Pesquisa sobre Criptografia

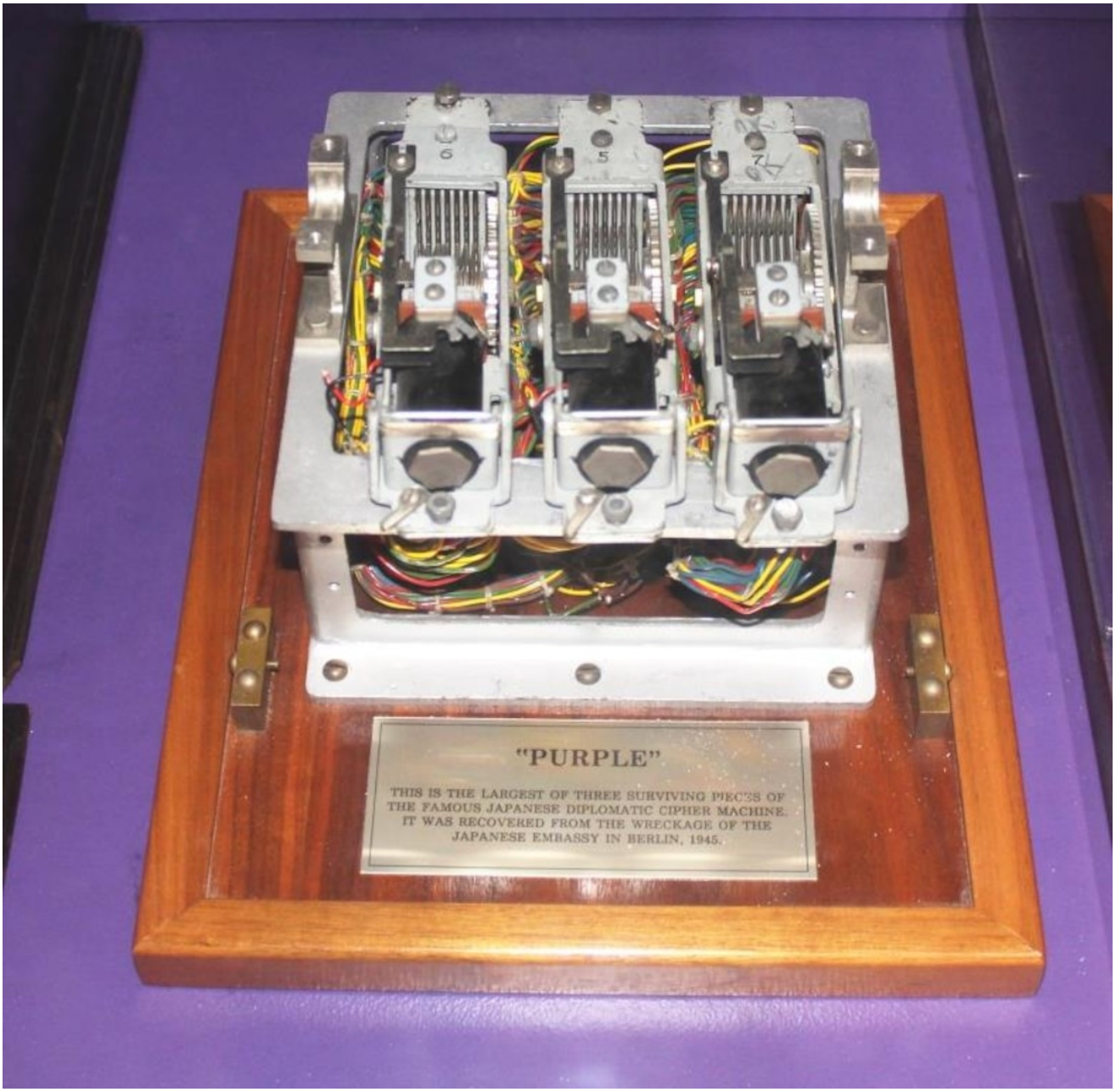
Citala

Cítala foi um sistema de criptografia utilizada pelos espartanos para envio de mensagens secretas. A cítala é formada por dus varas de espessura variável e uma tira de couro ou papiro. A mensagem era então escrita ao longo do comprimento do bastão, uma letra por volta. Ao desenrolar a tira, as letras apareciam embaralhadas e a mensagem se tornava ilegível.

