

# Tecnología Digital IV: Redes de Computadoras

## Clase 23: Seguridad en Redes de Computadoras - Parte 2

Lucio Santi & Emmanuel Iarussi

Licenciatura en Tecnología Digital  
Universidad Torcuato Di Tella

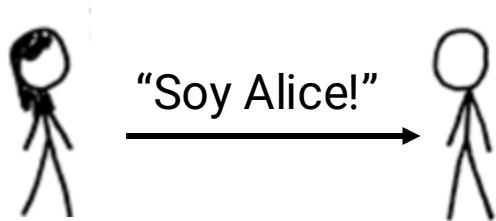
12 de junio de 2025

# Autenticación

# Autenticación

**Objetivo:** Alice quiere demostrar su identidad ante Bob

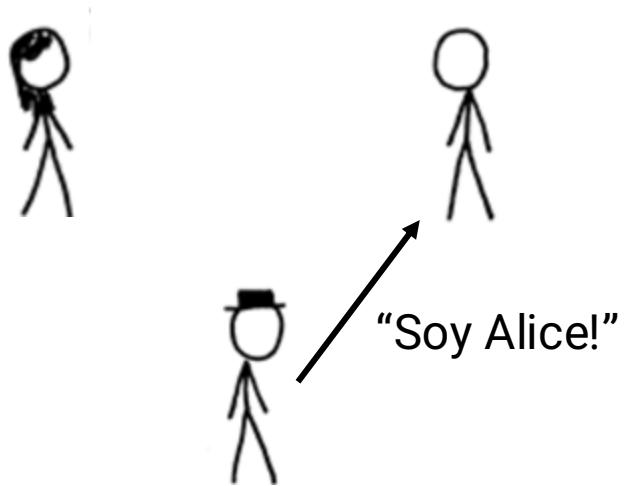
**Protocolo ap1.0:** Alice dice “Soy Alice!”



# Autenticación

**Objetivo:** Alice quiere demostrar su identidad ante Bob

**Protocolo ap1.0:** Alice dice “Soy Alice!”



En la red, Bob **no tiene forma** de ver a Alice:  
Trudy podría declarar que es Alice sin que Bob se dé cuenta

# Autenticación: segundo intento

**Objetivo:** Alice quiere demostrar su identidad ante Bob

**Protocolo ap2.0:** Alice dice “Soy Alice!” en un datagrama IP con su propia dirección

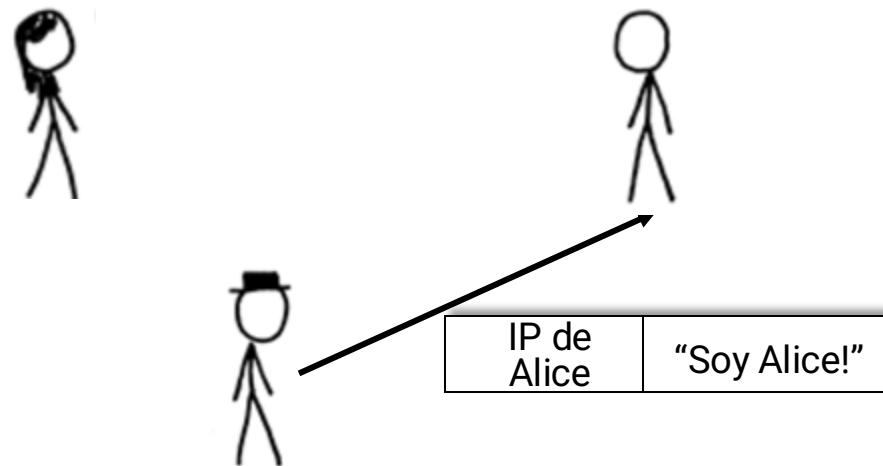


# Autenticación: segundo intento

**Objetivo:** Alice quiere demostrar su identidad ante Bob

**Protocolo ap2.0:** Alice dice “Soy Alice!” en un datagrama IP con su propia dirección

Trudy podría generar un  
datagrama artificial  
*spoofeando* la dirección  
IP de Alice



