

Ciberseguridad en la nube

Seguridad en redes

Universidad
Zaragoza

Criptografía práctica

Criptología

Ciencia que estudia los principios, métodos y medios del cifrado y descifrado de la información. Comprende dos ramas principales: la Criptografía y el Criptoanálisis.

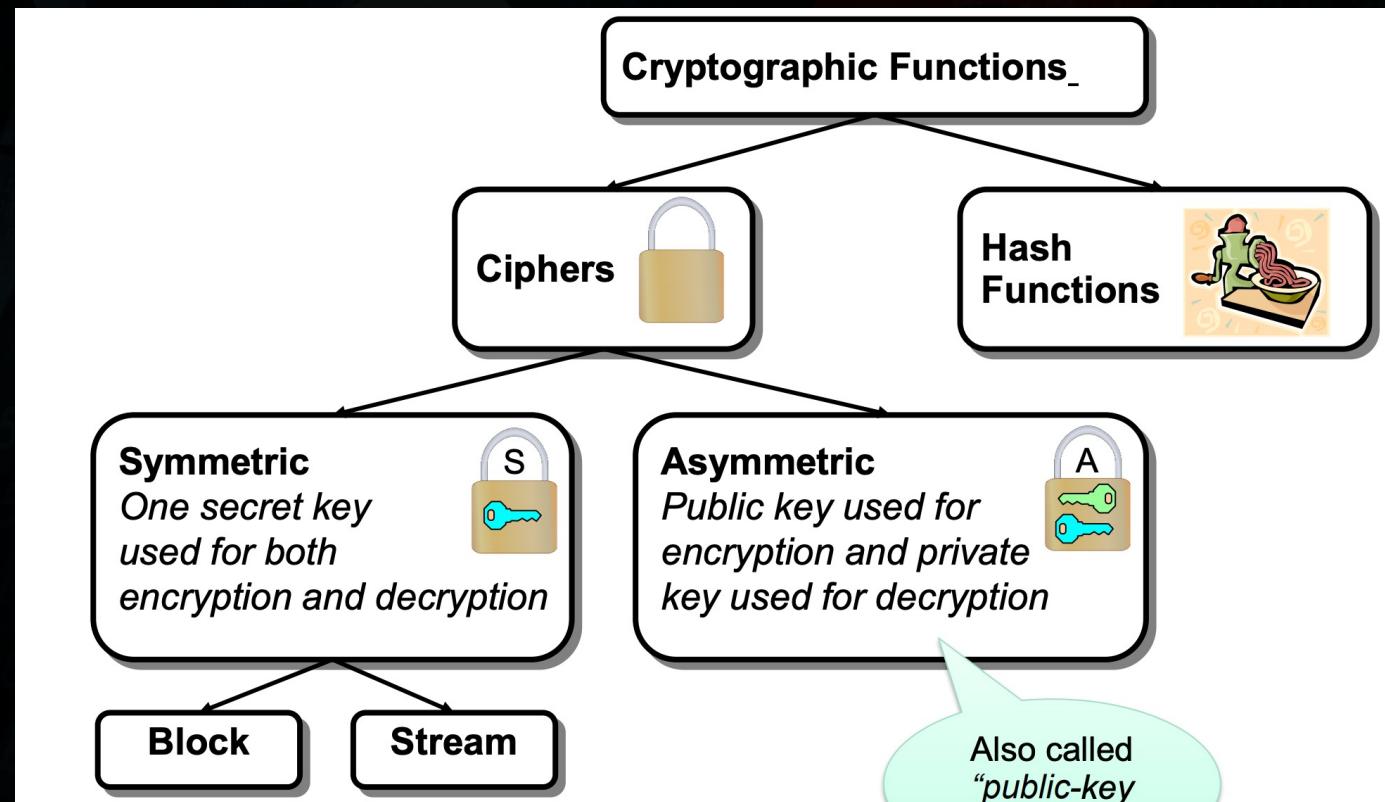
Criptografía

Disciplina que estudia los principios, métodos y medios de transformar los datos con objeto de ocultar la información contenida en los mismos, detectar su modificación no autorizada y/o prevenir su uso no permitido (ISO-7498-2)

Criptoanálisis

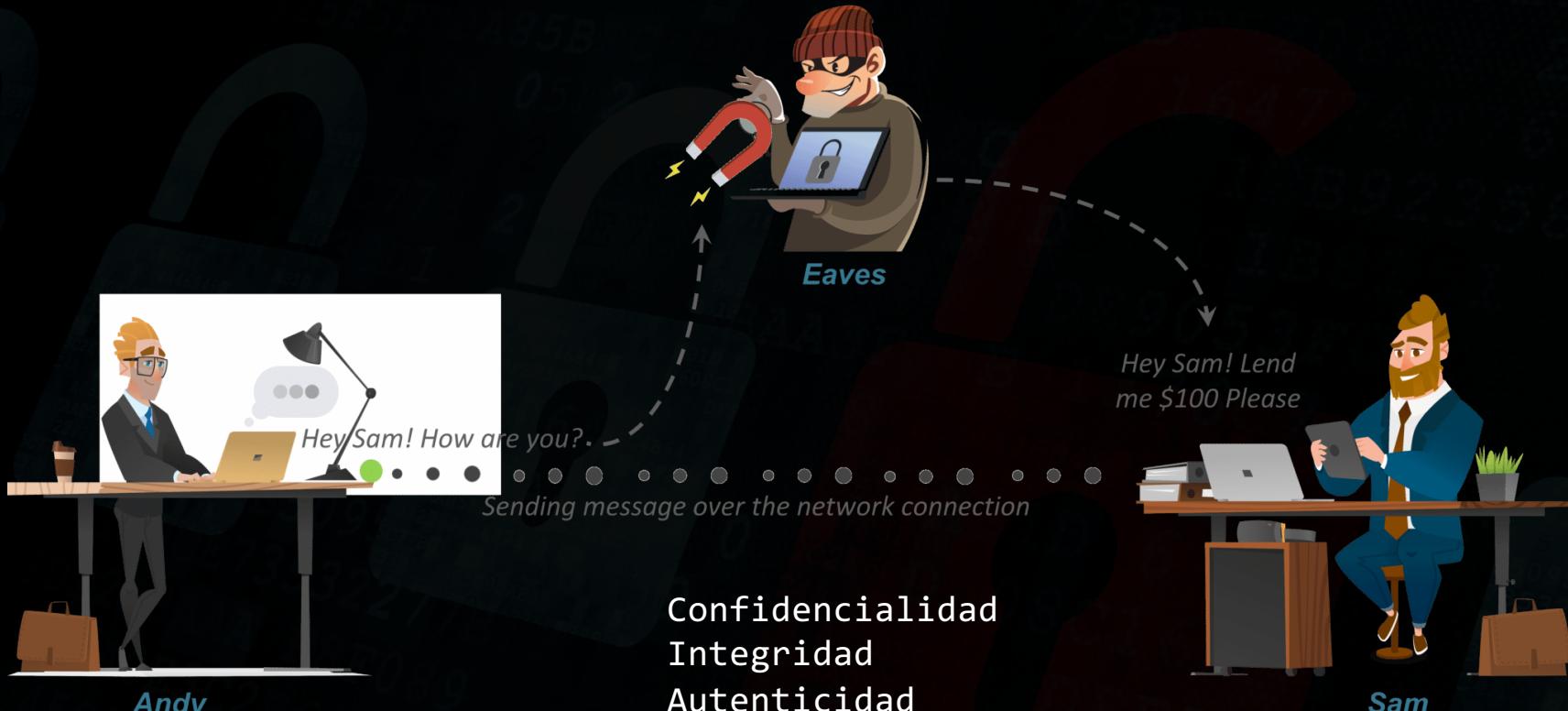
Análisis de un sistema criptográfico y/o sus entradas y salidas para derivar variables confidenciales y/o datos sensibles, incluido texto claro. [ISO-7498-2:1989]

Taxonomía de las funciones criptográficas





Universidad
Zaragoza



* Capacidad para corroborar que es cierta la reivindicación de que ocurrió un cierto suceso o se realizó una cierta acción por parte de las entidades que lo originaron. [UNE-ISO/IEC 27000:2014]



Universidad
Zaragoza

Criptografía Simétrica

Cifrado en bloque vs. Cifrado en flujo

Cifrando la información en trozos

El cifrado en bloque descompone los mensajes de texto plano en bloques de tamaño fijo antes de convertirlos en texto cifrado con una clave.

Ej: DES, 3DES, AES, Blowfish, RC5

Cifrando la información bit a bit

El cifrado en flujo, divide un mensaje de texto sin formato en bits individuales, que luego se convierten individualmente en texto cifrado utilizando los bits de la clave.

