SEGURIDAD DE LA INFORMACIÓN

TEMA 3 - PARTE 1

ESQUEMAS, PROTOCOLOS Y MECANISMOS DE SOPORTE

(A LA SEGURIDAD DE APLICACIONES Y DE REDES)

Índice del tema

- **Gestión de las "claves"**
  - Protocolos de distribución de claves simétricas
  - Mecanismos e infraestructuras de administración de claves públicas
    - El caso del DNI-e
  - Mecanismo de Single Sign-On para Autenticación

- **Mecanismos de Control de Acceso**
  - DAC, MAC, RBAC, ABAC
  - Otros

- **Protocolos criptográficos avanzados**
  - Protocolos de división y compartición de secretos
  - Protocolos de compromiso de bit (bit-commitment)
  - Protocolos de lanzamiento de moneda
  - Protocolo de póker mental

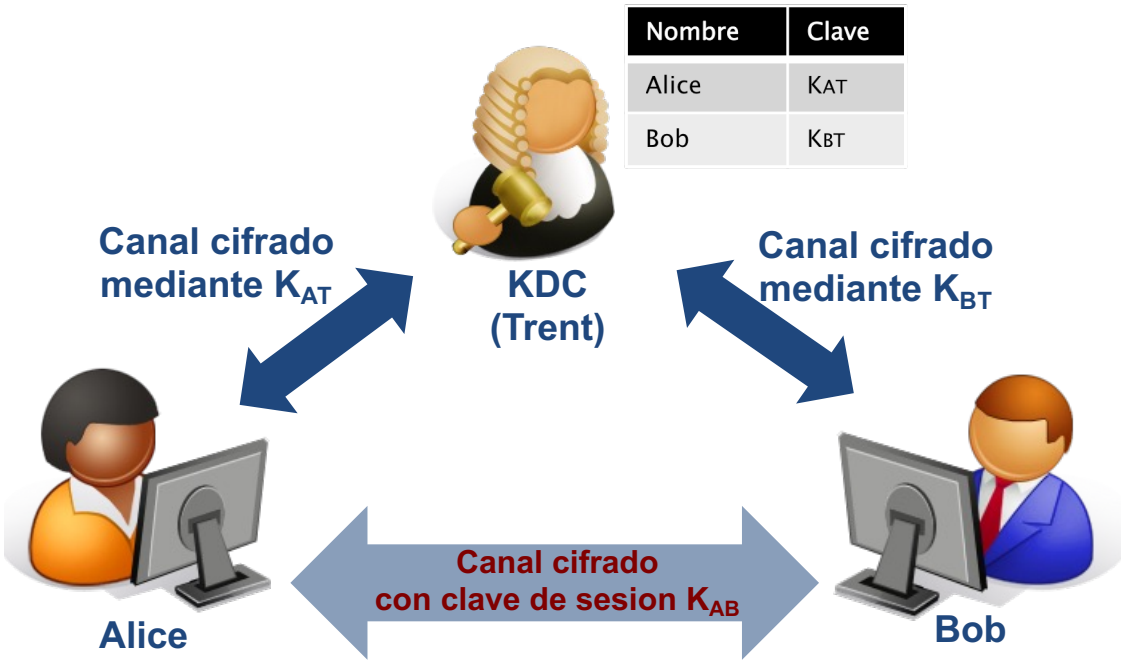
- **Referencias bibliográficas**

Gestión de las Claves

Protocolos de distribución de claves simétricas

- Hay escenarios donde la utilización de la criptografía de clave pública (o asimétrica) para el intercambio de una clave de sesión K_{AB} puede ser NO conveniente
  - p. ej. redes LAN grandes sin conexión a la red WAN. En este caso, *Alice* y *Bob* van a seguir necesitando de alguna solución que les permitan, aún estando geográficamente (moderadamente) lejanos, decidir esa clave de sesión K_{AB}
- En estos casos, la solución pasa por algún protocolo de **distribución centralizada de claves** para los usuarios del sistema
  - consiste en hacer uso de una tercera parte confiable (TTP – Trusted Third Party), que en este caso se denomina **Centro de Distribución de Claves** (o **KDC** – *Key Distribution Center*)
- Existen diferentes protocolos que proporcionan una solución para ese escenario:
  - Yahalom, Needam-Schroeder, Otway-Rees, Kerberos, …