# Autenticación: tercer intento

**Objetivo:** Alice quiere demostrar su identidad ante Bob

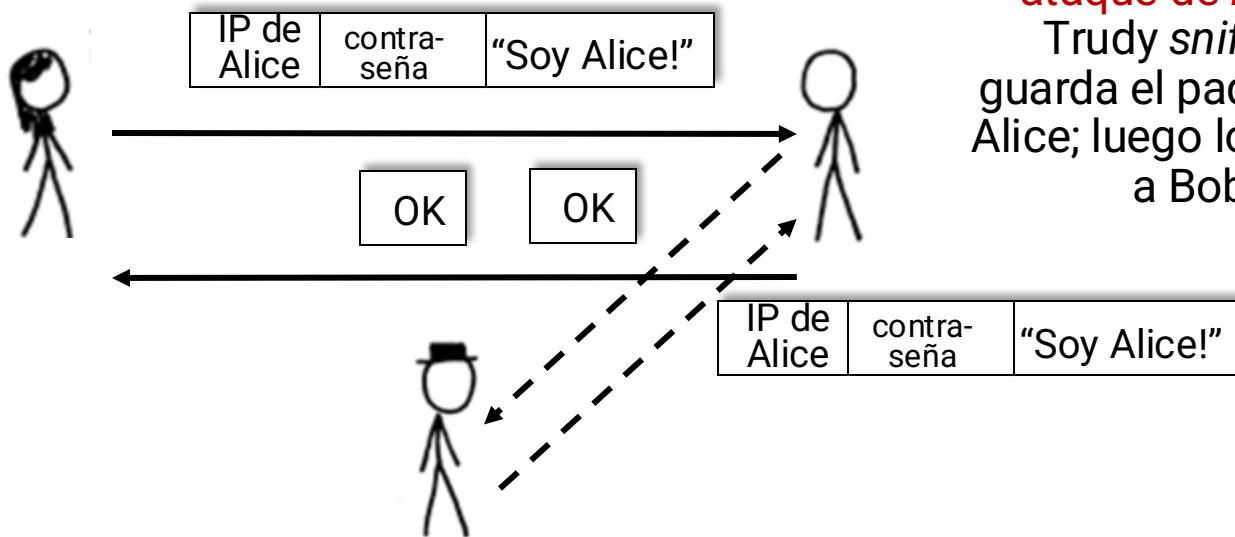
**Protocolo ap3.0:** Alice dice “Soy Alice!” y envía su contraseña para probar que es ella



# Autenticación: tercer intento

Objetivo: Alice quiere demostrar su identidad ante Bob

Protocolo ap3.0: Alice dice “Soy Alice!” y envía su contraseña para probar que es ella



# Autenticación: tercer intento mejorado

Objetivo: Alice quiere demostrar su identidad ante Bob

Protocolo ap3.1: Alice dice “Soy Alice!” y envía su contraseña **encriptada** para probar que es ella