Ej: ChaCha20, RC4

<https://www.thesslstore.com/blog/block-cipher-vs-stream-cipher/>

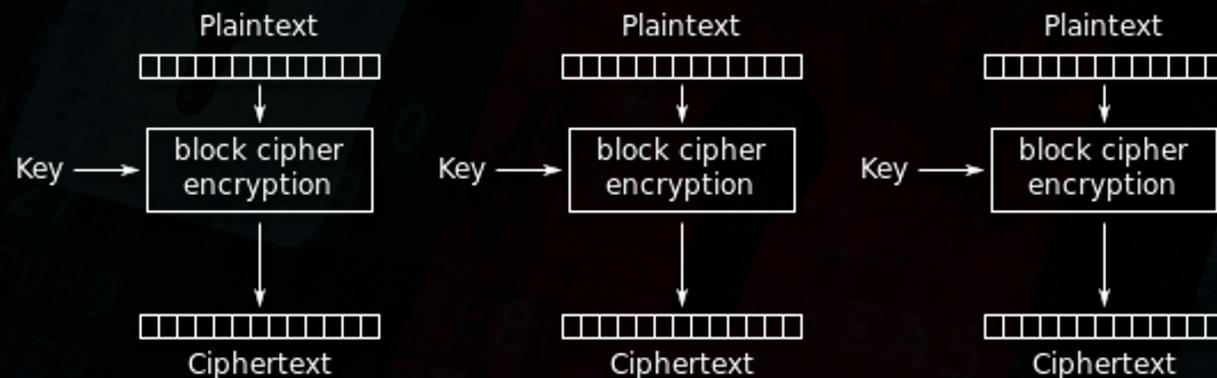
Cifrado en bloque

Información

Plaintext: Cryptography rules de world!

ASCII: 43 72 79 70 74 6F 67 72 61 70 68 79 20 72 75 6C 65 73 20 64 65 20 77 6F 72 6C 64 21 0A

Binary: 01000011 01110010 01111001 01110000 01110100 01101111 01100111 01110010 01100001 01110000
01101000 01111001 00100000 01110010 01110101 01101100 01100101 01110011 00100000 01100100
01100101 00100000 01110111 01101111 01110010 01101100 01100100 00100001 00001010



Electronic Codebook (ECB) mode encryption

From wikipedia

Cifrado en bloque

Modos de operación

- Electronic Code Book (**ECB**)
- Cipher Block Chaining (**CBC**)
- Output Feedback (**OFB**)
- Cipher Feedback (**CFB**)
- Counter Mode (**CTR**)
- Galois Counter Mode (**GCM**)



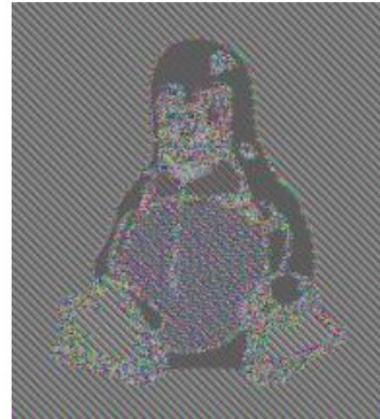
Universidad
Zaragoza

Cifrado en bloque

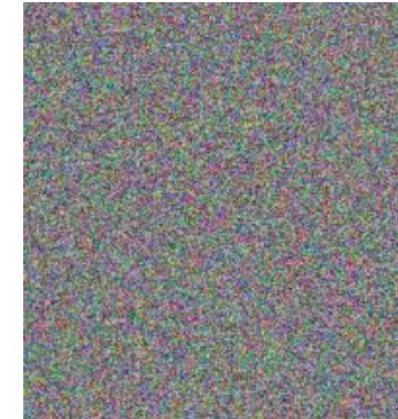
Modos de operación: ¡usa uno seguro!



Plaintext



ECB



CBC

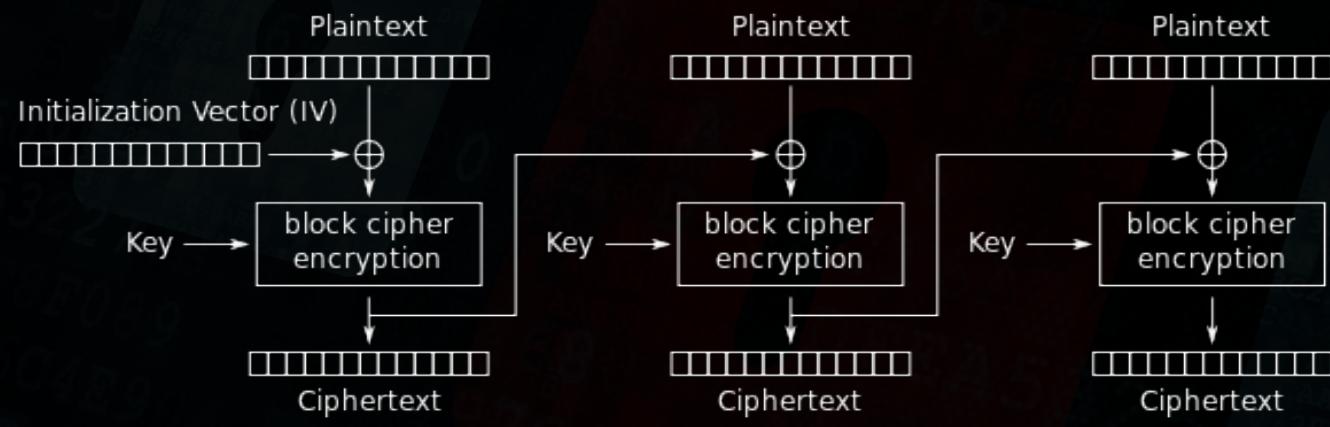
Cifrado en bloque

Información

Plaintext: Cryptography rules de world!

ASCII: 43 72 79 70 74 6F 67 72 61 70 68 79 20 72 75 6C 65 73 20 64 65 20 77 6F 72 6C 64 21 0A

Binary: 01000011 01110010 01111001 01110000 01110100 01101111 01100111 01110010 01100001 01110000
01101000 01111001 00100000 01110010 01110101 01101100 01100101 01110011 00100000 01100100
01100101 00100000 01110111 01101111 01110010 01101100 01100100 00100001 00001010



Cipher Block Chaining (CBC) mode encryption

AES: tamaño bloque 16 bytes

From wikipedia

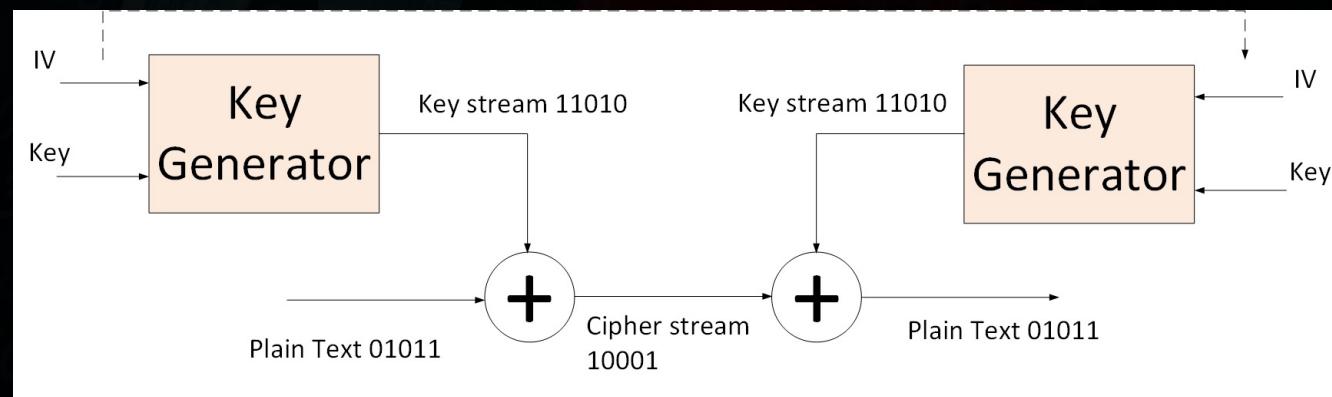
Cifrado en flujo

Información

Plaintext: Cryptography rules de world!

ASCII: 43 72 79 70 74 6F 67 72 61 70 68 79 20 72 75 6C 65 73 20 64 65 20 77 6F 72 6C 64 21 0A

Binary: 0100001101110010011110010111000001110100011011110110011101110010011000010111000001101000
011110010010000001110010011101010110110001100101011100110010000001100100011001010010000001110111
011011110111001001101100011001000010000100001010



XOR

Criptografía de clave pública

One-way functions

Modular power function

Dado $n = pq$, donde p y q son números primos, no hay algoritmos eficientes para encontrar p y q .