A segurança do sistema dependia inteiramente do diâmetro exato do bastão. Quem possuísse um bastão com o mesmo diâmetro poderia enrolar a tira e ler a mensagem corretamente. Quem não tivesse, veria apenas uma sequência aparentemente aleatória de letras. Este é um primitivo, porém eficaz, exemplo de uma chave simétrica física.

Importância Histórica: A Scitala é um marco por ser uma das primeiras formas de transposição (as letras mudam de posição, mas não sua identidade) em um dispositivo físico. Foi utilizada para comunicação militar em Esparta, demonstrando a necessidade prática de secretismo em operações de guerra desde a Antiguidade.

Type B Cipher Machine - PURPLE



Type B Cipher Machine, também conhecida como Purple, foi uma máquina eletromecânica de criptografia usada pelo Ministério das Relações Exteriores do Japão (Japanese Foreign Office) de fevereiro de 1939 até o fim da Segunda Guerra Mundial. Era utilizada para criptografar as mensagens diplomáticas mais sensíveis do Japão, comunicando-se com embaixadas e consulados em todo o mundo, incluindo em Berlim e Washington D.C.

A máquina PURPLE usava um princípio diferente das máquinas baseadas em rotores (como a Enigma alemã); ela utilizava comutadores de passo (stepping-switches), um tipo de componente mecânico comum em centrais telefônicas da época, sugerido pelo engenheiro Eikichi Suzuki.

O mecanismo criptográfico tinha uma falha de projeto herdada da máquina anterior (RED), que a equipe de criptanálise americana soube explorar:

Divisão do Alfabeto: As 26 letras do alfabeto (usado para telegrafia) eram separadas, por meio de um painel de conexões (plugboard), em dois grupos:

- Grupo dos Seis (Sixes): Seis letras com criptografia mais fraca.
- Grupo dos Vinte (Twenties): Vinte letras com criptografia mais complexa.

Criptografia dos Seis:

- O grupo dos seis era cifrado por um sistema polialfabético simples, com 25 alfabetos permutados fixos.

Criptografia dos Vinte:

- O grupo dos vinte passava por três estágios sucessivos (I, II e III) de comutadores de passo, conectados em série. Cada estágio realizava uma substituição complexa.

Chave e Movimento:

- O sistema de chave (parcialmente trocado diariamente e parcialmente a cada mensagem) era mais complexo que o da máquina RED, mas permitia que os criptanalistas americanos o atacassem. O movimento dos comutadores dos vinte era controlado, em parte, pelos comutadores dos seis, em uma ordem (rápido, médio, lento) que mudava a cada mensagem.

Criptografia de Chaves Simétricas

AES (Advanced Encryption Standard)

AES é um algoritmo de criptografia que utiliza chave simétrica para cifrar e decifrar os dados. É um criptogradia de bloco, ous seja, opera em blocos de tamanho fixo (128 bits, ou 16 bytes) usando chaves de 128, 192 ou 256 bits, ou seja, recebe como entradas um bloco de 128 bits (a mensagem) e uma chave do tamanho escolhido, e devolve uma saída também de 128 bits (a cifra). A função inversa (decifragem) recebe como entrada um bloco de 128 bits (a cifra) e devolve como saída um bloco de 128 bits. Se a chave for a chave correta, essa saída será idêntica à mensagem original.

O algoritmo AES utilizada de operações de multiplicação, adição e XOR com matrizes para poder cifrar os dados originais. Esse processo é realizado em várias rodadas para garantir um segurança da informação e dificutar possíveis tentativas de quebrar a criptografia por força bruta.

Algoritmo em pseudo código do AES:

```
Cifra(byte entrada[4*Nb], byte saida [4*Nb], palavra p[Nb*(Nr+1)])
   inicio
      byte estado[4,Nb]

      estado = entrada

      AddRoundKey(estado, p[0,Nb-1])

      para rodada = 1 faça 1 até Nr-1
         SubBytes(estado)
         ShiftRows(estado)
         MixColumns(estado)
         AddRoundKey(estado, p[rodada*Nb, (rodada+1) *Nb-1])
      fim para

      SubBytes(estado)
      ShiftRows(estado)
      AddRoundKey(estado, p[Nr*Nb, (Nr+1) *Nb-1)

      saida = estado
   fim
```

Fonte: Adaptado de NIST (stackoverflow)

Onde é usado:

- é comumente usada de várias maneiras, incluindo segurança sem fio, segurança do processador, criptografia de arquivo e SSL / TLS.