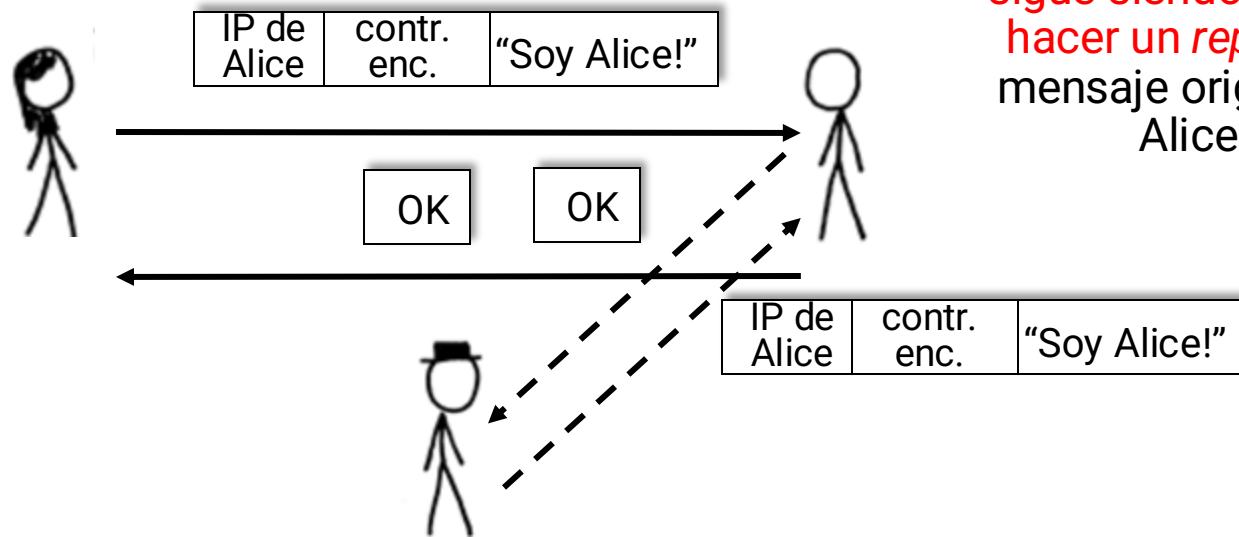


# Autenticación: tercer intento mejorado

**Objetivo:** Alice quiere demostrar su identidad ante Bob

**Protocolo ap3.1:** Alice dice “Soy Alice!” y envía su contraseña **encriptada** para probar que es ella

*Un atacante podría tomar la información cifrada y, sin necesidad de descifrarla, podría continuar enviando un pedido particular al proveedor, de este modo, ordenando productos una y otra vez bajo el mismo nombre e información de compra.*



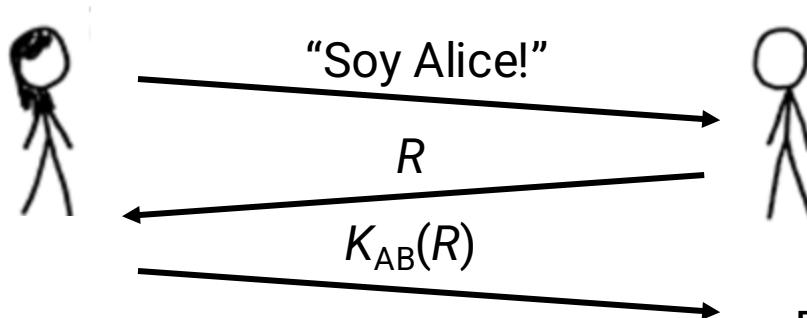
**sigue siendo posible hacer un *replay* del mensaje original de Alice!**

# Autenticación: cuarto intento

Objetivo: evitar ataques de *replay*

Protocolo ap4.0: Bob envía a Alice un *nonce*  $R$ ; Alice debe devolver  $R$  cifrado con una clave compartida

*nonce*: valor numérico utilizado una **única vez** ( $R$ )



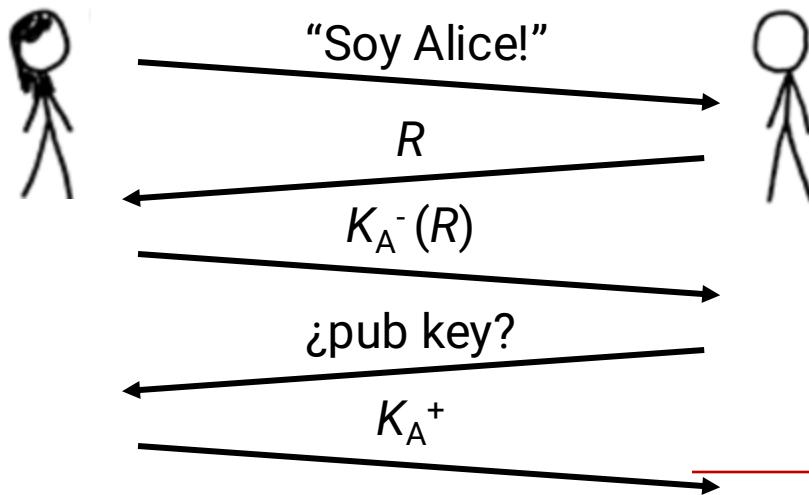
Bob sabe que Alice está activa y que del otro lado está ella (sólo comparte la clave con ella)

# Autenticación: cuarto intento mejorado

El protocolo anterior emplea una clave compartida

¿Podemos lograr autenticación utilizando criptografía de clave pública?