Elige un entero positivo b y define $f: \mathbb{Z}_n \rightarrow \mathbb{Z}_n$

$$f(x) = x^b \bmod n$$

Modular exponentiation

Dado un primo p , un generador g y una potencia modular $a = g^x \pmod p$, no hay algoritmo eficiente para encontrar x . $f: \mathbb{Z}_p \rightarrow \mathbb{Z}_p$

$$f(x) = g^x \bmod p$$

Diffie-Hellman (intercambio de claves)

1. **p y g son conocidos**

Dado un primo **p**, un generador **g**

2. El emisor selecciona un numero aleatorio **a**

Calcula $g^a \text{ mod } p$ y lo manda al receptor

3. El receptor selecciona un numero aleatorio **b**

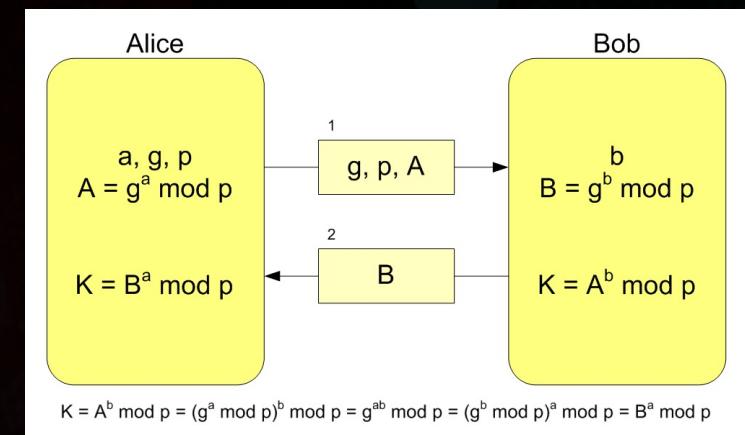
Calcula $g^b \text{ (mod } p)$ y lo manda al receptor

Calcula la clave $K = (g^a \text{ mod } p)^b \text{ mod } p$

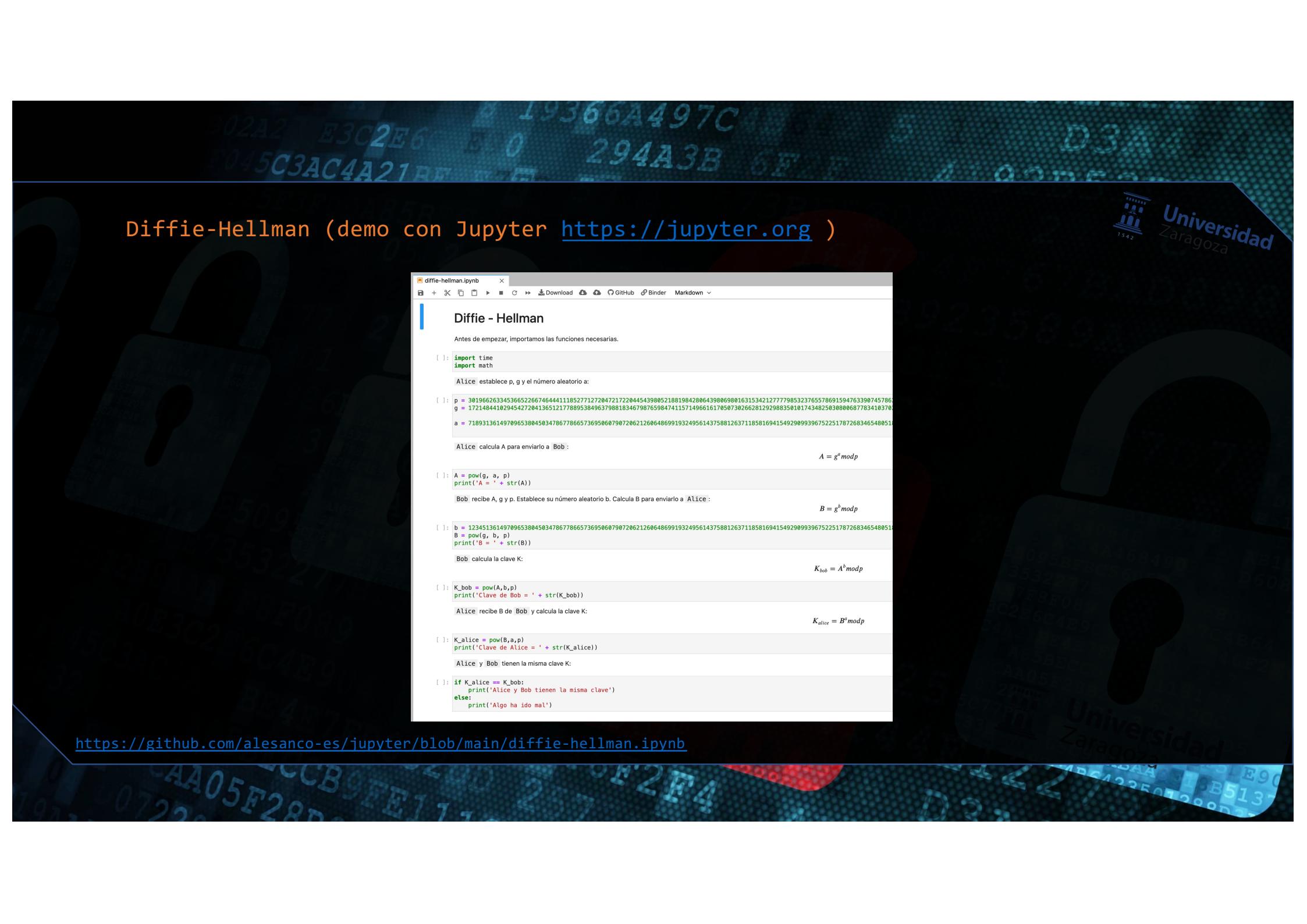
4. El emisor calcula la clave **K**

Calcula la clave $K = (g^b \text{ mod } p)^a \text{ mod } p$

No es viable calcular **a** a partir de g^a



Diffie-Hellman (demo con Jupyter <https://jupyter.org>)



A screenshot of a Jupyter Notebook titled "diffie-hellman.ipynb". The notebook contains Python code demonstrating the Diffie-Hellman key exchange protocol.

```
import time
import math

Alice establece p, g y el número aleatorio a:
p = 3019662633453665226674644411185277127204454398052188198428864398069801631534212777985323765578691594763398745786
g = 1721484410294542720413651217788953849637988183467987659847411571496616170587302662812988835101743482503800687783410370
a = 7189313614970965380450347867786657369506079072062126064869919324956143758812637118581694154929099396752251787268346548051

Alice calcula A para enviarlo a Bob :
A = gamodp

A = pow(g, a, p)
print('A = ' + str(A))

Bob recibe A, g y p. Establece su número aleatorio b. Calcula B para enviarlo a Alice :
B = gbmodp

b = 1234513614970965380450347867786657369506079072062126064869919324956143758812637118581694154929099396752251787268346548051
B = pow(g, b, p)
print('B = ' + str(B))

Bob calcula la clave K:
Kbob = Abmodp

Kbob = pow(A,b,p)
print('Clave de Bob = ' + str(Kbob))

Alice recibe B de Bob y calcula la clave K:
Kalice = Bamodp

Kalice = pow(B,a,p)
print('Clave de Alice = ' + str(Kalice))

Alice y Bob tienen la misma clave K:
if Kalice == Kbob:
    print('Alice y Bob tienen la misma clave')
else:
    print('Algo ha ido mal')
```

<https://github.com/alesanco-es/jupyter/blob/main/diffie-hellman.ipynb>

Aplicaciones de Diffie-Hellman

IPSec (IP Security)

IKE (Internet Key Exchange) es parte de la pila de protocolos IPSec
IKE se basa en Diffie-Hellman

TLS

El intercambio seguro de claves se puede hacer (entre otros) con Diffie-Hellman

RSA (Rivest Shamir Adleman)

