad** del contenido.

Caso de Uso	Justificación de la Preferencia
Transferencias Bancarias/Financieras	Se requiere una verificación inmediata y no se puede permitir que la orden de Páginao sea alterada ni por un segundo.
Documentación Legal de Larga Duración	Un documento legal que debe durar 30 años (como un título de propiedad) tiene mayor garantía de que no será alterado si se sella digitalmente.
Licitaciones/Contratos a Distancia	Permite a empresas de diferentes países cerrar acuerdos al instante con plena validez legal.

Página 114. Protocolos de Intercambio de Claves

Ejercicio 1.26: Diferencias entre Cifrado de Flujo y Cifrado de Bloque

Explica las principales diferencias entre el cifrado de flujo y el cifrado de bloque. Asegúrate de cubrir los siguientes aspectos:

1. Modo de operación
2. Ventajas y desventajas
3. Ejemplos de algoritmos que usan cada uno de estos métodos.

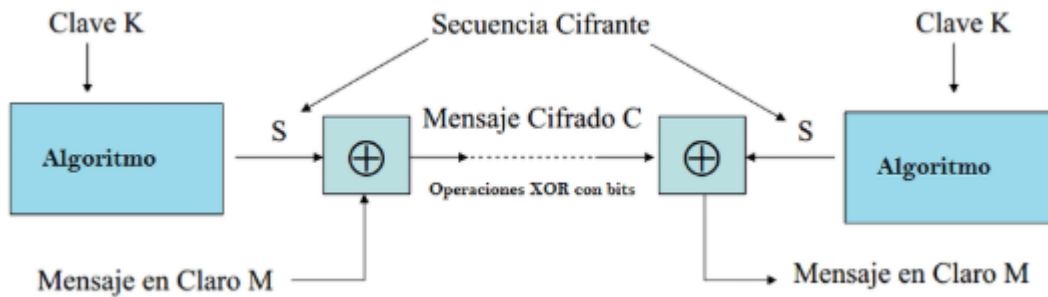
Aspecto	Cifrado de Flujo	Cifrado de Bloque
1. Modo de Operación	Cifra bit a bit o byte a byte .	Cifra datos en bloques de tamaño fijo (ej. 128 bits).
2. Ventaja Principal	Velocidad y Eficiencia (baja latencia).	Seguridad Estructural (mejor para datos fijos).
2. Desventaja Principal	Riesgo de Reutilización de Clave (catastrófico).	Requiere Relleno (Padding, aumenta la sobrecarga).
3. Ejemplos	ChaCha20, Salsa20	AES (Advanced Encryption Standard)

Ejercicio 1.27: Análisis de Casos de Uso (Video en Tiempo Real)

Se presenta un escenario en el cual necesitas asegurar una transmisión en tiempo real de video a través de Internet. ¿Qué tipo de cifrado (de flujo o de bloque) elegirías para proteger la transmisión y por qué?

- Se elegiría el **cifrado de Flujo**. La prioridad es la baja latencia. El cifrado de flujo procesa los datos de forma continua (*bit a bit*), lo cual es ideal para el *streaming* de video, evitando la espera de bloques completos que genera latencia.

https://www.researchgate.net/figure/Figura-1-Eschema-general-del-Cifrado-de-Flujo_fig2_333701648



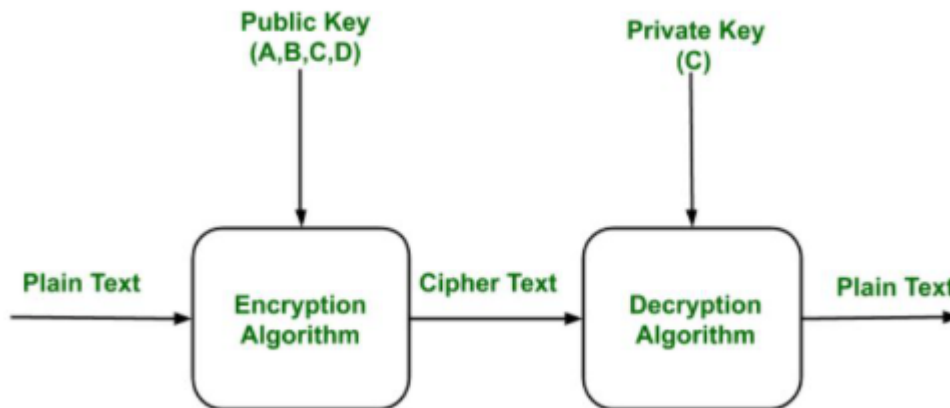
Ejercicio 1.28: Comparación de Protocolos de Intercambio de Claves

Compara los protocolos Diffie-Hellman y RSA en términos de:

1. Mecanismo de generación de claves
2. Uso de claves públicas y privadas
3. Seguridad y resistencia frente a ataques
4. Casos de uso más comunes.

Aspecto	Diffie-Hellman (DH)	RSA
Generación	Basado en el Logaritmo Discreto .	Basado en la Factorización de Primos (ej. $N = p \times q$).
Uso de Claves	Solo para acordar una clave secreta (Intercambio).	Cifrado, Descifrado y Firmas Digitales .
Seguridad	Ofrece Secreto Adelantado Perfecto (PFS) .	La seguridad se basa en la dificultad de factorizar.

Casos de Uso	Establecimiento de Sesiones Seguras (en TLS/VPN).	Cifrado de Datos/Certificados Digitales (Firma de CA).
---------------------	----------------------------------------------------------	---------------------------------------------------------------



<https://www.geeksforgeeks.org/computer-networks/difference-between-diffie-hellman-and-rsa/>

Análisis del Diagrama:

1. **RSA (Lado Izquierdo):** El diagrama muestra un proceso directo de cifrado y descifrado de datos. La Clave Pública se usa para cifrar un mensaje, y la Clave Privada correspondiente se usa para descifrarlo. RSA puede proteger el contenido del mensaje.
2. **Diffie-Hellman (Lado Derecho):** El diagrama muestra un proceso de intercambio de claves. El objetivo NO es cifrar un mensaje completo, sino que dos partes (Alice y Bob) colaboren para generar y acordar una clave secreta simétrica compartida (**K**) a través de un canal inseguro.

Conclusión:

- **DH:** Su uso es la generación de la llave maestra. Solo se usa para intercambio de claves.
- **RSA:** Su uso es la protección del contenido. Se usa para cifrar datos o crear/verificar firmas.

Ejercicio 1.29: Aplicación práctica de ECDH

Supón que estás implementando un sistema de comunicación seguro para un dispositivo IoT con recursos limitados. Explica por qué elegirías el protocolo ECDH para el intercambio de claves y describe cómo funcionaría.

1. ¿Por Qué Elegir ECDH (Curva Elíptica Diffie-Hellman)?

Elegiría ECDH sobre RSA o el Diffie-Hellman (DH) estándar por su Eficiencia Criptográfica Superior .

Ventaja Clave	Impacto Directo en Dispositivos IoT
Tamaño de Clave Reducido	ECDH logra el mismo nivel de seguridad que RSA, pero con claves significativamente más cortas (ej. 256 bits en ECDH es tan seguro como 3072 bits en RSA).
Bajo Consumo de Energía	Las operaciones de Curva Elíptica requieren menos cálculos computacionales , lo que se traduce en un menor consumo de batería y CPU.
Alta Velocidad	El intercambio de claves es mucho más rápido que con RSA o DH grandes, ideal para sesiones rápidas o intermitentes.

Funcionamiento de ECDH (El Intercambio "Liviano")

ECDH permite a dos partes acordar una clave simétrica secreta a través de un canal público e inseguro.

Paso	Parte A (Dispositivo IoT)	Parte B (Servidor)	Función en la Seguridad
1. Secreto Local	Elige un número privado/secreto d_A .	Elige un número privado/secreto d_B .	Base de la seguridad; este número nunca se comparte.
2. Clave Pública	Calcula el Punto Público : $P_A = d_A \times G$.	Calcula el Punto Público : $P_B = d_B \times G$.	G es un punto base de la curva conocida. Calcular el punto es fácil; revertirlo (encontrar d) es difícil.
3. Intercambio	Envía: P_A a B.	Envía: P_B a A.	Intercambian los puntos públicos.
4. Clave Compartida	Clave Compartida: $K = d_A \times P_B$.	Clave Compartida: $K = d_B \times P_A$.	Ambos llegan al mismo punto K. Este punto es la Clave Maestra Secreta Simétrica .

El resultado K (la Clave Compartida) se utiliza entonces para cifrar los datos de la sesión usando un algoritmo rápido como ChaCha20 o AES-128.

Prueba Práctica de ECDH (Intercambio de Claves)

Contexto:

- IoT A (Cliente)
- Servidor B

FASE 1: Generación de Claves (En cada Máquina)

Paso	Terminal A (IoT)	Terminal B (Servidor)	Notas
1. Generar Privada	<code>openssl ecparam -name prime256v1 -genkey -noout -out A_priv.pem</code>	<code>openssl ecparam -name prime256v1 -genkey -noout -out B_priv.pem</code>	Crea el secreto privado (d_A y d_B).
2. Generar Pública	<code>openssl ec -in A_priv.pem -pubout -out A_pub.pem</code>	<code>openssl ec -in B_priv.pem -pubout -out B_pub.pem</code>	Extrae la clave pública (P_A y P_B).
3. Leer Pública	<code>cat A_pub.pem</code>	<code>cat B_pub.pem</code>	CRÍTICO: Muestra la clave pública para el intercambio manual.

Máquina A:

```
(kali㉿kali)-[~]
└─$ openssl ecparam -name prime256v1 -genkey -noout -out A_priv.pem

(kali㉿kali)-[~]
└─$ openssl ec -in A_priv.pem -pubout -out A_pub.pem
read EC key
writing EC key

(kali㉿kali)-[~]
└─$ cat A_pub.pem
-----BEGIN PUBLIC KEY-----
MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAE/HueMLuiMbhlwUlx7b+syACSh5Vr
aTzat5yR//VUvb6U+lcALG5ChBzII59m0jBAhgQuvFq/yeqdPxsUqy2w9w==
-----END PUBLIC KEY-----
```

Máquina B:

```

(kali㉿kali)-[~]
$ openssl ecparam -name prime256v1 -genkey -noout -out B_priv.pem

(kali㉿kali)-[~]
$ openssl ec -in B_priv.pem -pubout -out B_pub.pem
read EC key
writing EC key

(kali㉿kali)-[~]
$ nano A_pub.pem

```

```

(kali㉿kali)-[~]
$ cat B_pub.pem
-----BEGIN PUBLIC KEY-----
MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAEVfYrzwSYmirF3waHzE3CCZYXDoPd
OagbzSsDAMCNyik3/TrOA5p0BuCMaMv8HSVDghwt8pmbnI2pLogT8BchMg==
-----END PUBLIC KEY-----

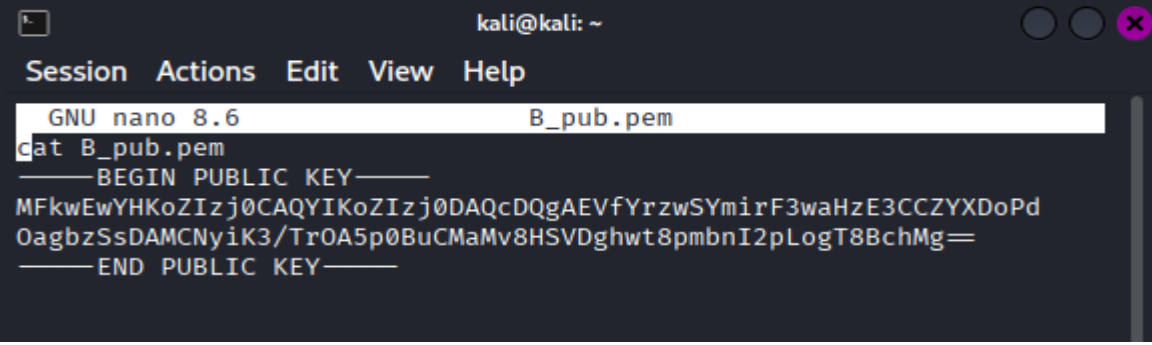
```

FASE 2: Intercambio Manual (La Clave del Éxito)

Copiar el texto completo de P_A (del Terminal A) y pegarlo en un nuevo archivo llamado **(A o B)_pub.pem** en el Terminal B (y viceversa para P_B).