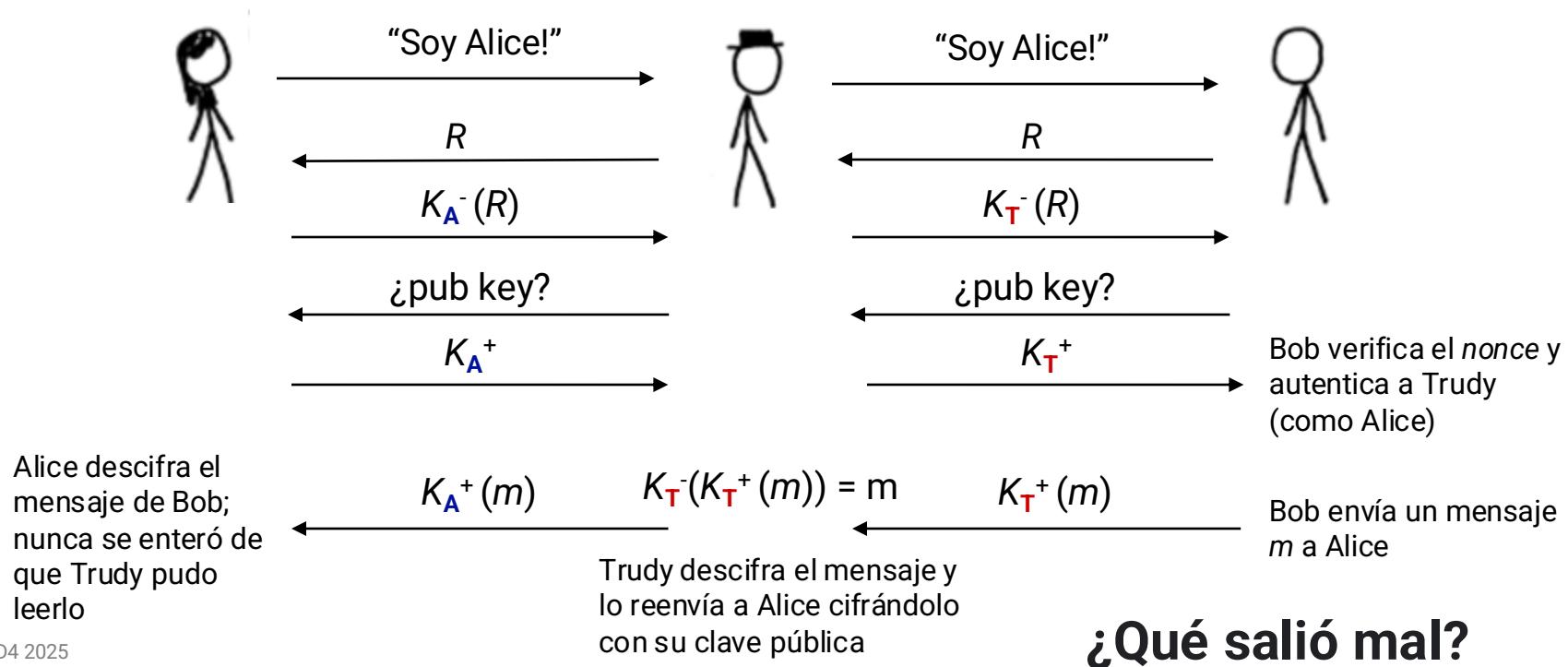
ap5.0: uso de *nonces* y claves públicas



Bob calcula  
 $K_A^+(K_A^-(R)) = R$   
y, sabiendo que sólo  
Alice posee la clave  
privada  $K_A^-$ , concluye  
que se trata de Alice