- En el modus operandi general de este tipo de protocolos, cada usuario del sistema comparte, de inicio, una **clave secreta** con el KDC
  - mediante algún proceso de registro o inscripción del usuario ante el KDC



KDC: modelos y protocolos

Canal cifrado mediante K_{AT}, K_{BT}

Canal cifrado mediante K_{AB}

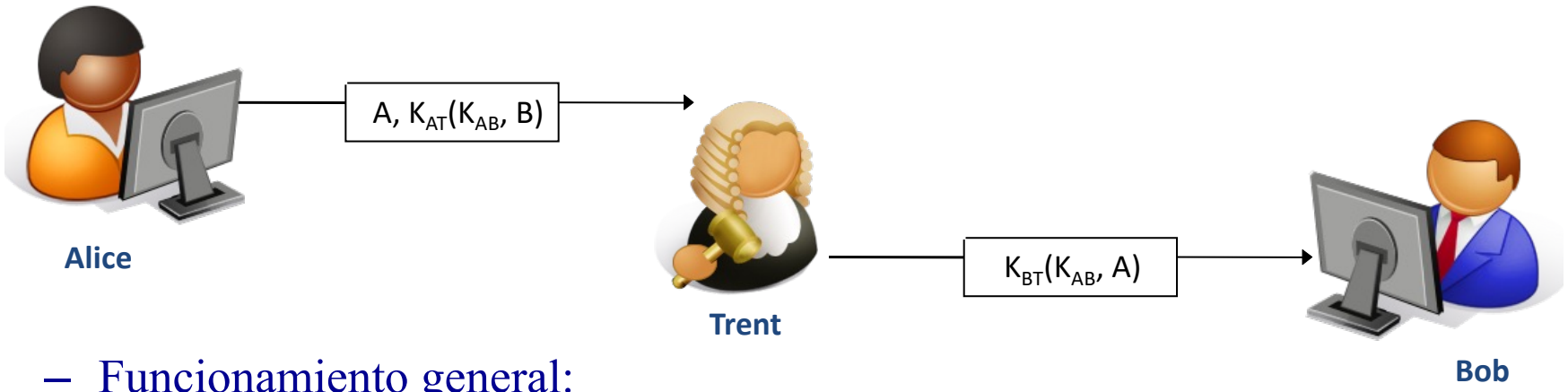
- El uso de un KDC se basa en el uso de claves jerárquicas, de manera que se requieren al menos dos niveles de claves

- La mayoría de las técnicas de distribución de claves se adaptan a situaciones, escenarios y aplicaciones específicas
  - de manera que son diversos los esquemas que se integran a entornos locales donde todos los usuarios tienen acceso a un **servidor común de confianza**

- Hay muchos modelos de distribución de claves:
  - **Simples**
  - **Genéricos**, y dentro de los genéricos nos podemos encontrar:
    - Los modelos **PULL** o modelos **PUSH**, o sus combinaciones