1. Bob genera dos números primos grandes p y q

Calcula $n=p \cdot q$

Calcula $\phi = (p-1) \cdot (q-1)$

2. Calcula el exponente de cifrado público e

e debe ser un entero

e debe cumplir que el $\text{mcd}(e, \phi) = 1$

e debe ser menor que phi

3. Clave pública de Bob $\rightarrow P_B^+$

$P_B^+ = (e, n)$

4. Clave privada de Bob $\rightarrow P_B^-$

d debe cumplir que $d \cdot (e-1)/\phi$ tiene resto 0

$P_B^- = (d, n)$

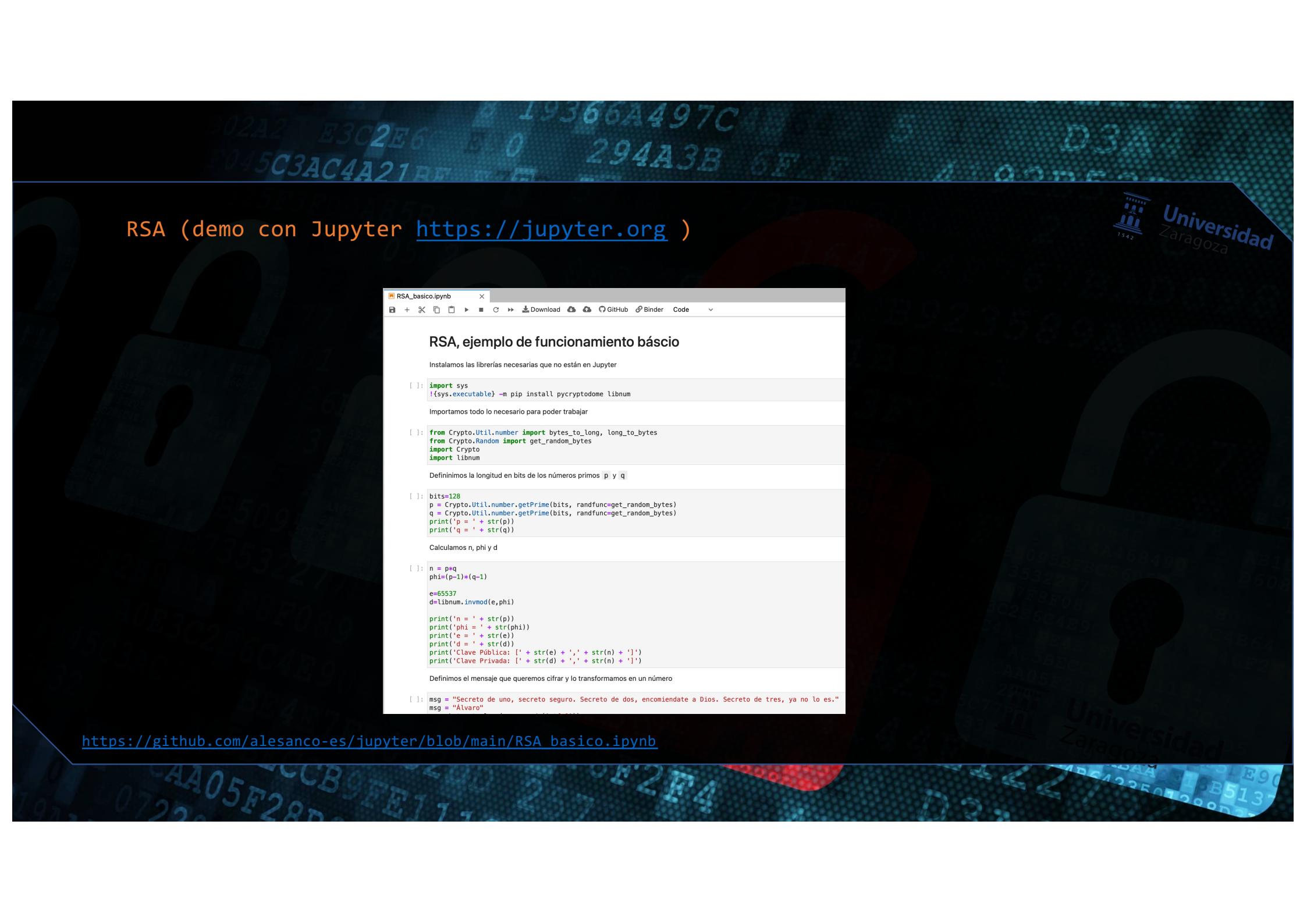
Cifrado mensaje M

$C = M^e \pmod{n}$

Descifrado

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

RSA (demo con Jupyter <https://jupyter.org>)



RSA, ejemplo de funcionamiento básico

Instalamos las librerías necesarias que no están en Jupyter

```
[ ]: import sys  
!{sys.executable} -m pip install pycryptodome libnum
```

Importamos todo lo necesario para poder trabajar

```
[ ]: from Crypto.Util.number import bytes_to_long, long_to_bytes  
from Crypto.Random import get_random_bytes  
import Crypto  
import libnum
```

Definimos la longitud en bits de los números primos p y q

```
[ ]: bits=128  
p = Crypto.Util.number.getPrime(bits, randfunc=get_random_bytes)  
q = Crypto.Util.number.getPrime(bits, randfunc=get_random_bytes)  
print('p = ' + str(p))  
print('q = ' + str(q))
```

Calculamos n, phi y d

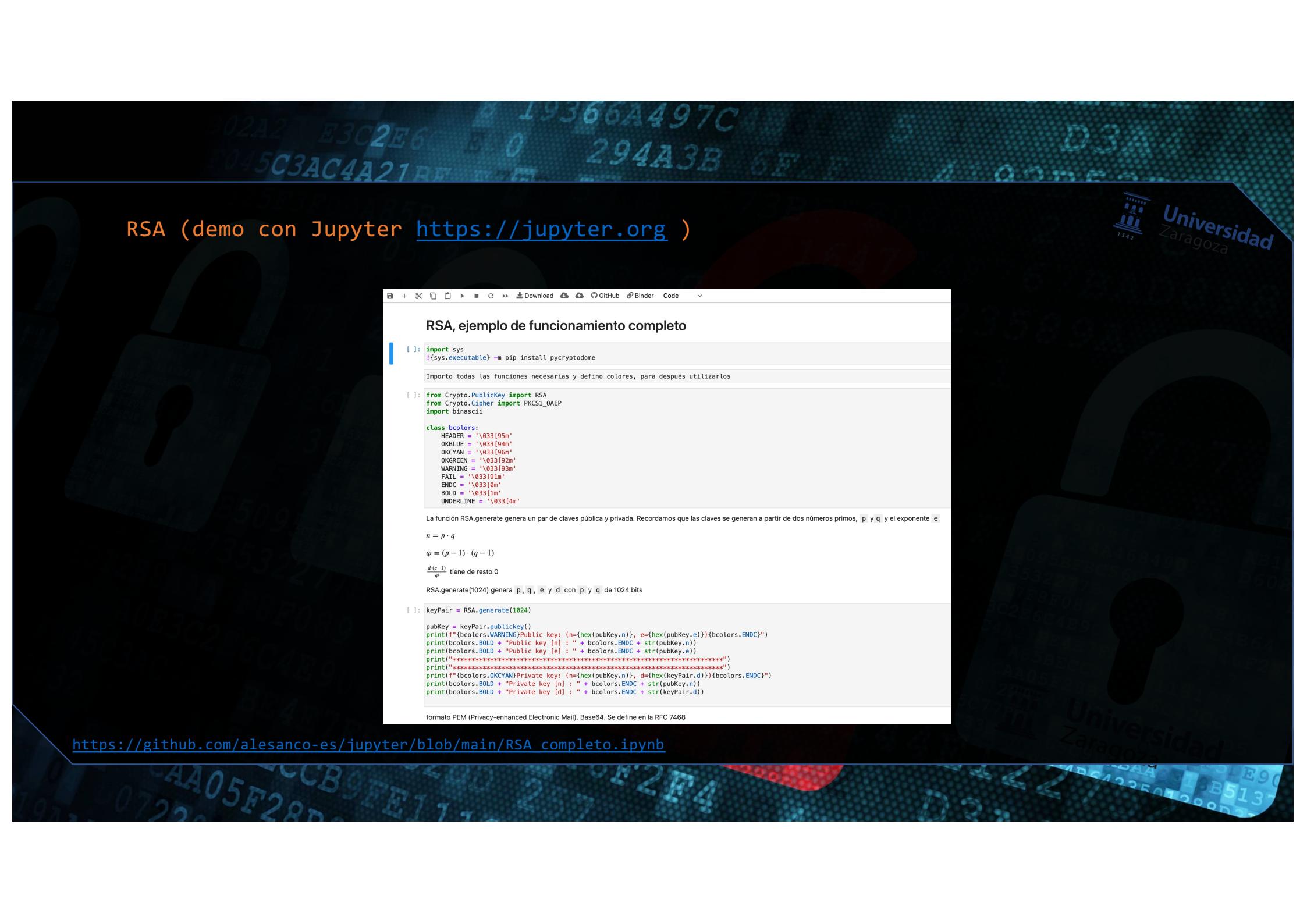
```
[ ]: n = p*q  
phi=(p-1)*(q-1)  
  
e=65537  
d=libnum.invmod(e,phi)  
  
print('n = ' + str(p))  
print('phi = ' + str(phi))  
print('e = ' + str(e))  
print('d = ' + str(d))  
print('Clave Pública: [' + str(e) + ',' + str(n) + ']')  
print('Clave Privada: [' + str(d) + ',' + str(n) + ']')
```