Máquina A:

- nano B_pub.pem



```

kali@kali: ~
Session Actions Edit View Help
GNU nano 8.6 B_pub.pem
cat B_pub.pem
-----BEGIN PUBLIC KEY-----
MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAEVfYrzwSYmirF3waHzE3CCZYXDoPd
OagbzSsDAMCNyik3/TrOA5p0BuCMaMv8HSVDghwt8pmbnI2pLogT8BchMg==
-----END PUBLIC KEY-----

```

Máquina B:

- nano A_pub.pem

```

kali@kali: ~
Session Acciones Editar Vista Ayuda
GNU nano 8.6 A_pub.pem
cat A_pub.pem
-----BEGIN PUBLIC KEY-----
MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAE/HueMLuiMbhlwUlx7b+syACSh5Vr
aTzat5yR//VUvb6U+lcALG5ChBzII59m0jBAhgQuvFq/yeqdPxsUqy2w9w==
-----END PUBLIC KEY-----

```

FASE 3: Cálculo del Secreto Compartido K (La Prueba)

Verificamos que ambos equipos llegaron a la misma clave secreta K usando la fórmula: $K = d_{propia} \times P_{ajena}$

Terminal	Comando Ejecutado	Resultado (Secreto Compartido)
Terminal B (Servidor)	<code>openssl pkeyutl -derive -inkey B_priv.pem -peerkey A_pub.pem</code>	7•DP•W?•k•••••6h•••,•9•••
Terminal A (IoT)	<code>openssl pkeyutl -derive -inkey A_priv.pem -peerkey B_pub.pem</code>	7•DP•W?•k•••••6h•••,•9•••

Máquina B:

- `openssl pkeyutl -derive -inkey B_priv.pem -peerkey A_pub.pem`

```

(kali@kali)-[~]
$ openssl pkeyutl -derive -inkey B_priv.pem -peerkey A_pub.pem
7•DP•W?•k•••••6h•••,•9•••

```

Máquina A:

- `openssl pkeyutl -derive -inkey A_priv.pem -peerkey B_pub.pem`

```
(kali㉿kali)-[~]  
$ openssl pkeyutl -derive -inkey A_priv.pem -peerkey B_pub.pem  
7♦♦DP♦W♦?♦♦k♦♦♦♦  
♦6h♦}♦,9♦♦♦>
```

Los resultados binarios son **idénticos**. Esto demuestra exitosamente que:

- 1. El protocolo **ECDH** funciona: Dos partes (A y B) acordaron un secreto compartido a través de un canal inseguro.
- 2. El uso de claves de **Curva Elíptica (prime256v1)** es ideal para IoT debido a su eficiencia.
- 3. Se ha realizado la prueba en dos máquinas virtuales de Kali Linux.

Ejercicio 1.30: Implementación y seguridad en protocolos de intercambio de claves

Identifica los riesgos de seguridad del protocolo **Diffie-Hellman sin autenticación** y propón una solución para mitigarlos.

- 1. El Riesgo Fundamental: Man-in-the-Middle (MitM)

El principal problema de Diffie-Hellman (DH) en su versión pura (sin certificados) es la **falta de autenticación**.

Riesgo	Descripción	Consecuencia
Ataque Man-in-the-Middle	El atacante (M) se interpone en la comunicación. Cuando A envía su clave pública (P_A), M la intercepta y le envía a B su propia clave pública (P_M), y viceversa.	M establece dos secretos compartidos: K_{AM} (con A) y K_{BM} (con B). M puede descifrar, leer y volver a cifrar todos los mensajes sin que A o B lo detecten.
Falsificación de Identidad	A y B creen que están hablando entre sí, cuando en	La comunicación se asegura, pero no se asegura la identidad de las partes.

	realidad están hablando con el atacante.	
--	------------------------------------------	--

2. Solución Propuesta: Infraestructura de Clave Pública (PKI)


La solución para el MitM es **probar la identidad** del interlocutor antes de proceder con el intercambio de claves. Esto se hace integrando DH/ECDH con la **Infraestructura de Clave Pública (PKI)** a través de certificados digitales.

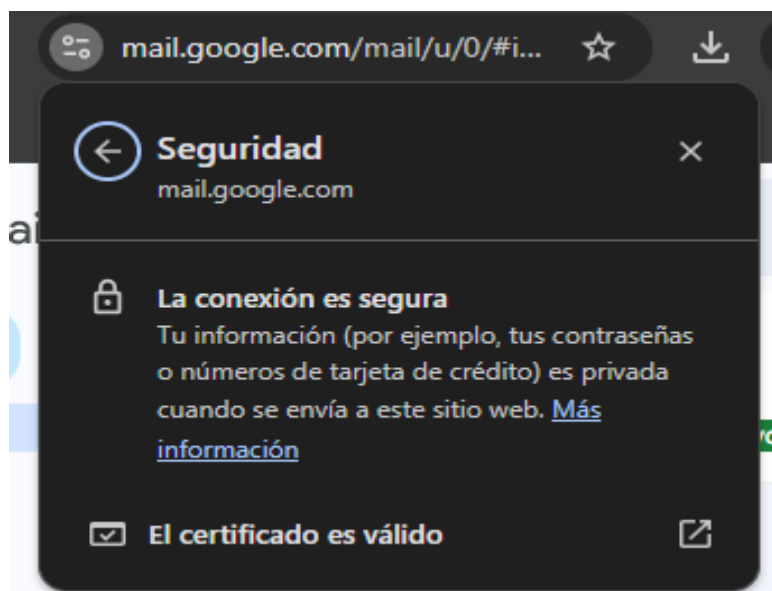
Solución	Mecanismo de Mitigación
Certificados Digitales	La clave pública de A y B se incluye en un Certificado Digital que está firmado por una Autoridad de Certificación (CA) de confianza (ej. Verisign).
Rol de la CA	Cuando B recibe la clave pública de A, B verifica la firma del certificado con la clave pública de la CA. Si la firma es válida, B sabe que la clave pública le pertenece legítimamente a A.
Mitigación del MitM	El atacante (M) no puede obtener un certificado válido de una CA para su clave falsa. Por lo tanto, el receptor detecta la falsificación y termina la conexión inmediatamente , frustrando el ataque.

Ejemplo Web (Página Sugerida)

Examinando un certificado TLS/SSL.

Elemento	Ejemplo	Lo que Demuestra

Página Web	Abre Google, tu banco, o cualquier web con HTTPS (ej. https://www.google.com).	Estas webs usan DH/ECDH para la clave de sesión, pero el Certificado RSA/ECDSA evita el MitM.
Visualización	Haz clic en el candado  en la barra de direcciones del navegador.	Verás los detalles del certificado: Quién lo emitió (CA) y Para quién es (el dominio). Esta prueba de identidad es el "salvavidas" contra el MitM.



Ejemplo en Kali Linux (Verificación de la Solución)

Objetivo: Obtener el certificado de un servidor (ej. Google) y forzar a Kali a verificar que la firma de ese certificado es correcta.

- `openssl s_client -showcerts -connect www.google.com:443 </dev/null`

```

(kali㉿kali)-[~]
$ openssl s_client -showcerts -connect www.google.com:443 </dev/nu
ll
Connecting to 142.250.200.68
CONNECTED(00000003)
depth=2 C=US, O=Google Trust Services LLC, CN=GTS Root R1
verify return:1
depth=1 C=US, O=Google Trust Services, CN=WR2
verify return:1
depth=0 CN=www.google.com
verify return:1
-----
Certificate chain
 0 s:CN=www.google.com
  i:C=US, O=Google Trust Services, CN=WR2
  a:PKEY: EC, (prime256v1); sigalg: sha256WithRSAEncryption
  v:NotBefore: Oct 13 08:39:42 2025 GMT; NotAfter: Jan  5 08:39:41
2026 GMT
-----BEGIN CERTIFICATE-----
MIIEVjCCAz6gAwIBAgIQKQo55F6Z128KPud0FAo6/zANBgkqhkiG9w0BAQsFADA7
MQswCQYDVQQGEwJVUzEeMBwGA1UEChMVR29vZ2xlIFRydXN0IFNlcnZpY2VzMQww
CgYDVQQDEwNXUjIwHhcNMjUxMDEzMDEzOTQyWhcNMjUxMDEzMDEzOTQyWjAZMRcw
FQYDVQQDEw53d3cuZ29vZ2xlLmNvbTBZMBMGByqGSM49AgEGCCqGSM49AwEHA0IA
BDUyyDGd5HTix8wWIk7jdr+wor6vHdXawVfnRekK3pYAan8mGAABdGohQN9FwY
Gr22zvHYtlrj/RReZffbc16jggJBMIICPTA0BgNVHQ8BAf8EBAMCB4AwEwYDVR0l
BAwwCgYIKwYBBQUHAWAwDAYDVDR0TAQH/BAIwADAdBgNVHQ4EFgQUkBOBgRZbSvHq
7Q+ySxMbDNZM2H4wHwYDVR0jBBgwFoAU3hse7XkV1D43JMMhu+w0OW1CsJAwWAYI
-----BEGIN CERTIFICATE-----

```

```

-----BEGIN CERTIFICATE-----
MIIEVjCCAz6gAwIBAgIQKQo55F6Z128KPud0FAo6/zANBgkqhkiG9w0BAQsFADA7
MQswCQYDVQQGEwJVUzEeMBwGA1UEChMVR29vZ2xlIFRydXN0IFNlcnZpY2VzMQww
CgYDVQQDEwNXUjIwHhcNMjUxMDEzMDEzOTQyWhcNMjUxMDEzMDEzOTQyWjAZMRcw
FQYDVQQDEw53d3cuZ29vZ2xlLmNvbTBZMBMGByqGSM49AgEGCCqGSM49AwEHA0IA
BDUyyDGd5HTix8wWIk7jdr+wor6vHdXawVfnRekK3pYAan8mGAABdGohQN9FwY
Gr22zvHYtlrj/RReZffbc16jggJBMIICPTA0BgNVHQ8BAf8EBAMCB4AwEwYDVR0l
BAwwCgYIKwYBBQUHAWAwDAYDVDR0TAQH/BAIwADAdBgNVHQ4EFgQUkBOBgRZbSvHq
7Q+ySxMbDNZM2H4wHwYDVR0jBBgwFoAU3hse7XkV1D43JMMhu+w0OW1CsJAwWAYI
KwYBBQUHAQEETDBKMCEGCGCsGAQUFBzABhhVodHRwOi8vby5wa2kuZ29vZy93cjIw
JQYIKwYBBQUHMAKGGWh0dHA6Ly9pLnBraS5nb29nL3dyMi5jcnQwGQYDVR0RBBIw
EII0d3d3Lmdvb2dsZS5jb20wEwYDVR0gBAwwCjAIBgZngQwBAgEwNgYDVR0fBC8w
LTARoCmgJ4YlaHR0cDovL2MucGtpLmdvb2cvd3IyL29RNm55cjhgMG0wLmNybDCC
AQQGCisGAQQB1nkCBAIEgfUEgfIA8AB2AJaXZL9VWJet900HaDcIQnfp8DrV9qTz
Nm5GpD8PyqnGAAABmdzwqDcAAAQDAEcwRQIgHzW4zdzhmZjV6vNnR/TK11tSW+87
m4qzJ65oV8KeVxACIQC2XKyaQ42Bem8ytoD1nRJTIIkhGQYP5GfI9Cl+uDhakAB2
ABaDLavwqSUPD/A6pUX/yL/II9CHS/YEKsf45x8zE/X6AAABmdzwqDsAAAQDAEcw
RQIgZWk9XDeKdcAqujtZ61CWQ9Wvptef2bNLxaHJXSihB70CIQCNrhJ6Xqwar9WI
Dohd+8LSRT9qzs7DX907bE+fhbCRmTANBgkqhkiG9w0BAQsFAAOCAQEAB0DpM1/i
GqzRkM4xU4ouPwfzNOBNKAIEKL+E6xtMtQ8k8G+7Qym2RFCKXB50LDLnxKwbuVWIn
5y9BtUlmWSIFkys6Iok17SvEhaWw4vVGXEMSLC9T+64uutk/LDQi+z8H7lG/qCQ6
a2yMXDDMrund4z15/NirOhpPh9eKCMst91VlgZ2mhKWAU161fpr3twZ3w5An3DC7
JQrZYbKKw5611lzvR/9kv1GmetwVnbzvNDWp0ngGt19rLZZ1Lg0hDOjY+UvUHg7o
VMn2zjlnICTDISQbpDGu0BtED3z79cy5aZTYVf04JQb8ReTBGgAwlf4IkAnIrBgn
/PWFe3fJMyD9ag=
-----END CERTIFICATE-----
 1 s:C=US, O=Google Trust Services, CN=WR2