# Autenticación: fallas en ap5.0

Ataque *man-in-the-middle*: Trudy se hace pasar por Alice hacia Bob y viceversa

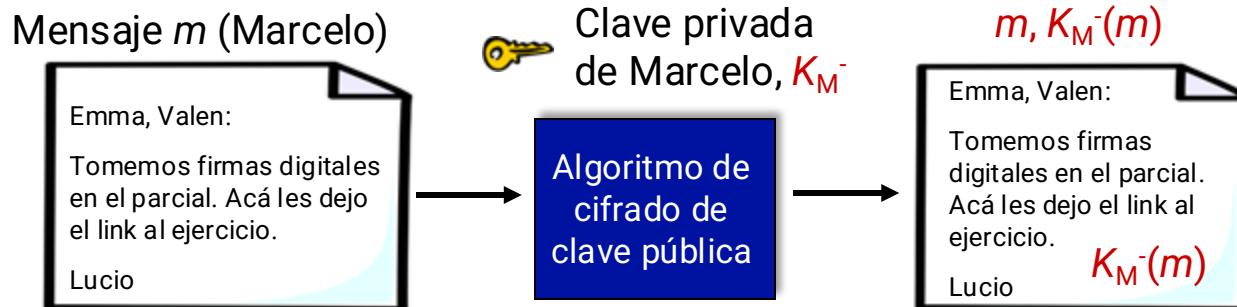


# Integridad

# Firmas digitales

Mecanismo criptográfico análogo a las firmas de puño y letra

- El emisor firma digitalmente un documento de su autoría
- **Verificable y no falsificable:** el receptor puede probar que sólo el emisor del documento produjo la firma que lo acompaña
- Esquema básico: firmar el mensaje  $m$  cifrándolo con la clave privada  $K$



# Firmas digitales

- Supongamos que Emma recibe el mensaje  $m$  con firma  $K_M^-(m)$
- Emma verifica la firma aplicando la clave pública  $K_M^+$  de Marcelo a la firma digital y asegurando que  $K_M^+(K_M^-(m)) = m$
- Si esto vale, quienquiera que haya firmado  $m$  debe haber empleado la clave privada de Marcelo

Luego, Emma verifica que:

- Marcelo firmó  $m$
- Ninguna otra persona firmó  $m$
- Marcelo firmó  $m$  y no otro mensaje  $m'$  (integridad)

No repudio:

- Emma puede demostrar que fue Marcelo quien firmó el mensaje  $m$

# Resúmenes de mensajes (*digests*)

Es **computacionalmente costoso** cifrar mensajes largos con criptografía de clave pública

**Objetivo:** obtener resúmenes de nuestros mensajes que sean de longitud fija y fáciles de computar

- Para calcular estos *digests* de mensajes, utilizamos **funciones de hash criptográficas**:



Algunas propiedades de las funciones de hash criptográficas:

- Produce una salida (*digest*) de longitud fija (e.g. 256 bits)
- No son inyectivas (“muchos a uno”)
- Dado un *digest*  $x$ , es **computacionalmente inviable** encontrar un  $m$  tal que  $H(m) = x$  (i.e. *resistente a la preimagen*)

# Resúmenes de mensajes (*digests*)

Es **computacionalmente costoso** cifrar mensajes largos con criptografía de clave pública.

Mensaje	Hash SHA-256
Hola	185f8db32271fe25f561a6fc938b2e264306ec304eda518007d1764826381969
HOLA	35f1358fb1ec85f96b78198a86cb4872a8394172f42b50f92c68d2c09aa4e9be

mensaje largo  $m$  → función de hash  $H$  →  $H(m)$

Algunas propiedades de las funciones de hash criptográficas:

- Produce una salida (*digest*) de longitud fija (e.g. 256 bits)
- No son inyectivas (“muchos a uno”)
- Dado un *digest*  $x$ , es **computacionalmente inviable** encontrar un  $m$  tal que  $H(m) = x$  (i.e. *resistente a la preimagen*)

# Ejemplo (malo): *checksum* de Internet

El algoritmo de *checksum* de Internet posee algunas de estas propiedades:

- Genera *digests* de longitud fija (16 bits)
- Es “muchos a uno”

...pero, dado un mensaje con cierto *digest*, es fácil encontrar otro mensaje con el mismo *digest* (**no es resistente a colisiones**)

mensaje	en hexa
I O U 1	49 4F 55 31
0 0 . 9	30 30 2E 39
9 B O B	39 42 D2 42

B2 C1 D2 AC

mensajes distintos,  
mismo *checksum*

mensaje	en hexa
I O U 9	49 4F 55 39
0 0 . 1	30 30 2E 31
9 B O B	39 42 D2 42

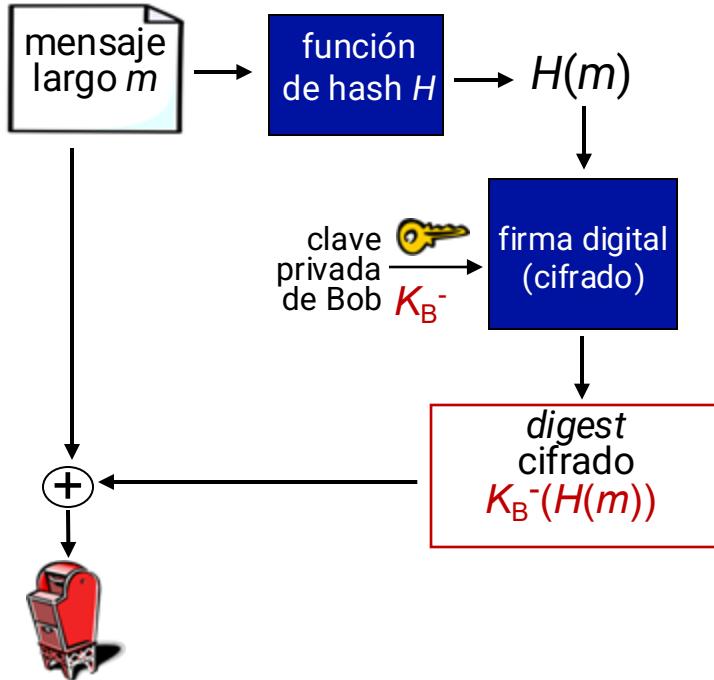
B2 C1 D2 AC

# Algunas funciones de hash populares

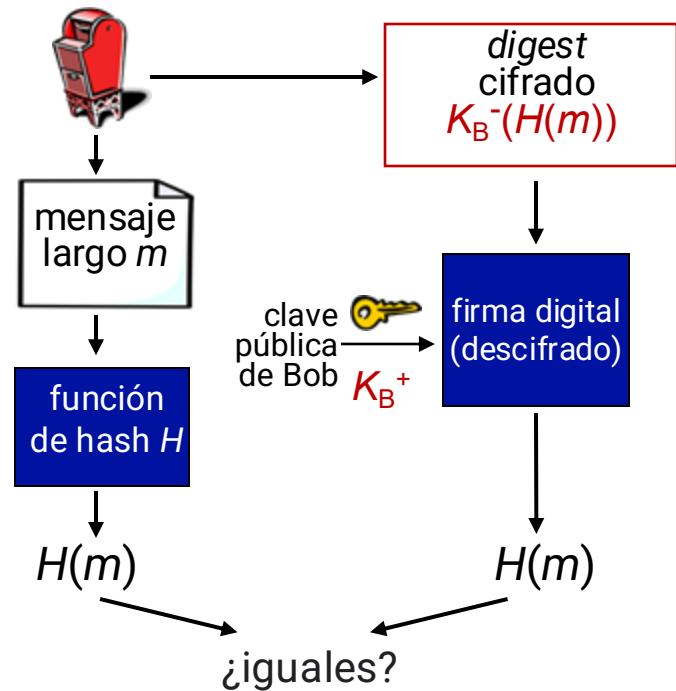
- **MD5 (RFC 1321)**
  - Calcula *digests* de 128 bits en un proceso de cuatro *rounds*
  - Si bien es **vulnerable**, continúa en uso (por ejemplo para verificar integridad en las transferencias de archivos)
  - En 2013 se descubrió un ataque para generar colisiones de MD5 que corre en menos de un segundo en una computadora estándar
- **SHA-1 es otra popular (también vulnerable)**
  - Estandarizada en EEUU por el NIST
  - Produce *digests* de 160 bits
  - SHA-2 y SHA-3 (sus sucesoras) se consideran seguras al día de hoy