Definimos el mensaje que queremos cifrar y lo transformamos en un número

```
[ ]: msg = "Secreto de uno, secreto. Secreto de dos, encomienda te a Dios. Secreto de tres, ya no lo es."  
msg = "Álvaro"
```

<https://github.com/alesanco-es/jupyter/blob/main/RSA%20basico.ipynb>

RSA (demo con Jupyter <https://jupyter.org>)



RSA, ejemplo de funcionamiento completo

```
[ ]: import sys
!{sys.executable} -m pip install pycryptodome

Importo todas las funciones necesarias y defino colores, para después utilizarlos

[ ]: from Crypto.PublicKey import RSA
from Crypto.Cipher import PKCS1_OAEP
import binascii

class bcolors:
    HEADER = '\x033[95m'
    OKBLUE = '\x033[94m'
    OKCYAN = '\x033[96m'
    OKGREEN = '\x033[92m'
    WARNING = '\x033[93m'
    FAIL = '\x033[91m'
    ENDC = '\x033[0m'
    BOLD = '\x033[1m'
    UNDERLINE = '\x033[4m'

La función RSA.generate genera un par de claves pública y privada. Recordamos que las claves se generan a partir de dos números primos,  $p$  y  $q$  y el exponente  $e$ 

n = p * q
 $\varphi = (p - 1) \cdot (q - 1)$ 
 $\frac{d \cdot (e-1)}{\varphi}$  tiene de resto 0

RSA.generate(1024) genera p, q, e y d con p y q de 1024 bits

[ ]: keyPair = RSA.generate(1024)

pubKey = keyPair.publickey()
print(f'{bcolors.WARNING}Public key: (n={hex(pubKey.n)}, e={hex(pubKey.e)}){bcolors.ENDC}')
print(bcolors.BOLD + "Public key [n] : " + bcolors.ENDC + str(pubKey.n))
print(bcolors.BOLD + "Public key [e] : " + bcolors.ENDC + str(pubKey.e))
print("*****")
print("*****")
print(f'{bcolors.OKCYAN}Private key: (n={hex(keyPair.n)}, d={hex(keyPair.d)}){bcolors.ENDC}')
print(bcolors.BOLD + "Private key [n] : " + bcolors.ENDC + str(keyPair.n))
print(bcolors.BOLD + "Private key [d] : " + bcolors.ENDC + str(keyPair.d))

formato PEM (Privacy-enhanced Electronic Mail), Base64. Se define en la RFC 7468
```

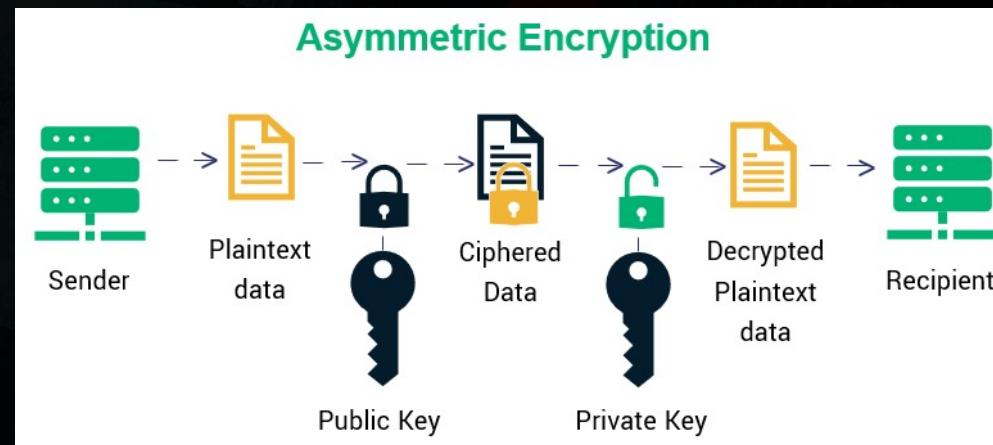
https://github.com/alesanco-es/jupyter/blob/main/RSA_completo.ipynb

Cifrado asimétrico: operación básica

¿Cuánta información cifro de una vez?

RSA is only able to encrypt data to a maximum amount equal to your key size (2048 bits = 256 bytes), minus any padding and header data (11 bytes for PKCS#1 v1.5 padding).

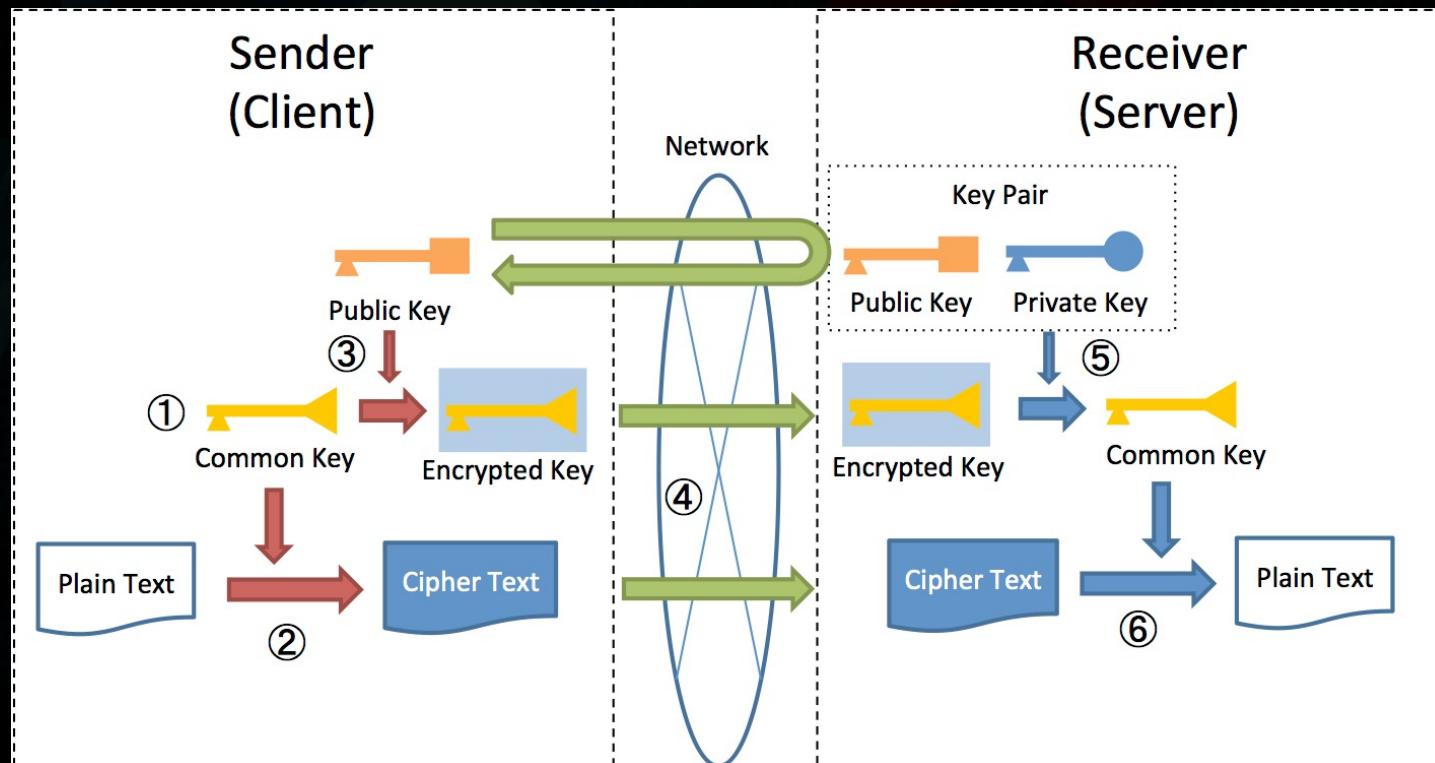
<https://info.townsendsecurity.com/bid/29195/how-much-data-can-you-encrypt-with-rsa-keys>



¿Qué hacemos entonces para cifrar ficheros?