```

```
+qduBmpvvYuR7hZL6Dupszfnw0Skfths18dG9ZKb59UhvmaSGZRVbNQpsg3BZlvi
d0lIK02d1xozcl0zgJXPYovJJiultzkMu34qQb9Sz/yilrbCgj8=
-----END CERTIFICATE-----

Server certificate
subject=CN=www.google.com
issuer=C=US, O=Google Trust Services, CN=WR2

No client certificate CA names sent
Peer signing digest: SHA256
Peer signature type: ecdsa_secp256r1_sha256
Negotiated TLS1.3 group: X25519MLKEM768

SSL handshake has read 5195 bytes and written 1763 bytes
Verification: OK

New, TLSv1.3, Cipher is TLS_AES_256_GCM_SHA384
Protocol: TLSv1.3
Server public key is 256 bit
This TLS version forbids renegotiation.
Compression: NONE
Expansion: NONE
No ALPN negotiated
Early data was not sent
Verify return code: 0 (ok)

DONE
```

La imagen confirma dos aspectos cruciales de la autenticación PKI:

A. Certificado del Servidor (La Identidad verificada)

Visualización clave: El bloque de texto codificado es el **Certificado Digital** que Google presentó para probar que es, de hecho, Google.

- CN=WR2 (Nombre Común del servidor)
- O=Google Trust Services (Organización que emitió el certificado)
- Este certificado está firmado por una Autoridad de Certificación (CA) de confianza.

B. Código de Verificación (La Mitigación del MitM)

Si el atacante intentara hacer un MitM y presentara un certificado falso, la máquina (cliente) lo rechazaría. El resultado final de la verificación es la prueba:

Segmento de la Salida	Significado	Riesgo Mitigado
Verify return code: 0 (ok)	La verificación criptográfica del certificado y de la cadena de confianza fue exitosa.	Si un atacante (MitM) hubiera interceptado la conexión y presentado un certificado no válido o falsificado, el código sería diferente de 0 (ej., 20 o 21), y la conexión se terminaría inmediatamente , frustrando el ataque.

Conclusión: El código 0 (ok) demuestra que el sistema PKI funciona, permitiendo que el intercambio de claves (DH/ECDH) continúe de forma segura.


Página 117. Ejercicio Herramientas GPG Online (Windows)


PGP Tool

A simple and secure online client-side PGP Key Generator, Encryption and Decryption tool. Generate your PGP Key pairs, encrypt or decrypt messages easily with a few clicks.


[Generate PGP Keys](#) [Sign](#) [Verify](#) [Encrypt](#) [Decrypt](#) [FAQ](#) [About](#)


Options


 Your Name Required


 you@domain.com Required


Email address: Why it is required?

 Optional comments

 Select algorithm... Required

 Select key size... Required

 Select expiry duration... Required


 Passphrase Required

Passphrase: What is this?

Generate keys


Public Key

Your public key will be generated here:

 Download public key (.ASC file) [Learn More](#)

Private Key

Your private key will be generated here:

 Download private key (.ASC file) [Learn More](#)

Paso 1. Generando Claves

PGP Tool

A simple and secure online client-side PGP Key Generator, Encryption and Decryption tool. Generate your PGP Key pairs, encrypt or decrypt messages easily with a few clicks.

Generate PGP Keys Sign Verify Encrypt Decrypt FAQ About

Options

Required
 Required
Email address: Why it is required?

 Required
 Required
 Required
 Required
Passphrase: What is this?

Finished

[Click here to regenerate another pair of key](#)

Public Key

```

-----BEGIN PGP PUBLIC KEY BLOCK-----
Version: Keybase OpenPGP v2.0.76
Comment: https://pgp.najm.uk

xsFN9GkTAwBEAC8cz2QWkEXuiefFgkXbaM+a2oHpcG8GWCDmn0XQp+plXtyCJ
3z09QT1seGbg/6ThaPdR/emISU7r0MEh8IMK7nULsAg1YYtq+jgaqBulyip7jY
0veYmDnEmKTW6I7Y1ucNEACXy9cPdeCRDXU82hkg2GLpQyIwK3hSAI+uJ+4H

```

[Download public key \(.ASC file\)](#)
[Learn More](#)

Private Key

```

-----BEGIN PGP PRIVATE KEY BLOCK-----
Version: Keybase OpenPGP v2.0.76
Comment: https://pgp.najm.uk

jcaGBGkTAwBEAC8cz2QWkEXuiefFgkXbaM+a2oHpcG8GWCDmn0XQp+plXtyCJ
3z09QT1seGbg/6ThaPdR/emISU7r0MEh8IMK7nULsAg1YYtq+jgaqBulyip7jY
0veYmDnEmKTW6I7Y1ucNEACXy9cPdeCRDXU82hkg2GLpQyIwK3hSAI+uJ+4H

```

[Download private key \(.ASC file\)](#)
[Learn More](#)

Password usada 'aagoes'.

Descargamos las claves:

equipo > Disco local (C:) > Usuarios > 2-DAW > Descargas

	Nombre	Fecha de modificación	Tipo	Tamaño
hoy (3)				
	0x96CB92D6-sec	11/11/2025 10:32	PGP Tool File	12 KB
	0x96CB92D6-pub	11/11/2025 10:32	PGP Tool File	6 KB

Paso 2. Firmando un mensaje

PGP Tool

A simple and secure online client-side PGP Key Generator, Encryption and Decryption tool. Generate your PGP Key pairs, encrypt or decrypt messages easily with a few clicks.

Generate PGP Keys **Sign** Verify Encrypt Decrypt FAQ About

Your Private Key

```

-----BEGIN PGP PRIVATE KEY BLOCK-----
Version: Keybase OpenPGP v2.0.76
Comment: https://pgp.najm.uk

xcG8GkTAwBEAC8czZQWEXuEFlgX0taM+a2oHpc
O8QWCDmX0hp+PctyCJ
3z09QT1snG0g/6TaPdR/ennidUtrdMEh88M907nULSAgI

```

Seleccionar archivo 0x96CB92D6-sec.asc

Your Message in Plain Text

Hola voy a firmar este mensaje.

Seleccionar archivo Ningún archivo seleccionado

Sign the message

Signed Message

Message successfully signed.

```

-----BEGIN PGP MESSAGE-----
Version: Keybase OpenPGP v2.0.76
Comment: https://pgp.najm.uk

yMGgAnkAVQCq3EDQMACgEJDSXNwEENQHLjHUAARMdK0tvsGEgdm9SgEgZ
mly
BWFYIGVzdGUpwWuc2FqZUxBXAAQsABgUCaRMDwAKCRAJDSXNwEENQX

```

Download signed message

Inicio Compartir Vista

Este equipo > Disco local (C:) > Usuarios > 2-DAW > Descargas

Buscar en Descargas

	Nombre	Fecha de modificación	Tipo	Tamaño
	hoy (4)			
	0x96CB92D6-sec.asc.signed	11/11/2025 10:36	Documento de te...	1 KB
	0x96CB92D6-sec	11/11/2025 10:32	PGP Tool File	12 KB
	0x96CB92D6-pub	11/11/2025 10:32	PGP Tool File	6 KB

Paso 3. Verificar un mensaje

PGP Tool

A simple and secure online client-side PGP Key Generator, Encryption and Decryption tool. Generate your PGP Key pairs, encrypt or decrypt messages easily with a few clicks.

Generate PGP Keys Sign Verify Encrypt Decrypt FAQ About

Signer's Public Key

```

-----BEGIN PGP PUBLIC KEY BLOCK-----
Version: Keybase OpenPGP v2.0.76
Comment: https://pgp.najm.uk

xsFNBGkTAwBEACBczZQWkEXueIFigXbaM+a2oHpc
G8GWC0mn00hp+plLtyCJ
3z09QT1snGbg6TiaPDRem5UTr0MEhB8MK7nULsAgT
Seleccionar archivo | 0x96C892D6-pub.asc

```

PGP-signed Message

```

-----BEGIN PGP MESSAGE-----
Version: Keybase OpenPGP v2.0.76
Comment: https://pgp.najm.uk

yMGgAnicAVQCq3EDQMACgEJDSXNwEENQHJhUAARM0k0hvoGEGom95iGegZ
mly
bWfYlGvz9GugbtWVuc2FqZC1BXAQAAQoABgUCaRMDkwAKCRAJDSXNwEENOX
Seleccionar archivo | 0x96C892D6 .asc signed.txt

```

Verify signature

Raw Message and Status

Message signature is verified with fingerprint:
65c653fec159c0ba5a80b59baab0dea96cb92d6

Hola voy a firmar este mensaje

Paso 4. Cifrando y firmando un mensaje

Primero creamos un nuevo par de claves:

PGP Tool

A simple and secure online client-side PGP Key Generator, Encryption and Decryption tool. Generate your PGP Key pairs, encrypt or decrypt messages easily with a few clicks.

Generate PGP Keys Sign Verify Encrypt Decrypt FAQ About

Options

bb Required

bb@go.es Required

Email address: Why it is required?

hola soy bb

RSA (Recommended) Required

4096 bits (more secure) Recommended Required

1 year Required

***** Required

Passphrase: What is this?

Finished

Click here to regenerate another pair of key

Public Key

```

-----BEGIN PGP PUBLIC KEY BLOCK-----
Version: Keybase OpenPGP v2.0.76
Comment: https://pgp.najm.uk

xsFNBGkTDZwBEACbRmX9Gcu8F9APFDGfPbtaMawYXbZLjvzmT4Pc2QGEVimYQV
pu
nDzCR0iVV4gZylejay4+0alVnd1X0+Ppavm5hTJR213QjOy4T9L+n4HDg6vHN2
Download public key (.ASC file) Learn More

```

Private Key

```

-----BEGIN PGP PRIVATE KEY BLOCK-----
Version: Keybase OpenPGP v2.0.76
Comment: https://pgp.najm.uk

xcaGBGkTDZwBEACbRmX9Gcu8F9APFDGfPbtaMawYXbZLjvzmT4Pc2QGEVimYQV
pu
nDzCR0iVV4gZylejay4+0alVnd1X0+Ppavm5hTJR213QjOy4T9L+n4HDg6vHN2
Download private key (.ASC file) Learn More

```

Utilizando Password 'bbgoes'.