# Firma digital: *digest* firmado

Bob envía el mensaje con su firma digital:

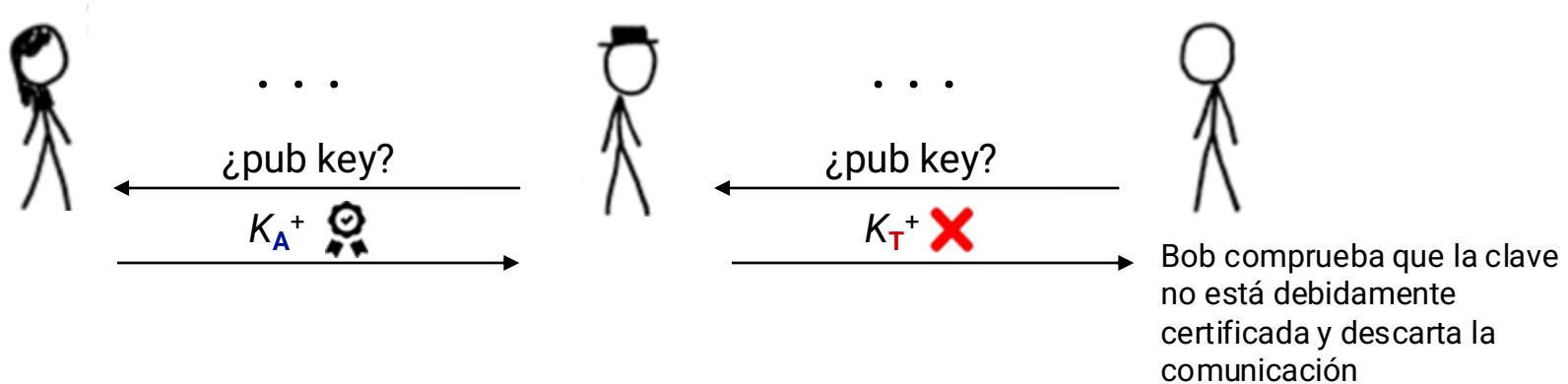


Alice valida la firma y la integridad del mensaje firmado:



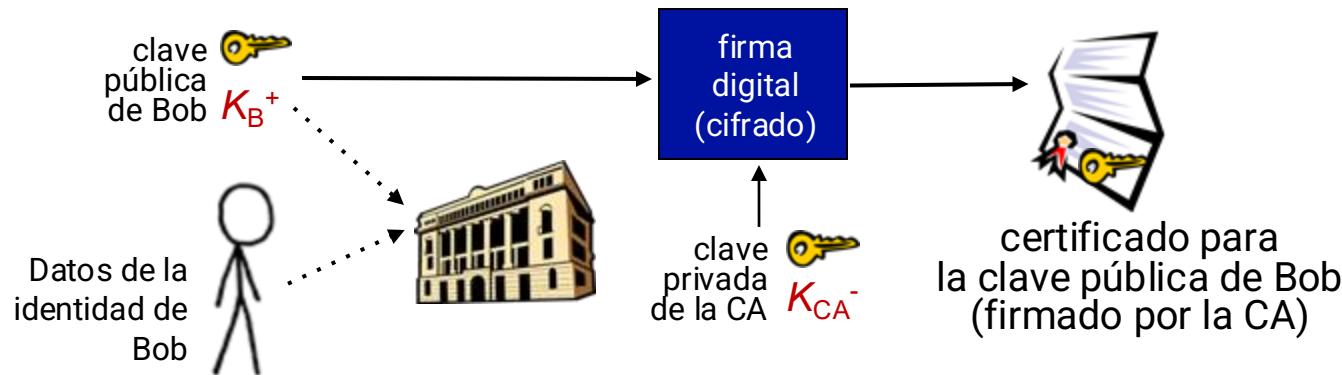
# Retomando el protocolo ap5.0

- Para corregir el protocolo ap5.0, necesitamos **certificar** las claves públicas
- Si Bob pudiera comprobar que la clave pública de Alice es auténtica, el ataque *man-in-the-middle* de Trudy no tendría efecto