Sistemas híbridos

Sistemas híbridos



Prestaciones de seguridad: comparación

AES Key Size	RSA Key Size	Elliptic curve Key Size
-	1024	163
128	3072	256
192	7680	384
256	15360	512

Funciones de Hash y HMAC

Funciones de hash

Aplicaciones

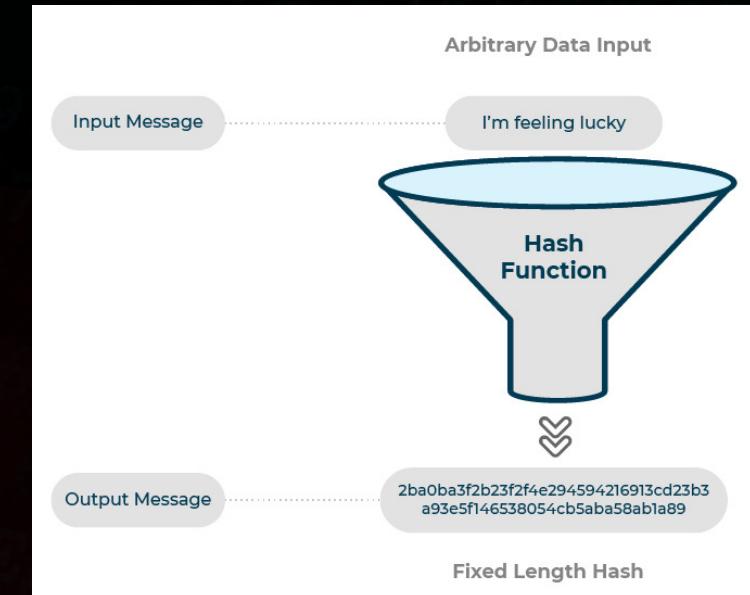
- Protección de passwords
- Comparar ficheros
- Autenticación de descargas
- Generar Message Authentication Codes (HMAC)
- Firmas digitales
- Generación de números pseudoaleatorios
- Derivación de claves

Propiedades

- Fáciles de calcular
- Único sentido
- Resistentes a la colisión**

Funciones usadas con frecuencia

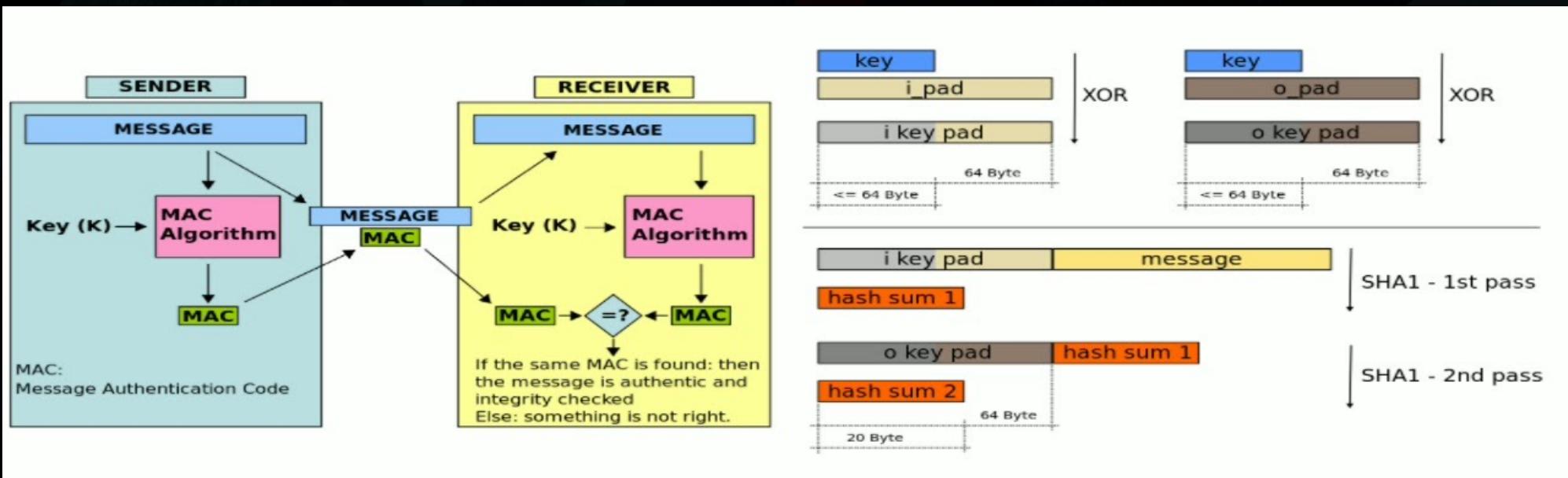
- MD5. 128 bits. Rota
- SHA-1: 160 bits. Poco fiable
- SHA-256, 384, 512. Seguras



Hash Message Authentication Code (HMAC)

Aplicaciones

Protección de integridad
Firma digital (poco útil)



Firmas digitales

Firmas digitales

Aplicaciones

- No repudio
- Verificación origen de los datos
- Integridad de los datos
- Autenticación en intercambios de información

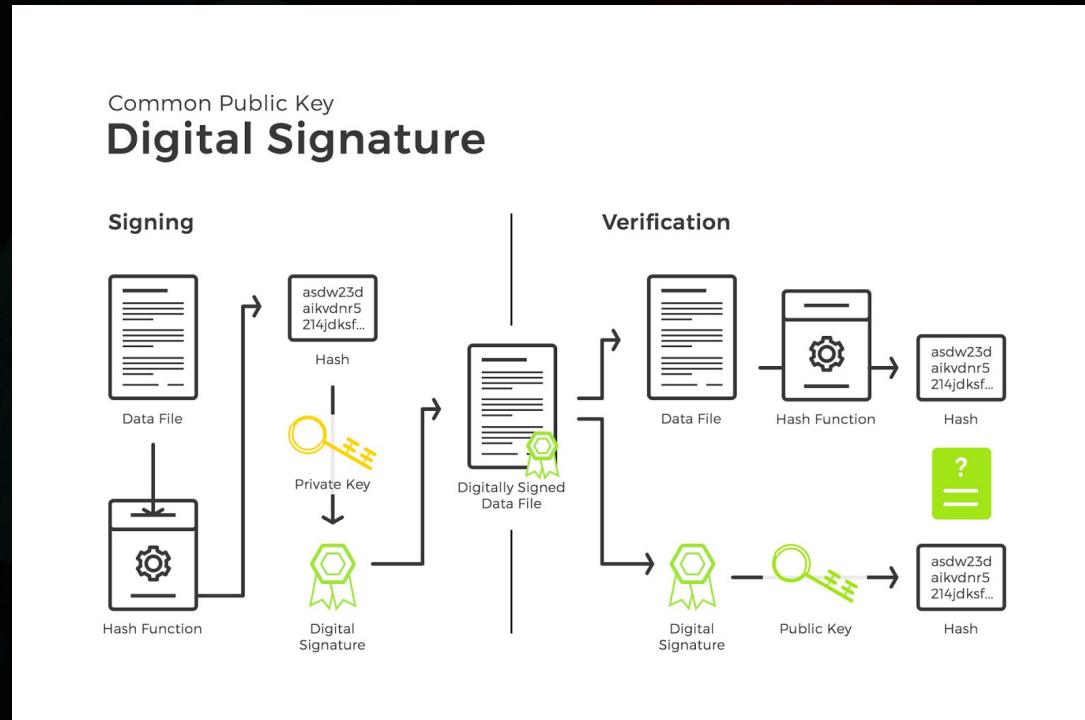
Mecanismos

- Generación de las claves
- Proceso de firma (privado)
- Verificación de la firma (público)

Algoritmos usados con frecuencia

- RSA
- DSA con ECDSA

Firmas digitales



¿Quién asegura a quién pertenece la clave pública?

Diferencias entre las Firmas digitales y HMAC

HMAC y las firmas digitales son mecanismos de autenticación

Cifrado con clave privada de un hash. Calculo de un hash concatenado con una clave.

HMAC: para verificarlo se necesita la clave que se utiliza para calcularlo

MAC no es adecuado como evidencia para un tercero ya que no tiene la clave.