Descargamos las claves:

VISTA

> Disco local (C:) > Usuarios > 2-DAW > Descargas

Buscar en Descargas

Nombre	Fecha de modificación	Tipo	Tamaño
▼ hoy (6)			
🔒 0xD31D15B7-sec	11/11/2025 11:21	PGP Tool File	12 KB
🔒 0xD31D15B7-pub	11/11/2025 11:21	PGP Tool File	6 KB

Utilizando la clave pública de BB y la clave privada de AA (con su password), AA escribe un mensaje cifrado a BB:

PGP Tool

A simple and secure online client-side PGP Key Generator, Encryption and Decryption tool. Generate your PGP Key pairs, encrypt or decrypt messages easily with a few clicks.

[Generate PGP Keys](#) [Sign](#) [Verify](#) [Encrypt](#) [Decrypt](#) [FAQ](#) [About](#)

Receiver's Public Key

```
-----BEGIN PGP PUBLIC KEY BLOCK-----
Version: Keybase OpenPGP v2.0.76
Comment: https://pgp.najm.uk

xsFNBGkTDZwBEADbRmX9Geu8F9APF3GIPtMawY
XbZUvzmT4Pc2QGEVmyQVpu
nDzCROIV4giZylejy4+0ahvNd1X8+Ppwm5hTJR213Qj
```

[Seleccionar archivo](#) 0xD31D1587-pub.asc

Signer's Private Key (For signing purpose)

```
-----BEGIN PGP PRIVATE KEY BLOCK-----
Version: Keybase OpenPGP v2.0.76
Comment: https://pgp.najm.uk

xcaGBGkTAwBEAC8czQWkEXuEfIgXbM+a2oHpc
G8GWCDmX0Xp+plUtyCJ
3z09QT1seGbg6TiaPdR/mSUT0MEh8IMK7nULsAgT
```

[Seleccionar archivo](#) 0x96CB92D6-sec.asc

[\[icon\]](#) *****

Your Message in Plain Text

Hola BB, Soy AA

[Seleccionar archivo](#) Ningún archivo seleccionado

[Encrypt the message](#)

Encrypted PGP Message

Message successfully encrypted and signed

```
-----BEGIN PGP MESSAGE-----
Version: Keybase OpenPGP v2.0.76
Comment: https://pgp.najm.uk

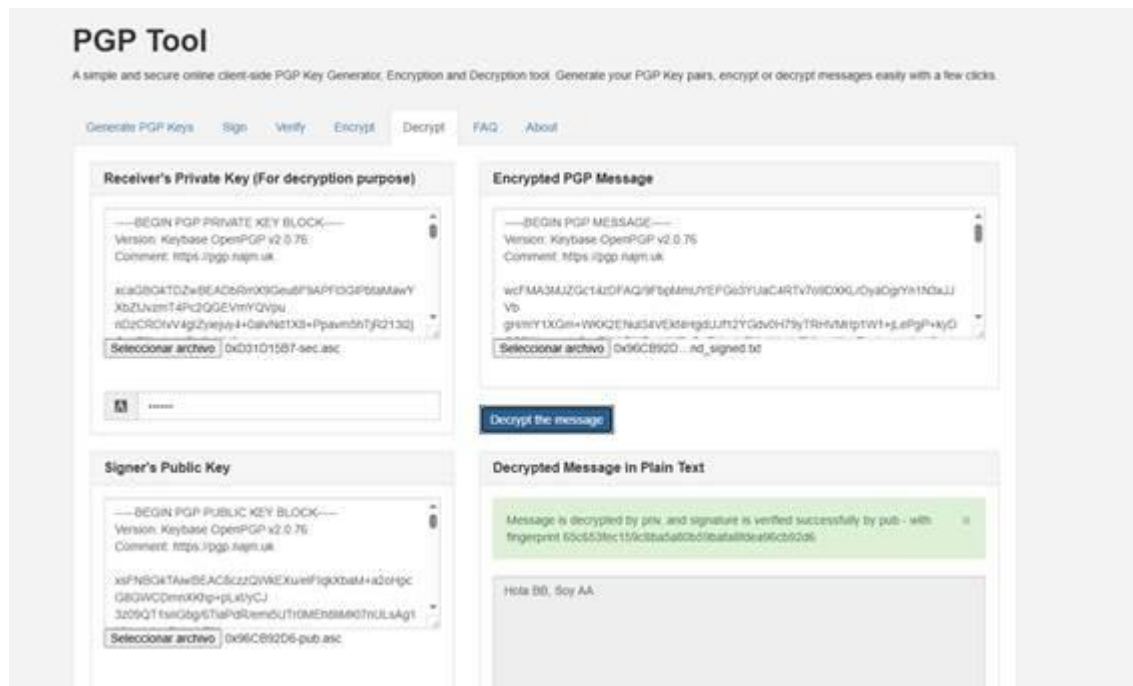
wcfMA3MJZGc14zDFAQ/9FbpMmLJYEFGe3YUaC4RTv7o9DXXLiDyaDgrYn1NdxUJ
Vb
ghHmY1XGm+WKQGENu3hVEXt4HgdLUH2YQdvKH79yTTRfVUtp1Wt+JLePgP+kjO
```

[Download encrypted message](#)

local (C:) > Usuarios > 2-DAW > Descargas

Buscar en C

Nombre	Fecha de modificación	Tipo	Tamaño
▼ hoy (7)			
0x96CB92D6-sec.asc.encrypted_and_sign...	11/11/2025 11:26	Documento de te...	2 KB
0xD31D1587-sec	11/11/2025 11:21	PGP Tool File	12 KB
0xD31D1587-pub	11/11/2025 11:21	PGP Tool File	6 KB
0x96CB92D6-sec.asc.signed	11/11/2025 10:36	Documento de te...	1 KB
0x96CB92D6-sec	11/11/2025 10:32	PGP Tool File	12 KB
0x96CB92D6-pub	11/11/2025 10:32	PGP Tool File	6 KB



Página 131. Herramientas de cifrado

Ejemplo Práctico: Uso de Luks en Linux

LUKS (Linux Unified Key Setup) es el estándar para cifrado de discos en Linux. Permite cifrar volúmenes completos, particiones o archivos contenedores.

1. Crear un archivo contenedor

Crear un archivo que funcionará como nuestro "disco virtual" cifrado.

```
dd if=/dev/zero of=/home/kali/Documents/box/volumen_ejercicio bs=1M count=100
```

Explicación de parámetros:

- `if=/dev/zero`: Fuente de datos (ceros)
- `of=~/.mi_volumen_cifrado.img`: Archivo de salida
- `bs=1M`: Tamaño de bloque (1 Megabyte)
- `count=1024`: Número de bloques ($1024 * 1\text{MB} = 1\text{GB}$)

```
~/Documents/box > dd if=/dev/zero of=/home/kali/Documents/box/volumen_cifrado bs=1M count=100
100+0 records in
100+0 records out
104857600 bytes (105 MB, 100 MiB) copied, 0.284529 s, 369 MB/s
```

2. Configurar el cifrado con LUKS

Aplicar el cifrado LUKS al archivo contenedor.

```
sudo cryptsetup luksFormat /home/kali/Documents/box/volumen_cifrado
```

Explicación:

- Te pedirá una contraseña (mínimo 8 caracteres recomendado)
- Esta contraseña será necesaria para abrir el volumen
- El comando formatea el archivo con encabezado LUKS

```
~/Documents/box > sudo cryptsetup luksFormat /home/kali/Documents/box/volumen_cifrado
WARNING!
=====
This will overwrite data on /home/kali/Documents/box/volumen_cifrado irrevocably.
Are you sure? (Type 'yes' in capital letters): YES
Enter passphrase for /home/kali/Documents/box/volumen_cifrado:
Verify passphrase:
```

3. Abrir el volumen cifrado

Descifrar el volumen y hacerlo accesible al sistema.

```
sudo cryptsetup luksOpen /home/kali/Documents/box/volumen_cifrado volumen_cifrado2
```

- Te pedirá la contraseña que configuraste en el paso 2
- Crea un dispositivo mapeado en /dev/mapper/mi_volumen
- Este dispositivo ahora puede formatearse y montarse

```
~/Documents/box > sudo cryptsetup luksOpen /home/kali/Documents/box/volumen_cifrado volumen_cifrado2
Enter passphrase for /home/kali/Documents/box/volumen_cifrado:
```

4. Formatear el volumen cifrado

Crear un sistema de archivos en el volumen descifrado.

```
sudo mkfs.ext4 /dev/mapper/cifrado2
```

Explicación:

- Crea un sistema de archivos ext4 en el volumen descifrado
- Solo se hace una vez (la primera vez que usas el volumen)

- Los datos se cifran/descifran automáticamente al escribir/leer

```
~/Documents/box > sudo mkfs.ext4 /dev/mapper/volumen_cifrado2
mke2fs 1.47.2 (1-Jan-2025)
Creating filesystem with 21504 4k blocks and 21504 inodes

Allocating group tables: done
Writing inode tables: done
Creating journal (1024 blocks): done
Writing superblocks and filesystem accounting information: done
```

5. Montar el volumen cifrado

Hacer accesible el sistema de archivos en el directorio deseado.

```
~/Documents/box > sudo mount /dev/mapper/volumen_cifrado2 /mnt
~/Documents/box > ls /mnt
lost+found
```

6. Desmontar y cerrar el volumen

Proteger los datos cifrándolos nuevamente.

```
~/Documents/box > sudo umount /mnt
~/Documents/box > sudo cryptsetup luksClose volumen_cifrado2
```

7. Volver a montar el volumen

Acceder nuevamente a los datos cifrados.

```
~/Documents/box > sudo cryptsetup luksOpen /home/kali/Documents/box/volumen_cifrado volumen_cifrado2
Enter passphrase for /home/kali/Documents/box/volumen_cifrado:
~/Documents/box > sudo mount /dev/mapper/volumen_cifrado2 /mnt
~/Documents/box > ll /mnt
total 24
drwxr-xr-x 3 root root 4096 Nov 11 08:45 ./
drwxr-xr-x 18 root root 4096 Oct 31 12:11 ../
drwx----- 2 root root 16384 Nov 11 08:45 lost+found/
```

Ejercicio 1.31: Cifrado y firma de archivos con GPG

GPG (GNU Privacy Guard) es una herramienta que permite cifrar, firmar y verificar archivos o mensajes usando criptografía de clave pública.

Esto garantiza la Confidencialidad (Sólo el destinatario puede leer el contenido),

Autenticidad (Permite verificar quién envió el archivo) e Integridad (Detecta si el archivo fue modificado).

1. Generar Par de Claves

`gpg --generate-key`

Explicación:

- Este comando inicia un asistente interactivo.
- Te pedirá:
 - Nombre y correo electrónico.
 - Tipo y tamaño de clave (por defecto RSA de 3072 o 4096 bits).
 - Fecha de expiración.
 - Una contraseña segura.

```
~/Documents/box > gpg --generate-key
gpg (GnuPG) 2.4.8; Copyright (C) 2025 g10 Code GmbH
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.

Note: Use "gpg --full-generate-key" for a full featured key generation dialog.

GnuPG needs to construct a user ID to identify your key.

Real name: LoreGraCris
Email address: loregracris@gmail.com
You selected this USER-ID:
  "LoreGraCris <loregracris@gmail.com>"

Change (N)ame, (E)mail, or (O)kay/(Q)uit? o
We need to generate a lot of random bytes. It is a good idea to perform
some other action (type on the keyboard, move the mouse, utilize the
disks) during the prime generation; this gives the random number
generator a better chance to gain enough entropy.
We need to generate a lot of random bytes. It is a good idea to perform
some other action (type on the keyboard, move the mouse, utilize the
disks) during the prime generation; this gives the random number
generator a better chance to gain enough entropy.
gpg: revocation certificate stored as '/home/kali/.gnupg/openpgp-revocs.d/9BEC818A9D0DF0AEB0349
223FEEEC4FA9FF1323C.rev'
public and secret key created and signed.

pub   ed25519 2025-11-11 [SC] [expires: 2028-11-10]
       9BEC818A9D0DF0AEB0349223FEEEC4FA9FF1323C
uid    LoreGraCris <loregracris@gmail.com>
sub    cv25519 2025-11-11 [E] [expires: 2028-11-10]
```

Resultado:

- **Clave privada:** se guarda localmente y **no debe compartirse**.
- **Clave pública:** se puede compartir con otros para que te envíen archivos cifrados o verifiquen tus firmas.