Exemplo de código em python

In [22]:
```python
import os, base64
from cryptography.hazmat.primitives.ciphers.aead import AESGCM

# 1) Gerar chave
key = AESGCM.generate_key(bit_length=256) # 256 bits

aesgcm = AESGCM(key)

# 2) Nonce
# Usado para gerar valores diferentes para para cada cifragem
nonce = os.urandom(12)  # 96 bits

dados = "Essa mensagem vai ser cifrada com AES".encode("utf-8")

# Associated_data -> cabeçalho para adicionar mais uma cama de criptografia
# Opcional
aad = None

# 4) Cifrar
ciphertext = aesgcm.encrypt(nonce, dados, aad)
```

```
print("Key:", base64.b64encode(key).decode())
print("Nonce:", base64.b64encode(nonce).decode())
print("Texto cifrado:", base64.b64encode(ciphertext).decode())

# 5) Decifrar
decrypted = aesgcm.decrypt(nonce, ciphertext, aad)
print("Texto decifrado:", decrypted.decode("utf-8"))
```

```
Key: AxjCt2gKmPkQ4lNvFlNY3PnnZKpI0/QmyIEtbTaAU/I=
Nonce: 4RrZKxidnhSI4Eox
Texto cifrado: ttxku9kg4CBliyAbAqQF1jT9I5L7HK8wV+TsrHVsbkxh646IsHbG7jxRkwT7WSLQSFOlASk=
Texto decifrado: Essa mensagem vai ser cifrada com AES
```

ChaCha20 (AEAD)

O ChaCha20 é algoritmo de criptografia de fluxo (stream cipher) desenvolvido por Daniel J. Bernstein em 2008, como uma variante otimizada do algoritmo Salsa20. Ele é rápido, seguro e amplamente utilizado em protocolos modernos de criptografia.

O ChaCha20 transforma uma chave secreta e um nonce (número aleatório único) em um fluxo pseudoaleatório de bits, que é então é realizado uma operação XOR com o texto original para gerar o texto cifrado (ou vice-versa, no caso de descriptografia).

Ele não cifra diretamente os dados como o AES (bloco por bloco), mas gera um keystream de 64 bytes por vez e mistura os dados com ele.

Onde usar:

- É usado em TLS (HTTPS), VPNs, SSH, OpenSSH, WireGuard e até no TLS 1.3 como alternativa ao AES.

Exemplo de código em python

In [28]:
```
from cryptography.hazmat.primitives.ciphers import Cipher, algorithms, modes
import os

# Gerar chave
key = os.urandom(32)

# Nonce de 16 bytes
nonce = os.urandom(16)

# Mensagem de exemplo
mensagem ="Essa mensagem vai ser cifrada com Chacha20".encode("utf-8")

# Criação do cifrador
algorithm = algorithms.ChaCha20(key, nonce)
cipher = Cipher(algorithm, mode=None)
encryptor = cipher.encryptor()

# Cifrar
cifrado = encryptor.update(mensagem)
print("Key:", base64.b64encode(key).decode())
print("Nonce:", base64.b64encode(nonce).decode())
print("Texto cifrado:", cifrado)

# Decifrar (usa a mesma chave e nonce)
decryptor = cipher.decryptor()
decifrado = decryptor.update(cifrado)
print("Texto decifrado:", decifrado.decode("utf-8"))
```

```
Key: 5jCaI5ihWAAbRcZq5eTSy/EkG/DtmczSL8fyNpDD+7A=
Nonce: 5veXIOocfP+9hHmXOB8hvQ==
Texto cifrado: b'\xcck\xfa\x84a\xdcbk\x17\x14\xf1A\xc1\xbf\x01\x92&\xd8\x9a\x1f\xb3\x8b`\xde\x9eLG&\xc2W\x17\xcb
do\x1e\xe8|8w9&\xc4'
Texto decifrado: Essa mensagem vai ser cifrada com Chacha20
```

Criptografias de Chaves Assimétricas

RSA

Algoritmo RSA é um algoritmo de criptografia criado por Rivest, Shamir e Adleman. A segurança desse algoritmo se baseia na dificuldade de fatoração de números primos grandes, pois para um atacante saber a chave privada D, ele teria que saber quais foram os primos P e Q que foram utilizados para achar o N, para assim calcular o Phi(N). Como o RSA utiliza números de 2048 bits, significa então 2^{2048} possibilidades, ou seja, 10^{617} sendo extremamente maior do que a quantidade de átomos no universo observável, que estima-se ser na ordem de grandeza de 10^{81}. Fazer a multiplicação de dois números primos de 2048 bits é facil (computacionalmente falando), porém para achar os números primos originários de um N de 4096 bits é preciso fatorar, algo que é computacionalmente inviável, levando milhares de anos para achar os dois números primos que foram multiplicados para dar o N. Sendo assim um algoritmo seguro para comunicação na internet.