Las firmas digitales pueden ser validadas por una tercera parte

Soportan el no repudio

Soportan la autenticación



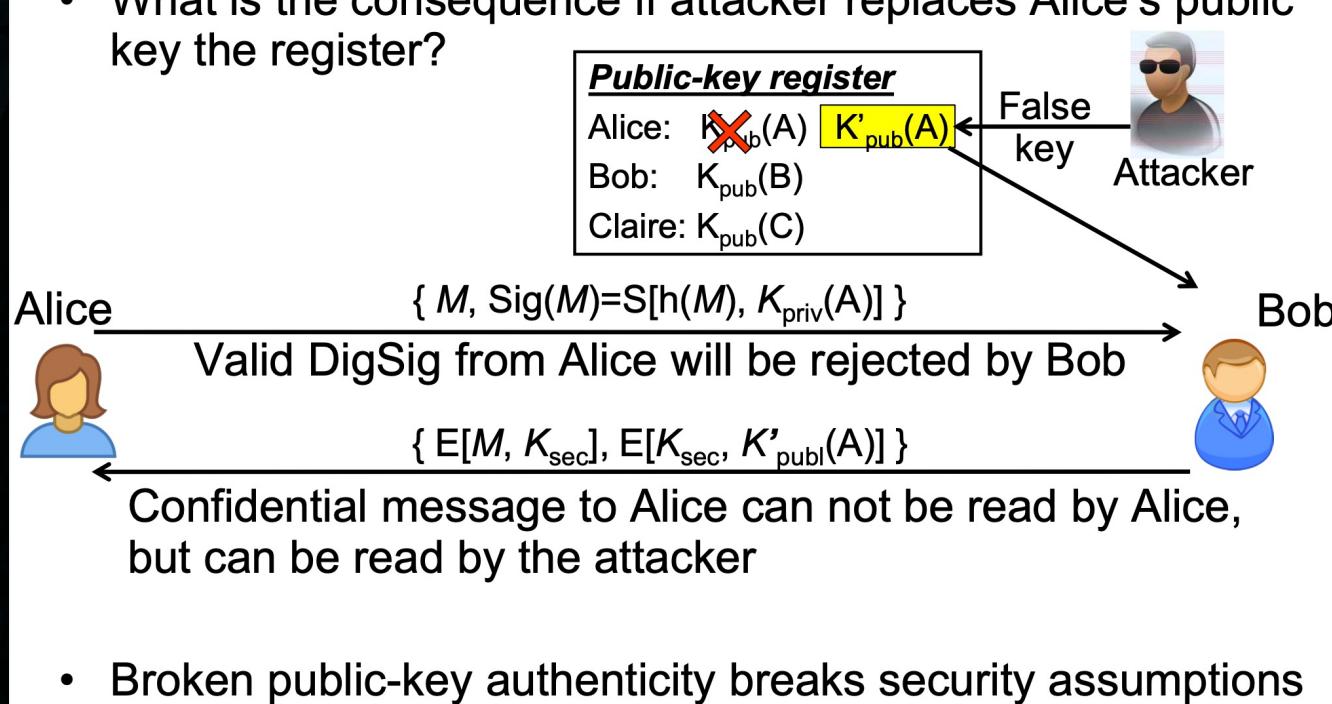
Universidad
Zaragoza

Public Key Infrastructure (PKI)

Problema de las claves públicas no autenticas (fake)

En un mundo ideal, asumimos que las claves están guardadas en un registro público

- What is the consequence if attacker replaces Alice's public key in the register?



Debido al problema de las claves falsas, las claves públicas deben ser firmadas digitalmente antes de su distribución

El propósito principal de una PKI es asegurar la autenticidad de las claves públicas

Una PKI se compone de:

Políticas (definen las reglas para la gestión de los certificados)

Tecnologías (implementan las políticas y generan, almacenan y gestionan las claves)

Procedimientos (relacionados con la gestión de las claves)

Estructura de certificados de clave pública (claves públicas con firmas digitales)

Certificados de clave pública

Un certificado de clave pública es un registro de datos que contiene un nombre distintivo del sujeto (y más cosas) y una clave pública firmado por una Autoridad de Certificación (CA)

Une las propiedades de un sujeto a una clave pública

Las CAs firman las claves públicas

Se necesita una copia de la clave pública de la CA para verificar el certificado

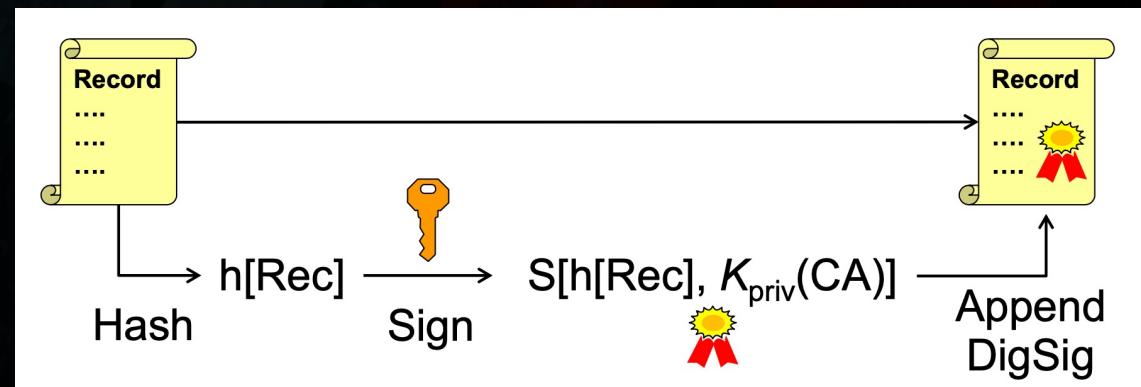
¿Cómo se genera un certificado?

Junta toda la información (nombre, clave pública, etc.) en un registro
El estándar X.509 define los componentes de un registro

Haz un hash del registro

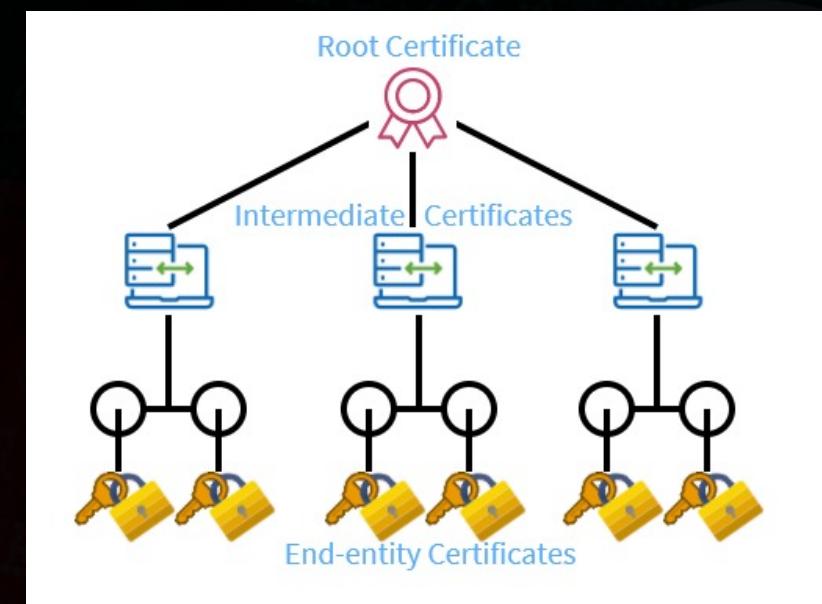
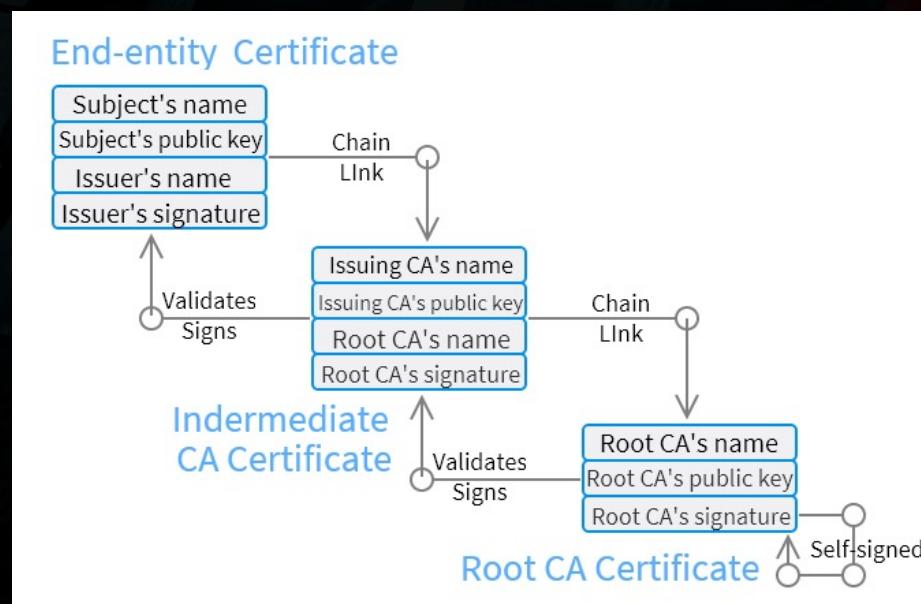
Firma el hash

Añade la firma al registro



Cadena de confianza

Existe una cadena que empieza en las CAs



¿Qué se puede hacer con un certificado?