Verificar claves generadas

`gpg --list-keys`

```
uid          [ultimate] LoreGraCris <loregracris@gmail.com>  
sub   cv25519 2025-11-11 [E] [expires: 2028-11-10]
```

2. Cifrar y firmar un archivo

Supongamos que tienes un archivo llamado `archivo_secreto.txt` y quieres cifrar y firmar para enviarlo al destinatario.

`gpg --encrypt --sign --armor -r destinatario@email.com archivo_secreto.txt`

Parámetros explicados:

- `--encrypt`: Cifrar el archivo
- `--sign`: Añade firma digital
- `--armor`: Crea salida en texto ASCII (opcional, útil para email)
- `-r`: Especifica el destinatario (su clave pública)
- `--output`: Especifica archivo de salida

```
~/Documents/box > echo "nota secreta" > archivo_secreto.txt  
~/Documents/box > cat archivo_secreto.txt  
nota secreta
```

```
~/Documents/box > gpg --encrypt --sign --armor -r loregracris@gmail.com archivo_secreto.txt  
gpg: checking the trustdb  
gpg: marginals needed: 3 completes needed: 1 trust model: pgp  
gpg: depth: 0 valid: 1 signed: 0 trust: 0-, 0q, 0n, 0m, 0f, 1u  
gpg: next trustdb check due at 2028-11-10
```

Se genera un archivo cifrado, normalmente con extensión `.asc` o `.gpg` con el contenido del mensaje.

```
~/Documents/box > cat archivo_secreto.txt.asc
-----BEGIN PGP MESSAGE-----

hF4DClRYfk7+CK4SAQdAh6gZtT0yaJLsW4hqUxSRwtNL2gifNkAD0DbDv6WQ4Dkw
j0E1iT86r/6dKVSR4BAqaP+ckupjSuxYilYHTVuR0n/mRsV7zl+2RUwvbVCyo1Hg
0sAUAYIYr01EKl5AfIPVRxFmWMmEwlmCytLI9odc04E+ehhGv+e1DylsP4wLq44q
i9M0TnCCVa0FZQm/kPpa/pWi79z+Vjx8tCG4F4iuHtrTaf8g9DGaiug8mTJaz5y
AkbsnZt47ci+0m9cmuSJHEPUK3SEGzYLh09ex/Pg2LreZ4qrSa/Io24gutfWLHYP
Fkl+zKptQs4W94LWFRUiUDhccPSTXodL0qeIH80BNnWdtN30Eb0+PmN39/3CcNh
dqaIfqlAN0uv/40BBTYTTdcDjXsSFn0=
=TwAi
-----END PGP MESSAGE-----
```

3. Exportar tu clave pública para compartirla

Para que otros puedan enviarte archivos cifrados o verificar tus firmas, debes compartir tu clave pública.

```
gpg --export --armor loregracris@gmail.com > mi_clave_publica.asc
```

- --export: Exporta tu clave pública.
- --armor: La convierte en texto legible (ASCII).
- >: Guarda la salida en un archivo (mi_clave_publica.asc).

```
~/Documents/box > gpg --export --armor loregracris@gmail.com > mi_clave_publica.asc
~/Documents/box > cat mi_clave_publica.asc
-----BEGIN PGP PUBLIC KEY BLOCK-----

mDMEaRLroRYJKwYBBAHaRw8BAQdA/gnMUHCK5nSs0aNLpTpPD0ewUZA8gOLYgTSc
dIJMXRa0I0xvcmVHcmFDcmIzIDxs3JlZ3JhY3Jpc0BnbWFpbC5jb20+iJYEEYK
AD4WISb7IGKnQ3wrrA0kiP+7sT6n/EyPAUCaRLroQIbAwUJBa0agAULCQgHAgYV
CgkICwIEFgIDAQIeAQIXgAAKCRD+7sT6n/EyPK/tAQDh09ogmXrs0W03EB/+r0Gu
W8JIFv/VmKtMBT4EFLzITgD/e0fgVI43Ymv1efKhhWRLR6Drm2FbDuSWzkDn3gDi
0Aa40ARpEuuhEgorBgEEAZdVAQUBAQdALHuxwdZGBt0vXucxf+NPS6LAG/4a9Lu2
435l9cuw1UEDAQgHiH4EGBYKACYWISb7IGKnQ3wrrA0kiP+7sT6n/EyPAUCaRLr
oQIbDAUJBa0agAAKCRD+7sT6n/EyPHexAP9FpNxFmYJwA4rfT+qo3Rlnrd+XAaj8
9lqia2J0nmE/ygD9FF9MwULGOVjaTHBRdbdYEvth9dkm0GV6CX+xRgn0wY=
=C6B0
-----END PGP PUBLIC KEY BLOCK-----
```

4. El proceso de envío del archivo cifrado (En el remitente)

Adjuntar el archivo cifrado .gpg o .asc , y la clave pública con el que fué cifrado generado en el apartado anterior

Envía por correo o medio seguro:

- El archivo cifrado (archivo_secreto.txt.asc o .gpg).

- Tu clave pública (mi_clave_publica.asc), si el destinatario aún no la tiene.

Métodos de envío:

- Email: Adjuntar ambos archivos
- Servicios cloud: Subir a Google Drive, Dropbox, etc.
- Mensajería: Enviar por WhatsApp, Telegram, Signal

Recomendaciones de seguridad:

- Enviar archivos por canales diferentes
- Verificar la autenticidad del destinatario
- Usar contraseñas seguras para el email

5. Proceso de Recepción (Destinatario). Importar clave pública del remitente

En el equipo del destinatario, se debe importar la clave pública de quien envía. Esto añade la clave del remitente al llavero de GPG, lo que permitirá verificar su firma digital al descifrar.

gpg --import clave_publica_remitente.asc

```
~/Documents/box > gpg --import clave_publica_remitente.asc
gpg: key FEEEC4FA9FF1323C: "LoreGraCris <loregracris@gmail.com>" not changed
gpg: Total number processed: 1
gpg: unchanged: 1
```

6. Descifrar y verificar el archivo

GPG verificará automáticamente la firma del remitente, si está incluida.

gpg --decrypt archivo_secreto.txt.asc > archivo_descifrado.txt

- --decrypt: Descifra el contenido.
- >: Redirige la salida a un archivo de texto plano.

```
~/Documents/box > gpg --decrypt archivo_secreto.txt.asc > archivo_descifrado.txt
gpg: encrypted with cv25519 key, ID 0A54587E4EFE08AE, created 2025-11-11
    "LoreGraCris <loregracris@gmail.com>"
gpg: Signature made Tue 11 Nov 2025 09:02:50 AM CET
gpg: using EDDSA key 9BEC818A9D0DF0AEB0349223FEEEC4FA9FF1323C
gpg: Good signature from "LoreGraCris <loregracris@gmail.com>" [ultimate]
```

Ya podemos leer el archivo descifrado

```
~/Documents/box > cat archivo_descifrado.txt
nota secreta
```

Otros comandos útiles

Listar claves

`gpg --list-keys` *# Claves públicas*

`gpg --list-secret-keys` *# Claves privadas*

Eliminar claves

`gpg --delete-key email@destinatario.com` *# Clave pública*

`gpg --delete-secret-key email@propio.com` *# Clave privada*

Ejercicio 1.32: Configuración de GPG en un cliente de correo electrónico

1. Instalar un cliente de correo

`sudo apt update`

`sudo apt install thunderbird`

```
~ > sudo apt install thunderbird
The following packages were automatically installed and are no longer required:
base58 python3-flask-sqlalchemy
binutils-mingw-w64-i686 python3-flaskext.wtf
binutils-mingw-w64-x86-64 python3-flatbuffers
gcc-mingw-w64-base python3-gevent
gcc-mingw-w64-i686-win32 python3-gevent-websocket
gcc-mingw-w64-i686-win32-runtime python3-html2text
gcc-mingw-w64-x86-64-win32 python3-hupper
gcc-mingw-w64-x86-64-win32-runtime python3-kombu
gir1.2-vte-2.91 python3-log-symbols
libadwaita1 python3-marshmallow
```

2. Configuramos la cuenta de correo para que pueda usar los protocolos IMAP y POP desde otro dispositivo

Esto dependerá del proveedor del servicio de correo electrónico. En Outlook es:

Email aliases

Set default From address

loregracris@outlook.com

Email aliases

[Manage or choose a primary alias](#)

POP and IMAP

☒ Let devices and apps use POP

Devices and apps that use POP can be set to delete messages from Outlook after download.

- ☐ Don't allow devices and apps to delete messages from Outlook. It will move the messages to a special POP folder instead.
- ☒ Let apps and devices delete messages from Outlook

☒ Let devices and apps use IMAP

[View POP, IMAP, and SMTP settings](#)

Save

Discard

3. Configurar Thunderbird con nuestra cuenta de correo personal

Your full name
loregracris ⓘ

Email address
loregracris@outlook.com ⓘ

Password
●●●●●●●●●● ⓘ

☒ Remember password

Your login
YOURDOMAIN\yourusername ⓘ

Manual configuration

INCOMING SERVER

Protocol: IMAP ▾

Hostname: outlook.office365.com

Port: 993 ▴ ▾

Connection security: SSL/TLS ▾

Authentication method: OAuth2 ▾

Username: loregracris@outlook.com

OUTGOING SERVER

Hostname: smtp-mail.outlook.com

Port: 587 ▴ ▾

Connection security: STARTTLS ▾

Authentication method: OAuth2 ▾

Username: loregracris@outlook.com

Aparecerá una ventana para confirmar la vinculación que dependiendo del servicio de correo electrónico dará distintos métodos para identificarse. En Outlook se ve tal que:

Enter credentials for loregracris@outlook.com on outlook.office365.com

update?mkt=EN-US&uiflavor=host&id=292841&lqsp=ntprob%3d-1&ru



loregracris@outlook.com



Let this app access your
info? (1 of 1 apps)

Mozilla Thunderbird 

Thunderbird needs your permission to:



Read and write access to your mail

Thunderbird will be able to read, update, create, and delete email in your mailbox. Does not include permission to send mail.



Read and write access to your mail

Thunderbird will be able to read, update, create, and delete email in your mailbox. Does not include permission to send mail.