As aplicações deste algoritmo vão desde o Login que você faz no Facebook, até as chaves que são utilizadas por um servidor SSH.

Exemplo de execução do algoritmo

1º Passo:

Para gerar a chva pública precisa primeiramente gerar dois múmeros que vamos chamar de P e Q. P e Q precisam ser primos e muito grandes. Contudo, para esse exemplo vou usar número s pequenos para facilitar os cálculos.

- P = 17 e Q = 41.

2º Passo:

Agora calcularemos o N, sendo a multiplicação de P e Q.

- N = P * Q = 17 * 41 = 697
- N = 697

3º Passo:

Utilizaremos agora a função totiente de Euller, também chamada de phi, no N. Como P e Q são primos, phi é P -1 * Q-1.

- Phi(N) = Phi(P) * Phi(Q)
- Phi(N) = (P-1) * (Q-1)
- Phi(N) = (17–1) * (41–1) = 640
- Phi(N) = 640.

4º Passo:

Agora teremos que achar um outro número Aleatório E, que tem que satisfazer as condições: ser maior que 1 e menor que Phi(N), e também ser primo entre Phi(N).

- 1 < E < Phi(N) => 1 < E < 640
- mdc(640, E) == 1
- E = 13
- 1 < 13 < 640 e mdc(640, 13) == 1 (Condições atendidas)

5º Passo:

A chave pública é composta pelo N e o E, 697 e 13 respectivamente. Agora criptografar o nosso texto que será enviado. Transformaremos os caracteres em ASCII, podendo assim fazer operações aritméticas com os valores das letras.

- Mensagem = ‘oi’
- ASCII(‘o’) == 111
- ASCII(‘i’) == 105

6º Passo:

Para criptografar devemos seguir o seguinte algoritmo: Para cada caracter, elevaremos seu valor em ASCII por E e faremos a operação modular com N.

- Valor cifrado de ‘o’ == 111 ^ 13 = 388328016259855395064461231
- 388328016259855395064461231 mod 697 = 110
- Valor cifrado de ‘o’ == 110
- Valor cifrado de ‘i’ == 105 ^ 13 = 1885649142323235607910156 25
- 1885649142323235607910156 25 mod 697 = 318
- Valor cifrado de ‘i’ == 318
- Mensagem cifrada = [110, 318] é enviada para o servidor.

7º Passo:

Agora devemos achar a chave privada que será usada para descriptografar a Mensagem cifrada enviada pelo cliente. A chave privada é chamada de D e é encontrada seguindo o algoritmo:

- D * E mod Phi(N) == 1
- D * 13 mod 640 == 1
- D = 197

8º Passo:

Usaremos o D para descriptografar a Mensagem cifrada, para cada número da mensagem, iremos elevalo por D e fazer a operação modular por N.

- Mensagem cifrada = [110, 318]
- 110 ^ 197 mod (N) == 111
- 317 ^ 197 mod (N) == 105
- 111 == ASCII('o')
- 105 == ASCII('i')