Depende del tipo. No todos valen para todo.

¿Firmar otros certificados?

Cifrado + Firma de documentos

Certificados de Servidor

Certificados de Cliente

¿Qué formato tienen los certificados?

Existen varios formatos

PKCS12 (RFC 7292 - PKCS #12: Personal Information Exchange Syntax v1.1)

Contenedor cifrado

PEM (RFC 1421 - Privacy Enhancement for Internet Electronic Mail)

Contenedor sin cifrar

Contenedores para certificados (ficheros)

<https://serverfault.com/questions/9708/what-is-a-pem-file-and-how-does-it-differ-from-other-openssl-generated-key-file>

En resumen, hay cuatro formas diferentes de presentar certificados y sus componentes:

PEM: regido por RFC 1421, se utiliza normalmente por software de código libre. Puede tener una variedad de extensiones (.pem, .key, .cer, .cert, más)

PKCS7: un estándar abierto utilizado por Java y compatible con Windows. No contiene material de clave privada.

PKCS12: un estándar privado de Microsoft que luego se definió en la RFC 7292 que brinda seguridad mejorada en comparación con el formato PEM de texto sin formato. Puede contener material de clave privada. Se utiliza normalmente en los sistemas Windows, FNMT y se puede convertir libremente a Formato PEM mediante el uso de openssl.

DER: el formato principal de PEM. Es útil pensar en él como una versión binaria del archivo PEM codificado en base64. No se usa mucho de forma rutinaria fuera de Windows.

Ejemplos de certificados



AC RAIZ FNMT-RCM

Autoridad de certificación raíz
Caduc: martes, 1 de enero de 2030, 1:00:00 (hora estándar de Europa central)
Este certificado está marcado como fiable para esta cuenta

Nombre del sujeto
País o región ES
Empresa FNMT-RCM
Unidad organizativa AC RAIZ FNMT-RCM

Nombre del emisor
País o región ES
Empresa FNMT-RCM
Unidad organizativa AC RAIZ FNMT-RCM

Número de serie 5D 93 8D 30 67 36 C8 06 1D 1A C7 54 84 69 07
Versión 3
Algoritmo de firma SHA-256 con encriptación RSA (1.2.840.113549.1.1.1)
Parámetros Ninguno

No válido antes de miércoles, 29 de octubre de 2008, 16:59:56 (hora estándar de Europa central)
No válido después de martes, 1 de enero de 2030, 1:00:00 (hora estándar de Europa central)

Información de la clave pública
Algoritmo Encriptación RSA (1.2.840.113549.1.1.1)
Parámetros Ninguno
Clave pública 512 bytes: BA 71 80 7A 4C 86 6E 7F ...
Exponente 65537
Tamaño de la clave 4096 bits
Usos de la clave Verificar
Firma 512 bytes: 07 90 4A DF F3 23 4E F0 ...

Extensión Uso de la clave (2.5.29.15)
Crítico Sí
Usos Firmar certificado de clave, Firmar CRL

Extensión Restricciones básicas (2.5.29.19)
Crítico Sí

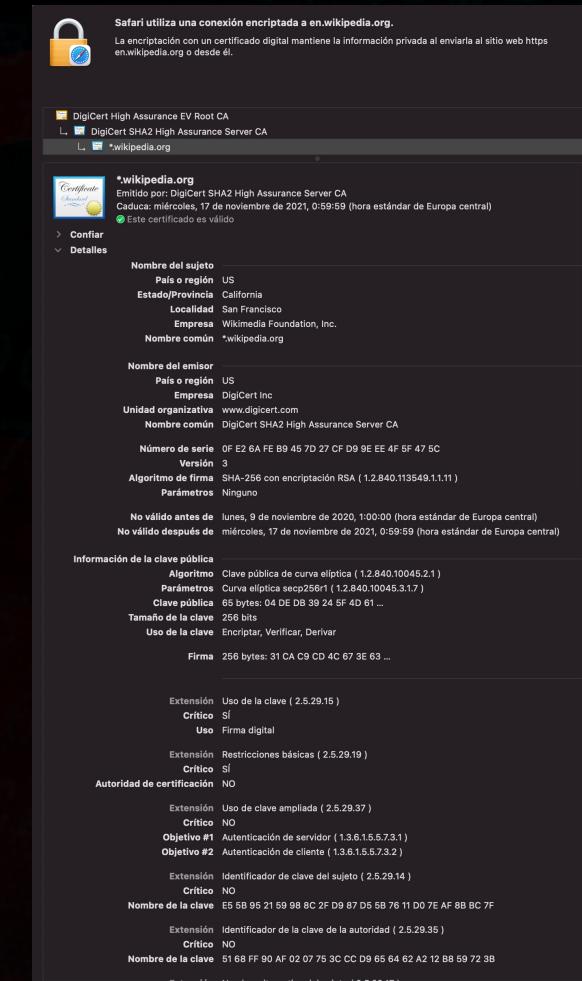
Autoridad de certificación Sí

Extensión Identificador de clave del sujeto (2.5.29.14)
Crítico NO

ID de la política #1 Cualquier identificador de política (2.5.29.32.0)
ID del calificador núm. 1 Instrucción de prácticas de certificación (1.3.6.1.5.7.2.1)

Identificador URI CPS <http://www.cert-fnmt.es/dpcs/>

Huellas digitales
SHA-256 EB C5 57 0C 29 01 8C AD 67 B1 AA 12 7B AF 12 F7
03 B4 61 1E BC 17 B7 DA B5 57 38 94 17 9B 93 FA
SHA-1 EC 50 35 07 B2 15 C4 95 62 19 E2 A8 9A 5B 42 99
2C 4C 2C 20



Safari utiliza una conexión encriptada a en.wikipedia.org.
La encriptación con un certificado digital mantiene la información privada al enviarla al sitio web https://en.wikipedia.org o desde él.

DigiCert High Assurance EV Root CA
└ DigiCert SHA2 High Assurance Server CA
└ *wikipedia.org

*wikipedia.org
Emitted by: DigiCert SHA2 High Assurance Server CA
Caduc: miércoles, 17 de noviembre de 2021, 0:59:59 (hora estándar de Europa central)
Este certificado es válido

Nombre del sujeto
País o región US
Estado/Provincia California
Localidad San Francisco
Empresa Wikimedia Foundation, Inc.
Nombre común *wikipedia.org

Nombre del emisor
País o región US
Empresa DigiCert Inc.
Unidad organizativa www.digicert.com
Nombre común DigiCert SHA2 High Assurance Server CA

Número de serie 0F E2 6A FE B9 45 7D 27 CF D9 9E EE 4F 5F 47 5C
Versión 3
Algoritmo de firma SHA-256 con encriptación RSA (1.2.840.113549.1.1.1)
Parámetros Ninguno

No válido antes de lunes, 9 de noviembre de 2020, 1:00:00 (hora estándar de Europa central)
No válido después de miércoles, 17 de noviembre de 2021, 0:59:59 (hora estándar de Europa central)

Información de la clave pública
Algoritmo Clave pública de curva elíptica (1.2.840.10045.2.1)
Parámetros Curva elíptica secp256r1 (1.2.840.10045.3.1.7)
Clave pública 65 bytes: 04 DE DB 39 24 5F 4D 61 ...
Tamaño de la clave 256 bits
Usos de la clave Encriptar, Verificar, Derivar
Firma 256 bytes: 31 CA C9 CD 4C 67 3E 63 ...

Extensión Uso de la clave (2.5.29.15)
Crítico Sí
Usos Firma digital

Extensión Restricciones básicas (2.5.29.19)
Crítico Sí

Autoridad de certificación NO

Extensión Uso de clave ampliada (2.5.29.37)
Crítico NO
Objetivo #1 Autenticación de servidor (1.3.6.1.5.7.3.1)
Objetivo #2 Autenticación de cliente (1.3.6.1.5.7.3.2)
Extensión Identificador de clave del sujeto (2.5.29.14)
Crítico NO

Nombre de la clave E5 5B 95 21 59 98 8C 2F D9 87 D5 5B 76 11 D0 7E AF 8B BC 7F

Extensión Identificador de la clave de la autoridad (2.5.29.35)
Crítico NO

Nombre de la clave 51 6B FF 90 AF 02 07 75 3C CC D9 65 64 62 A2 12 B8 59 72 3B