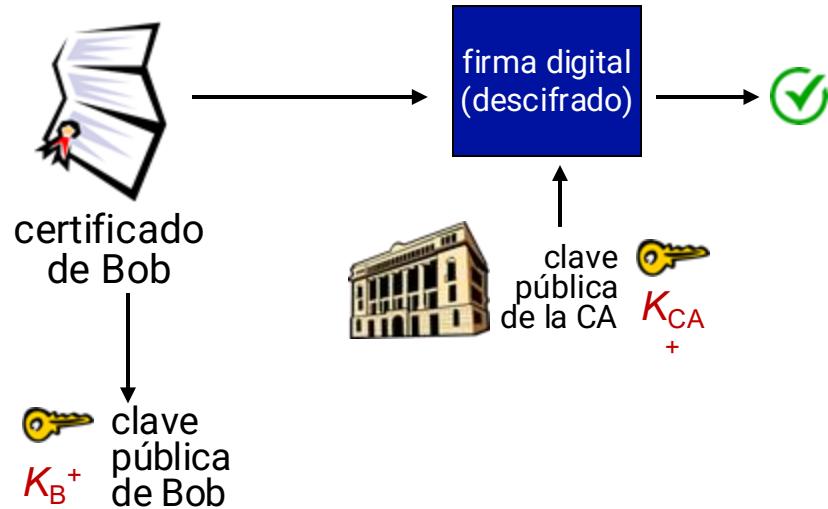
# Autoridades de Certificación (CAs)

- Las **autoridades de certificación (CAs)** vinculan una clave pública con una *entidad* (persona física, sitio web, router, etc.)
- Cuando una entidad  $E$  desea certificar su clave pública, la CA debe verificar la identidad del solicitante
  - La CA genera un **certificado** vinculando la identidad de  $E$  con su clave pública
  - Dicho certificado está firmado digitalmente por la CA



# Autoridades de Certificación (CAs)

- Cuando Alice quiere utilizar la clave pública de Bob,
  - Obtiene el certificado de Bob
  - Utiliza la clave pública de la CA para validar el certificado
  - Extrae del certificado la clave pública de Bob



# Certificados en Internet

- En los protocolos de Internet, los certificados de clave pública siguen el estándar **X.509** de la ITU (*International Telecommunication Union*)
  - Describe el formato de los certificados y el mecanismo de autenticación para utilizarlos
  - Incluye datos de identidad de la entidad dueña del certificado, el período de validez del mismo, los algoritmos criptográficos utilizados y la firma digital de la CA, entre otros campos
- Dos tipos de certificados: *certificado de CA* y *certificado de entidad final*
  - Los certificados de CA se pueden usar para firmar otros certificados
  - Al certificado al comienzo de la cadena de certificación se lo conoce como **certificado root**
- Los navegadores suelen traer una lista de certificados *root* de las CAs más importantes (e.g. IdenTrust, DigiCert, Let's Encrypt, etc.)
- Los servidores web envían la cadena completa de certificados intermedios para que los navegadores puedan realizar las validaciones correspondientes

# Demo!

- Usemos [openssl](#) para generar un certificado X.509 autofirmado:
  - ¿Qué información debemos suministrar?
- Inspeccionemos los certificados de nuestro navegador:
  - ¿Cuáles son las CAs que emitieron los certificados *root* de confianza?
  - ¿Cómo es la cadena de certificados proveniente de <https://www.utdt.edu>?
  - ¿Qué pasa si agregamos nuestro certificado anterior a la lista de certificados *root*?