Access to sending emails from your mailbox

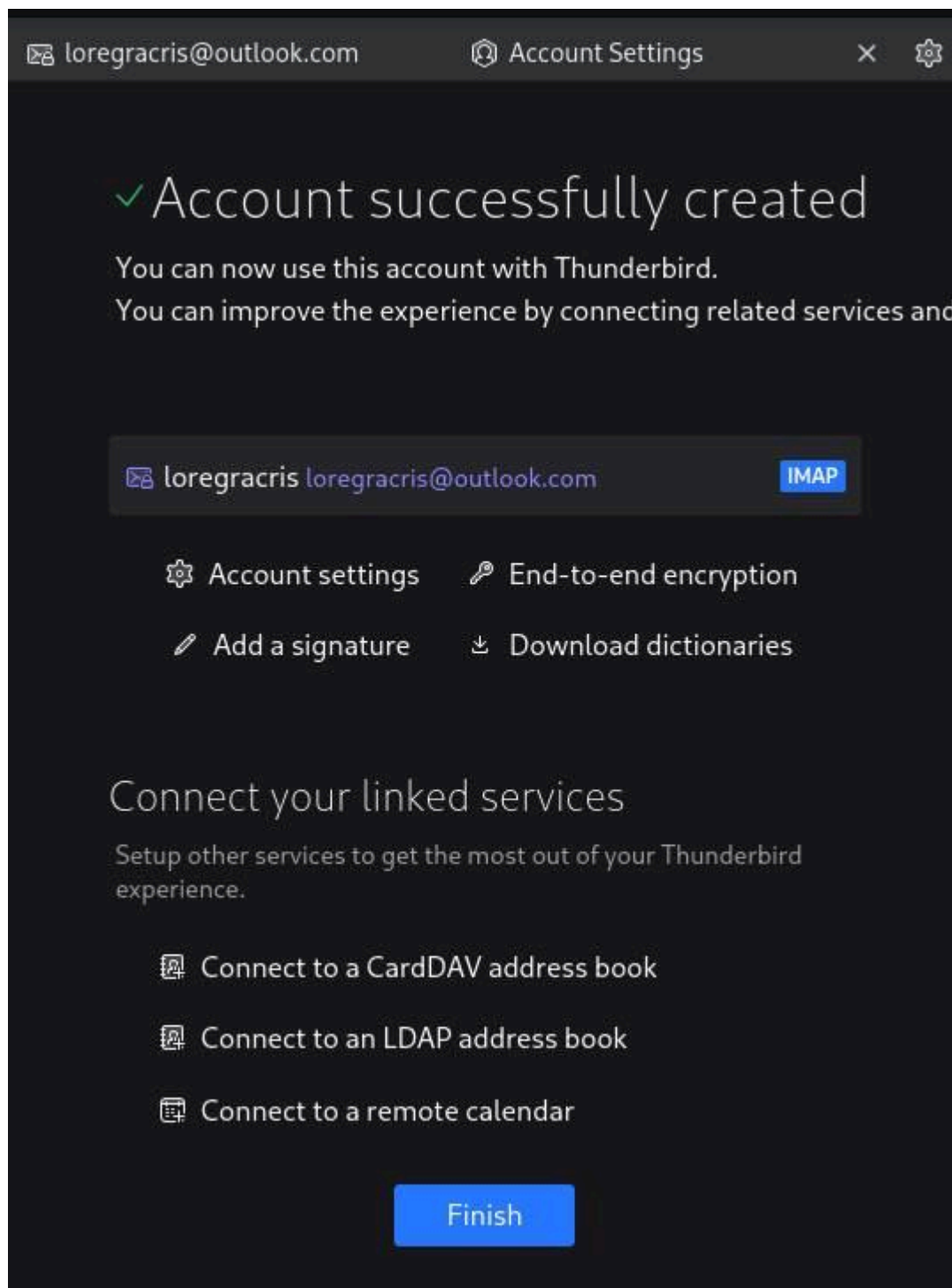
Thunderbird will be able to send email on your behalf from your mailbox.



**Maintain access to data you have given
Thunderbird access to**

Allows Thunderbird to see and update the data you gave it access to, even when you are not currently using the app. This does not give Thunderbird any additional permissions.

Aceptamos y aparecerá la pantalla de cuenta conectada correctamente



Ahora para poder mandar correos cifrados y firmados deberemos tener en nuestro equipo el conjunto de claves privada y pública asociado a nuestra cuenta de correo electrónico, y para poder descifrar correos recibidos necesitaremos la clave pública de la persona que nos mande el mensaje.

Podemos generar las claves desde la terminal e importarlas en Thunderbird, mediante:

gpg --generate-key

```
~ > gpg --generate-key 5s
gpg (GnuPG) 2.4.8; Copyright (C) 2025 g10 Code GmbH
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.

Note: Use "gpg --full-generate-key" for a full featured key generation dialog.

GnuPG needs to construct a user ID to identify your key.

Real name: Loregracris
Email address: loregracris@outlook.com
You selected this USER-ID:
  "Loregracris <loregracris@outlook.com>"

Change (N)ame, (E)mail, or (O)kay/(Q)uit? o
We need to generate a lot of random bytes. It is a good idea to perform
some other action (type on the keyboard, move the mouse, utilize the
disks) during the prime generation; this gives the random number
generator a better chance to gain enough entropy.
We need to generate a lot of random bytes. It is a good idea to perform
some other action (type on the keyboard, move the mouse, utilize the
disks) during the prime generation; this gives the random number
generator a better chance to gain enough entropy.
gpg: revocation certificate stored as '/home/kali/.gnupg/openpgp-revocs.d/5FEE55DDCD50D605B1024
4D9B3D94E5B048C9760.rev'
public and secret key created and signed.

pub  ed25519 2025-11-11 [SC] [expires: 2028-11-10]
     5FEE55DDCD50D605B10244D9B3D94E5B048C9760
uid  Loregracris <loregracris@outlook.com>
sub  cv25519 2025-11-11 [E] [expires: 2028-11-10]
```

Verificar claves generadas

gpg --list-keys

```
~ > gpg --list-keys loregracris@outlook.com
pub  ed25519 2025-11-11 [SC] [expires: 2028-11-10]
     5FEE55DDCD50D605B10244D9B3D94E5B048C9760
uid  [ultimate] Loregracris <loregracris@outlook.com>
sub  cv25519 2025-11-11 [E] [expires: 2028-11-10]
```

Exportar tu clave pública

`gpg --export --armor loregracris@outlook.com > clave_publica_loregracris.asc`

```
~ > cat clave_publica_loregracris.asc
-----BEGIN PGP PUBLIC KEY BLOCK-----

mDMEaRMCyHYJKwYBBAHaRw8BAQdACftFW2Q/32tvIp8G+yzozL5LQRH+qwibkAYj
AjFZyM20JUxvcmVncmFjcmIzIDxsY3JlZ3JhY3Jpc0BvdXRsb29rLmNvbT6IlgQT
FgoAPhYhBF/uVd3NUNYFsQJE2bPZTlsEjJdgBQJpEwLKAhsDBQkFo5qABQsJCAcC
BhUKCQgLAGQWAgMBAh4BAheAAAJELPZTlsEjJdgxEAA/0+QYSYgznVyeuehuwj3
p70cDnMHYAVb4rgLwopQQR7qAQDwtLQWhSD9iMdo26bMikrN+VX8X5bnkh+wncEy
DhFuAbg4BGkTAsSCisGAQQBl1UBBQEBB0AAQib0jZP7jw+cFnL37PXt486pPMUh
n07HWLUhxxNTJQMBCAeIfgQYFgoAJhYhBF/uVd3NUNYFsQJE2bPZTlsEjJdgBQJp
EwLKAhsMBQkFo5qAAAoJELPZTlsEjJdgPbgBAPAIBuTUtS8xRLg3AhcXUjJbBgAm
3uV8nUbJPGuqFAGbAQDaBBap/A9yqMmrmlPTOWiq2qJoEy4IKyo7tKJ1kbCQ==
=7tBh
-----END PGP PUBLIC KEY BLOCK-----
```

O crearlas desde el propio Thunderbird para mayor comodidad

The screenshot shows the 'Generate OpenPGP Key' window in Thunderbird. It has a dark theme. At the top, the title is 'Generate OpenPGP Key'. Below it, the 'Identity' field is set to 'loregracris <loregracris@outlook.com> - loregracris@outlook.com'. The 'Key expiry' section has a description: 'Define the expiration time of your newly generated key. You can later control the date to extend it if necessary.' There are two radio buttons: 'Key expires in' (selected) and 'Key does not expire'. The 'Key expires in' option has a dropdown set to '3' and a unit dropdown set to 'years'. The 'Advanced settings' section has a description: 'Control the advanced settings of your OpenPGP Key.' It includes 'Key type' set to 'RSA' and 'Key size' set to '3072'. At the bottom right, there are 'Cancel' and 'Generate key' buttons.

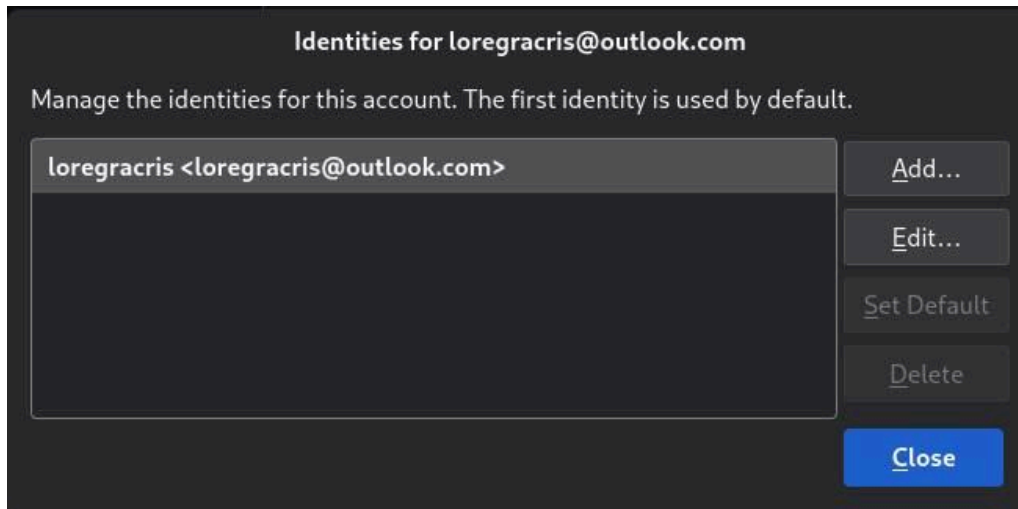
4. Configurar OpenPGP en Thunderbird

Debemos asegurarnos que los servicios de tu servidor de correos estén configurados y accesibles desde Thunderbird

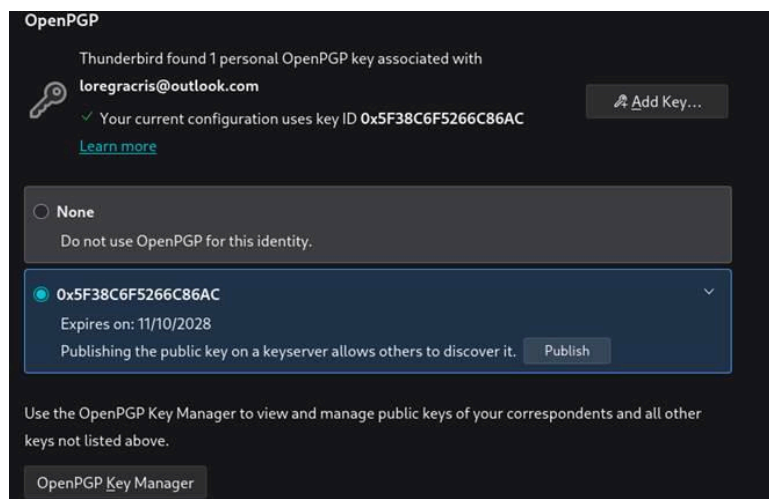
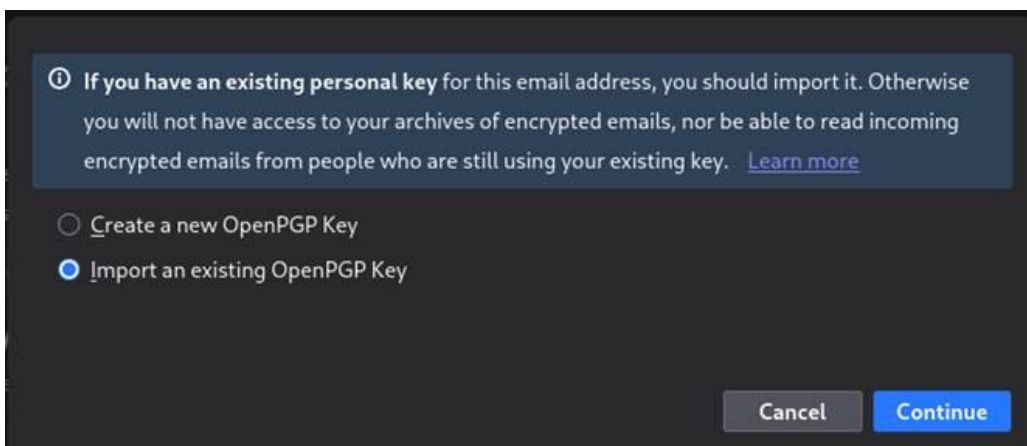
The screenshot shows the 'Outgoing Server (SMTP)' configuration in Thunderbird. It has a dark theme. The 'Outgoing Server (SMTP):' label is followed by a dropdown menu showing 'Outlook.com (Microsoft) - smtp-mail.outlook.c...'. To the right of the dropdown is a button labeled 'Edit SMTP server...'. Below these, there is a button labeled 'Manage Identities...'.