Exemplo de código em python

In [34]:
```python
import base64
from cryptography.hazmat.primitives.asymmetric import rsa, padding
from cryptography.hazmat.primitives import hashes

# 1️⃣ Gerar chave privada
private_key = rsa.generate_private_key(
    public_exponent=65537,
    key_size=2048
)

# 2️⃣ Extrair chave pública
public_key = private_key.public_key()

# 3️⃣ Mensagem original
mensagem = "Mensagem secreta com RSA!".encode("utf-8")

# 4️⃣ Cifrar com chave pública
cifrado = public_key.encrypt(
    mensagem,
    padding.OAEP(
        mgf=padding.MGF1(algorithm=hashes.SHA256()),
        algorithm=hashes.SHA256(),
        label=None
    )
)

# Extrair números das chaves
private_numbers = private_key.private_numbers()
public_numbers = public_key.public_numbers()

# Chave privada (expoente d)
private_bytes = private_numbers.d.to_bytes((private_numbers.d.bit_length() + 7) // 8, "big")
# Chave pública (módulo n)
public_bytes = public_numbers.n.to_bytes((public_numbers.n.bit_length() + 7) // 8, "big")

print("Private Key (base64):", base64.b64encode(private_bytes).decode())
print("Public Key (base64):", base64.b64encode(public_bytes).decode())
print("Texto cifrado:", cifrado)

# 5️⃣ Decifrar com chave privada
decifrado = private_key.decrypt(
    cifrado,
    padding.OAEP(
        mgf=padding.MGF1(algorithm=hashes.SHA256()),
        algorithm=hashes.SHA256(),
        label=None
    )
)
print("Texto decifrado:", decifrado.decode("utf-8"))
```

```
Private Key (base64): AlEFFJlUTgIq7DeOsKyZ0ldp2/nQCYjSg1JuV44LGk/NZBa2K1bJFkOpOStY0l2z2Zi1Su8AxLq7jDa44TGwkIsyxf
IvYBidTAEdI96KYBCmE6ugVSZVsdNFExIkH0zIvpsTAxLTeZi50VBtKjRl+7pD358EwjBSUfcdmpzIKn5XeJx5fw9G1eiZOELG+LfiJvlLhe0CAF
Fo3/TDiRx3qqQAaLJwqEXxN5P16l+ozxAlh8HRZvsCd1X4QQAgAXAbnzO43ioVzVILH3gRlnlGB7RZXG5li0/w7A8u1sbscm9jGoC/1kxeQ9kau6
0IzdiFRGhyM+aTHz0l3cj/pBgNwQ==
Public Key (base64): sIePXZd8031M6h1xL2q3DXchKGLU8MbcUnBrwQH7V8kjsED4RokOmdKZ2OBxoE4OFGhzZBtMIAAVHnFxF2NZ95g0uzm
L/Z2WNkyLmrIVBHjl5k/HE9hc7AuUV+DNOp0vckDHfGE0hDoreO2v0NZhrLmhTkGZ9ZrxM6Cdr74l4vxK7ID+spfslPSE01rmAUHOHsjfM4GRb+O
Wj+LxdaZzGnFxYRSFrCBcXCGzSpiJIyP0S8a2LLceCBfCBKvRxgUYkSiyTBSHNEQMW40OSfOQnTQfI2X8JBUMdKjYJX9mzO/I0ePtlv3Lehi92Ev
6oOKx6iqPlAiS/RbcbHEe6GZoSQ==
Texto cifrado: b'\x91\x8dY\x9c\xa25\xbf\xab\xfe\xe2H S\xb6\xc1j\xfc}\x18B\n\x80\x93\xec\r\x1f\xf7?k\x16\xce`w\x8
a\xc4\x1c`\xd2^\xc0\xf3\xbc\x19\xec6`\x0c\xd3\xfe\xe8\xfa\xbf\x0cP\xacH\xd1TZ\xa9\x90\xe3$\xcdl\xaa\x17S[\xd7\x8
0W\x98\xc6\x0f\x99`\xc7\x13\xe1\xfa7\t\x19MH4\x18\xdf\xad*)^\xa8\xa9\xaeg\xf4\xef\xf2\xb2\xd1\x1c\x06"\xf6\\\x97
P}\xc2b\xect\xa3\xbb\xfa\x1d5\x15\xab+\xd9\x86Q\x06\x125\x11l \xf7\x9bZ<-\xbdcM\n\xd8\xd1\xfat\xbb\x1a\xf4\x7f\x
b1Q\x0f {b\xd28\x8b\x10\x9e8\xee\x97\xd5\xb5\'\xca\xc9\'\x8f\x89z\xf2\xae\xb4\xe4-=\xd8\x9b\xdeY\\\xa61\xc6\xd0\
xaeU\xc7?3=/)\xfe\x95\x89R\x87\x10&\x8b\x0fqU\xe3\xc5c?\x8f\x1dy\xb2"R\xd7\x87\xae\x0cu\xfc4\x0f\xf5\xed\x1d\xed
%\x9b\x1b7\xeb\xf0\x9a\x95\x81\xd6\x15\x8e0\xcc\x88\xaa8\xbb\xe8\x02\xad\xc5\xb0\x97\x7f8\x17\xeae'
Texto decifrado: Mensagem secreta com RSA!
```

ECC

A Elliptic Curve Cryptography (ECC) é uma forma avançada de criptografia de chave assimétrica, baseada em propriedades matemáticas de curvas elípticas sobre corpos finitos. Assim como outros algoritmos de chave pública, a ECC usa duas chaves distintas:

A principal inovação da ECC é que ela oferece alta segurança com chaves menores, comparadas a algoritmos como RSA. Isso significa que o mesmo nível de segurança pode ser alcançado com menos processamento e menos espaço de armazenamento.