Y nuestra clave privada para cifrar los mensajes esté correctamente importada y accesible.

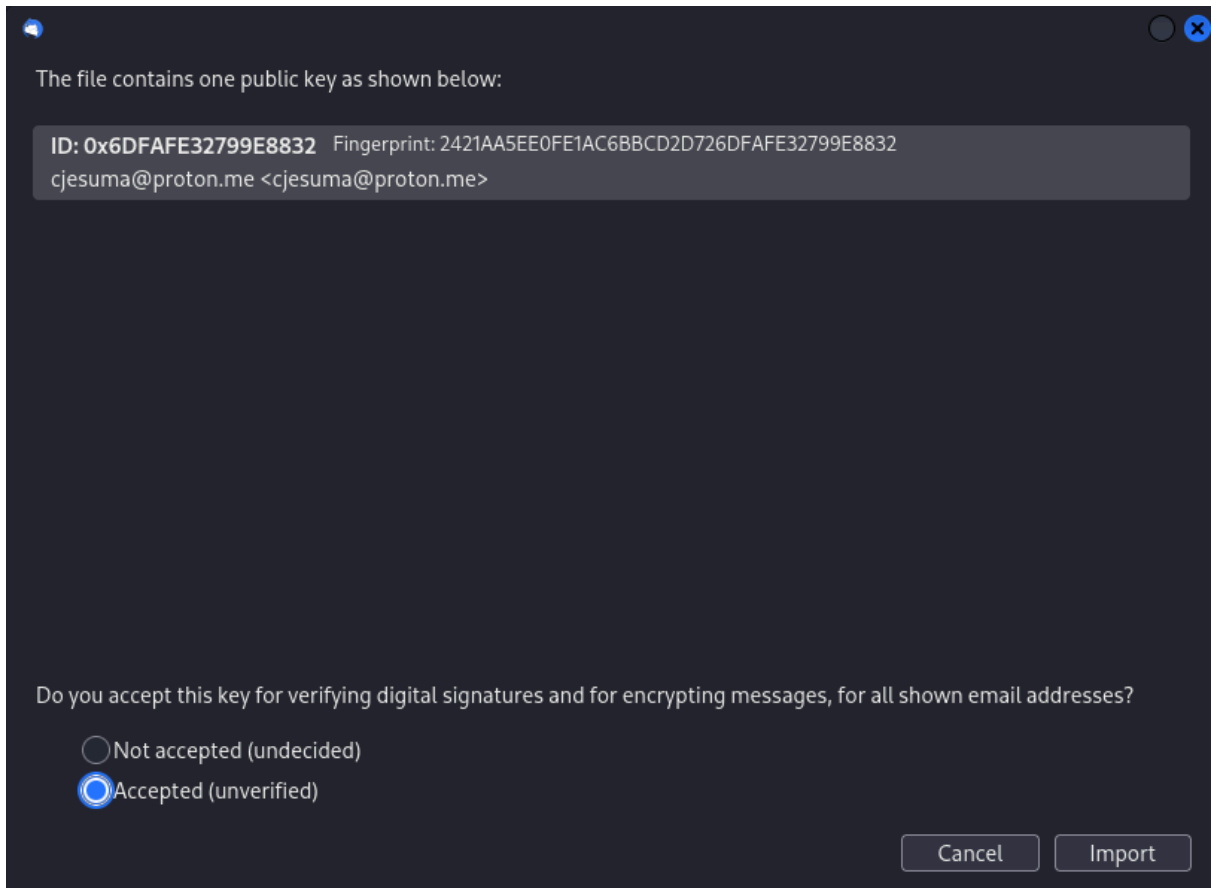
En la configuración de la aplicaciones buscamos la pestaña de Cifrado Punto a Punto o End-To-End-Encryption, en importamos con el botón Edit o Add.



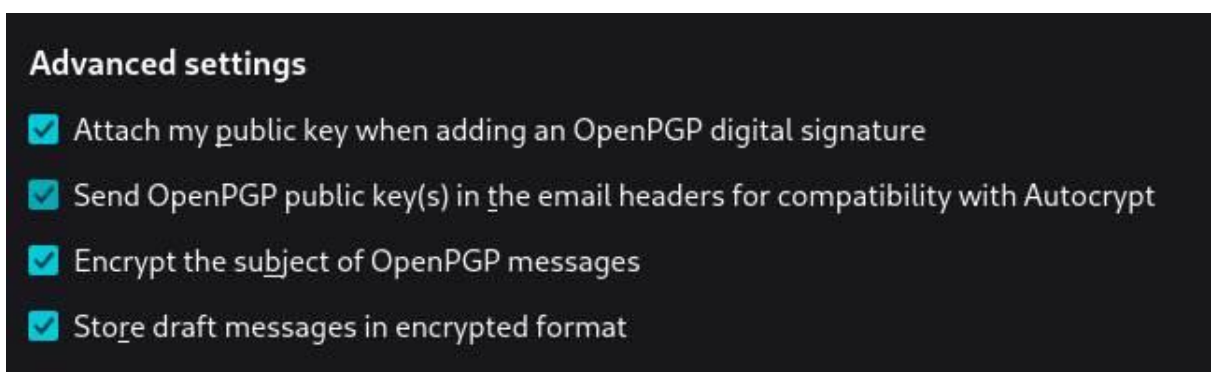
Importamos o creamos nuestra clave personal



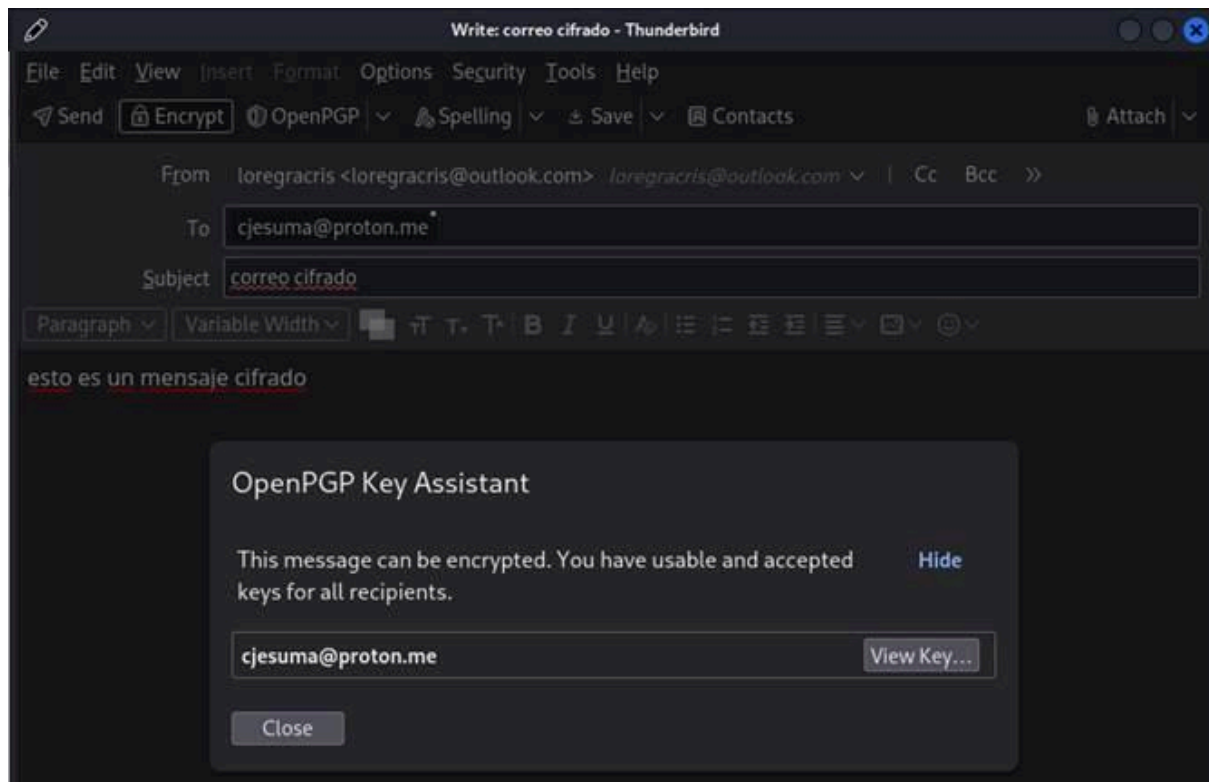
Importamos la clave pública asociada al correo al cual queremos enviar el mensaje cifrado



Para automatizar todo y no tener que configurar todo cada vez que mandemos un mensaje, debemos asegurarnos de tener tildados los siguientes checkboxes que aparecen al final de la pestaña de Cifrado Punto a Punto o End-To-End-Encryption.



Al intentar enviar un mensaje puede suceder que tengamos que importar la clave pública de la persona a la que queremos mandar el correo. Esto es así porque queremos establecer un canal seguro de comunicación, es decir: Nosotros generamos una clave privada y otra pública al igual que la persona con la que queremos compartir mensajes, ambos intercambiamos la clave pública. Con la clave pública de la otra persona podremos cifrar los mensajes que enviaremos para que descifre con su clave privada y viceversa, una vez configurados e importadas las claves correctamente en el cliente de correo, será algo automatizado. De esta forma podremos estar seguro que al mandar correos a ciertas direcciones de email estos se enviarán cifrados y al recibirse se descifrarán.

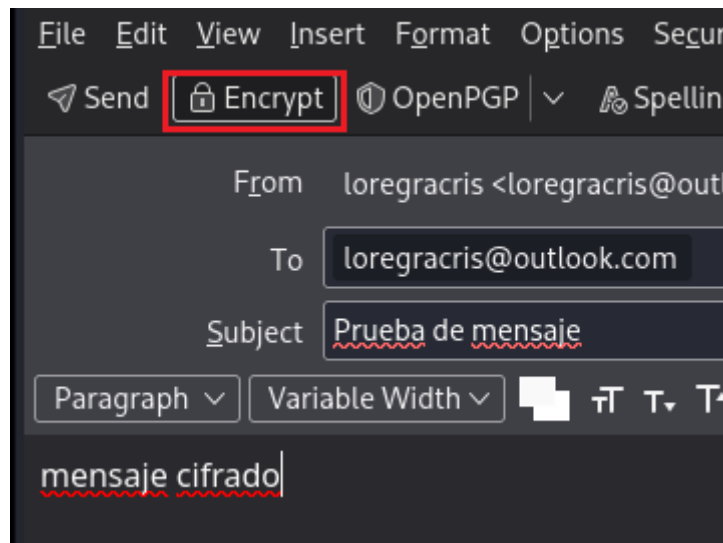


La imagen de arriba muestra a quién enviaremos el correo cjesuma@proton.me, desde nuestra cuenta de correo loregracris@outlook.com.

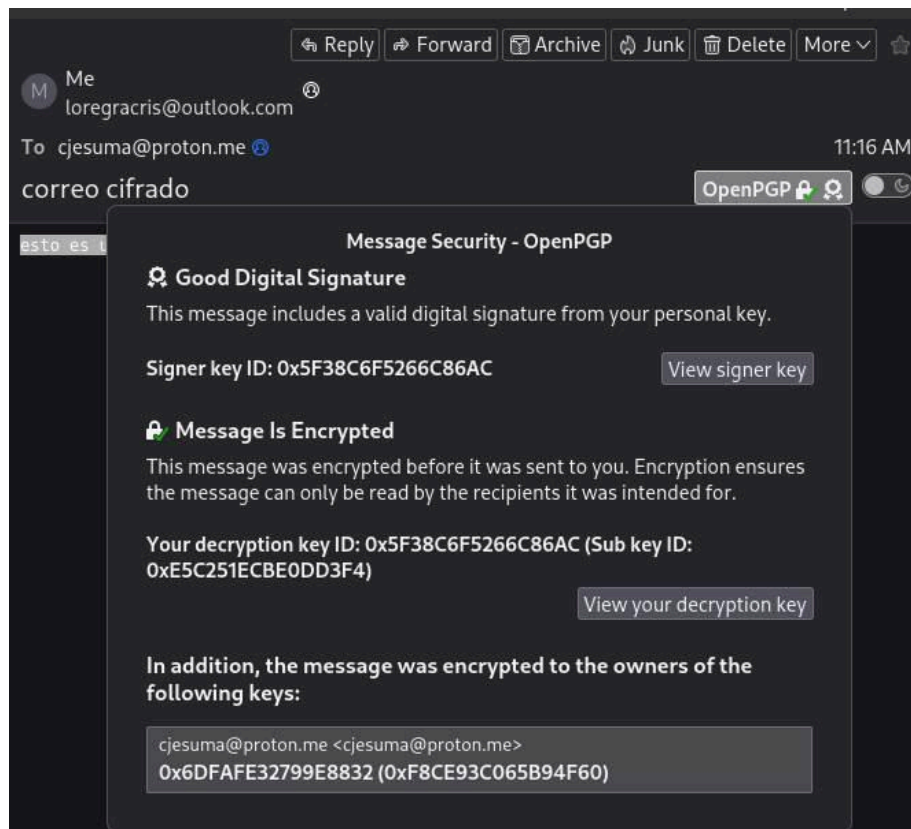
Abajo se ve como el envío cifrado está disponible.



Debemos asegurarnos de que la opción de cifrado esté habilitada



Tanto al recibir como al mandar veremos un check verde que indicará que se está usando cifrado con OpenPGP similar al candado para SSL en Páginas web y podremos ver más información al respecto, cómo la validez de la firma y la confirmación de que el mensaje está cifrado.



Si abrimos el correo desde un cliente que no esté configurado para usar OpenPGP como por ejemplo el cliente web de Outlook, podremos ver que en realidad el correo no se envía como texto plano, sino como 2 archivos adjuntos. En uno de ellos aparecerá la versión del protocolo usado, y en el otro el mensaje cifrado.

LL

LoreGracris LoreGracris

To: cjesuma@proton.me

PGP/MIME version ide...
Downloaded

OpenPGP encrypted ...
Downloaded

2 attachments (7 KB) [Save all to OneDrive](#) [Download all](#)

OpenPGP encrypted message.asc: Bloc de notas

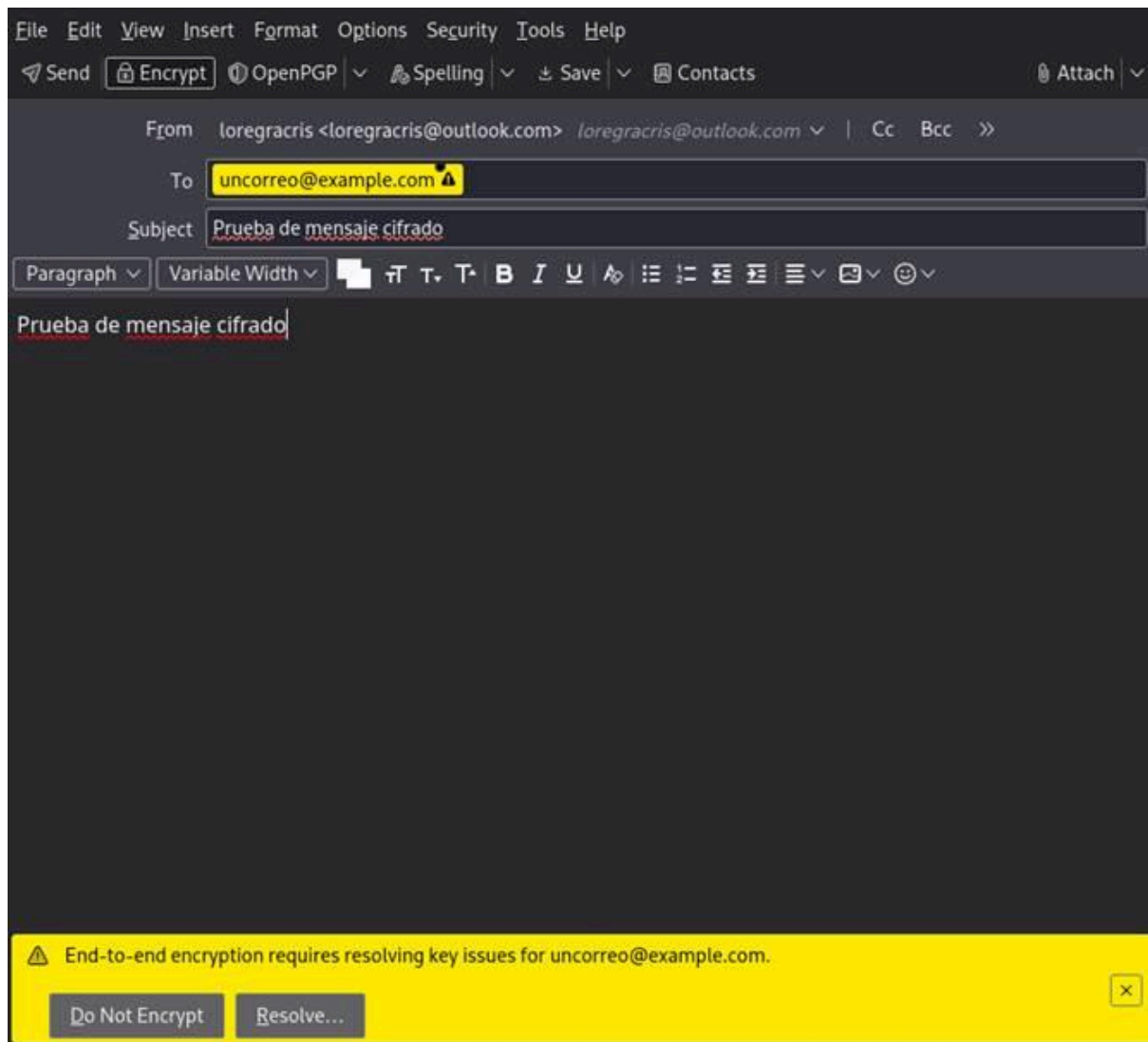
Archivo Edición Formato Ver Ayuda

-----BEGIN PGP MESSAGE-----

wcDMA+XCUey+DdP0AQv/SvWpb8FJRgtTVh/AguxGaXwn+Av11WuZgWoiBL+xncLNGAoAMlw/L
VuckfbSM9guybxCHZt3n+qMPj2VyYrdF8c+8XaGm4cmo3fANPUhQB6Uni37AoA6Q7mxpVVD\
CF2Ws1QZXxVQhqNv6mUwR/OX4D1JdZWslzLF6XIocLejg5TH/1ZmxDZ2BGC0BWFx6TrUJG3OF
siKb4yLxJ8w/1HmS1/mhbw5NmxxbLIdiuEifXhZNG8ce1Vxp1LnH9MsdeIYmR/8yZq5y1GYUC
5MSPMF0JoLDow1dWj1f9f51Ddo08+uLuUwQ7ErmsQZp5FHuzHuRbwGUHuZ8LOIwiBdgZ3LxG8
19W0276m1V2RKDJiIZ2Tx8CTibVvOW3ttUarXyhaFH0vD1a2cj8TtIKHaxZmYetEPa599swq:
BPr3l1Q+pRN8rUnsBKPqa1BTZZiSPjGUTqIzM/2xbvrbk5vyUldI/MFwuD1a6hiAI74PqdpKr
wV4D+M6TwGW5T2ASAQdAQAJ1fb608KIdV+MayUhL+DnMpdWfwzr0jIrgcpffBE0w4e5kbDah\
0z3ldmKOSia4QbS/H7Gwb+f/gZeZ7DPq/K64GwQQNBEnDL9NiQew0s6BAayad6UzvTeASL+6F
fT6vUuqSWqBKWsnYJ91VsEhFZOv0tcfac+DqSZoNJY/X7n1YTRjHxeGE3W7Th0zPC9hJ6ES92
MeeJVF45v6GoAwXtwa7uoaVpYfcJgBD6z0Pctz8XLboJwvSxjSpkiFJB10l+ tqU0cDWuIot:
rJr7QklpFRzf6BG6TA5B6Gd3CDcYgoxXtQskWxV2MojD03+1wYMn+TGFHfyeMqBgLWvVR24h:
VjprBSudEhey8AFx0W9jY0dZG14j439hyn+9ZW5qCk/0egGU5MoHwI1YcP82x/D/B12im71it
FQMJKk9Jzg05QHjECMoIX776xGLD19BBChg9jiPwBcv0d2SL0Cem8Q+121Aq/d9gsRef46AYt
LrWzV158UqTZbGW2RqP4s5Ps95+8Scv0KK/ufP7kySgOfwk1+IHJLX1GYJ1/QnxBDy+alv7Bc
1BCYRdfd8G6WT1vhzbLNDm+gzmnUB0dKjhgTjhd3ceZmvjUBKxaw02Vc+KYnk6+X8fQ8qjdWL
AK1gHT35jPJmVdDhHr9DgBKk7LNzgKf1GitRiSvgsAGbdFRiKeD5m/LmEz4MX0ieaigLHOHa:
RmSndzgp9TPkwCfUlvM+DwwusBS3B2pF+1eNyH1oe17gGh2HMr939urkgp+WsnN9zuEJPn/bl
c9+AsRODHlrLAVdnx0+HaDBEVIslmROOLtgMb1aWy5osaS40pgKF8Pr5vHePjCpne/1hI1d:
-----END PGP MESSAGE-----

Línea 1, columna 1 100% Windows (CRLF) UTF-8

Si por el contrario intentamos enviar un correo a una dirección de la cual no disponemos de su clave pública, Thunderbird nos avisará que no es posible enviar el mensaje cifrado a menos que importemos dicha clave. Aunque si nos dejará enviarlo sin cifrar.



Resolución de errores:

- "No se puede cifrar el mensaje"
 - - Verificar que tienes la clave pública del destinatario
 - - Comprobar que la clave no ha expirado
 - - Confirmar que el destinatario tiene tu clave pública
- "Firma no verificada"
 - - Re-importar la clave pública del remitente
 - - Verificar el fingerprint de la clave
 - - Actualizar desde servidor de claves