Uma curva elíptica é definida pela equação:

y^2 = x^3 + a*x + b

onde a e b são coeficientes que garantem que a curva não tenha pontos singulares. Na criptografia, a curva é escolhida sobre um campo finito (normalmente F_p, números módulo primo p), o que limita os valores possíveis de x e y.

Operações básicas na curva

- **Ponto gerador G**: um ponto específico na curva usado para gerar chaves.
- **Multiplicação de ponto**: operação central da ECC, usada para gerar a chave pública K = k * G, onde k é a chave privada.
- **Problema do logaritmo discreto elíptico (ECDLP)**: dado G e K = k * G, é computacionalmente inviável determinar k. Esse é o pilar da segurança da ECC.

Funcionamento do algoritmo

1º Passo geração de chaves:

- Escolhe-se uma curva elíptica segura e um ponto gerador G.
- Gera-se uma chave privada k aleatória.
- Calcula-se a chave pública K = k * G.

2º Passo 2 ciframento (ECDH - Elliptic Curve Diffie-Hellman)

- Dois usuários (Alice e Bob) querem compartilhar uma chave simétrica.
- Alice envia K_A = k_A * G a Bob.
- Bob envia K_B = k_B * G a Alice.
- Cada um calcula a chave compartilhada:

  S = k_A * K_B = k_A * k_B * G = k_B * K_A

Agora ambos têm a mesma chave secreta, que pode ser usada para cifrar dados com algoritmos simétricos (AES ou ChaCha20).

3º Passo assinatura digital (ECDSA)

- **Assinatura de uma mensagem**: calcula-se o hash da mensagem e usa-se a chave privada para gerar a assinatura com operações na curva.
- **Verificação**: a chave pública e a assinatura são usadas para validar o hash da mensagem. Qualquer alteração na mensagem ou assinatura resulta em falha.

Aplicações práticas

- TLS/HTTPS moderno: certificados ECC em sites seguros.
- Assinaturas digitais: ECDSA para autenticação de software, documentos e blockchain.
- Protocolos de troca de chaves: ECDH em VPNs, SSH e aplicações móveis.
- Blockchain e criptomoedas: Bitcoin, Ethereum e outras usam ECC para gerar endereços e assinar transações.

Exemplo de código em python

In [26]:
```python
from cryptography.hazmat.primitives.asymmetric import ec
from cryptography.hazmat.primitives import hashes
from cryptography.hazmat.primitives import serialization

# Gerar chave privada
private_key = ec.generate_private_key(ec.SECP256R1())

# Extrair chave pública
public_key = private_key.public_key()

# Mensagem
mensagem = b"Mensagem para assinar"

# Criar assinatura
assinatura = private_key.sign(
    mensagem,
    ec.ECDSA(hashes.SHA256())
)
print("Assinatura ECDSA:", assinatura)

# Verificar assinatura
```

```
try:
    public_key.verify(
        assinatura,
        mensagem,
        ec.ECDSA(hashes.SHA256())
    )
    print("Assinatura válida!")
except:
    print("Assinatura inválida!")
```

```
Assinatura ECDSA: b'0D\x02 B,\xc9\x85\x83\xf6Y>\xb6\xd4U\xeb\xa80\xc2]\x8a\xcbI\x04\x99L\x8fS#U]\xb9\\\x12\xb7t\
x02 \x7f\xdf\xb5\x1a\x04|0wq\x92\xedj\x94(\x82_\xd5\xd7\x1e}\x00\xb8\x18\xf6%\x1c\x9f\x98n\xa6\xbe\xac'
Assinatura válida!
```

Fontes

https://pt.wikipedia.org/wiki/RSA_(sistema_criptogr%C3%A1fico)

https://medium.com/@tarcisioma/algoritmo-de-criptografia-assim%C3%A9trica-rsa-c6254a3c7042

https://en.wikipedia.org/wiki/Type_B_Cipher_Machine

https://cryptoid.com.br/criptografia/aes-padrao-de-criptografia-avancado-o-que-e-e-como-funciona/

https://pt.stackoverflow.com/questions/43492/como-funciona-o-algoritmo-de-criptografia-aes

https://securityboulevard.com/2020/04/advanced-encryption-standard-aes-what-it-is-and-how-it-works/