KDC: modelos y protocolos – Modelo Simple

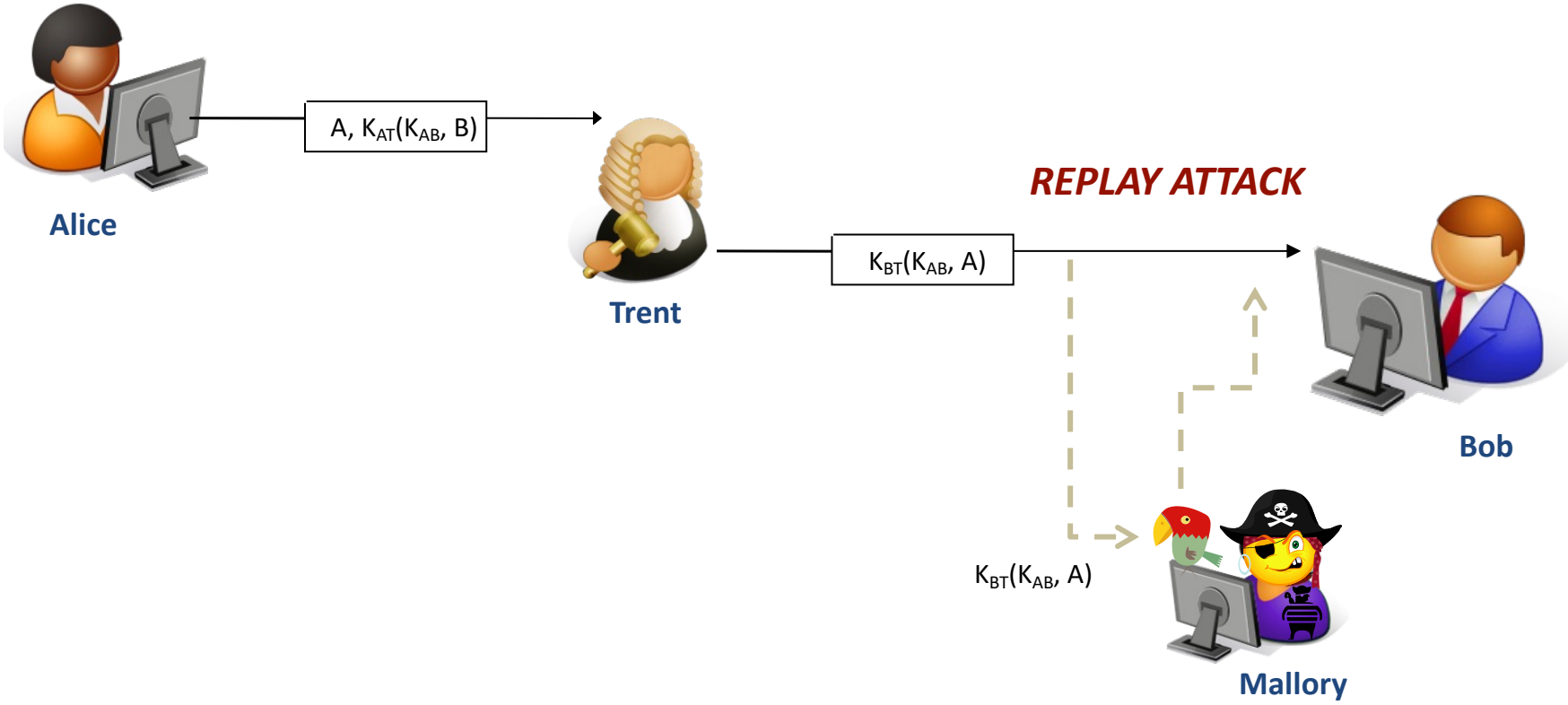
- El protocolo siguiente es un ej. de modelo simple para la distribución de claves:



  - Funcionamiento general:
    - **Paso 1**: A genera una clave de sesión K_{AB} y se la envía al KDC
      - el mensaje incluye la identidad de A, la identidad de B y la clave de sesión cifrada con el K_{AT}
    - **Paso 2**: el KDC verifica la identidad de A y reenvía la K_{AB} a B cifrado con K_{BT}
    - **Paso 3**: B verifica la identidad de KDC por la K_{BT} y obtiene la clave de sesión
- Como se puede observar existe **validación de identidad**:
  - Las claves con el KDC son secretas, por lo que nadie más habría sido capaz de cifrar la clave secreta K_{AB}, además existe autenticación de cada parte involucrada

KDC: modelos y protocolos – Modelo Simple

- Sin embargo, existe un **fallo de seguridad**:



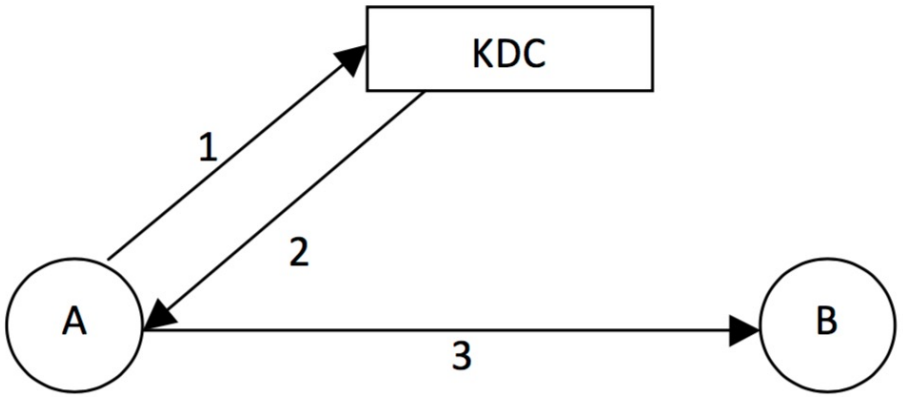
Si Mallory intercepta el canal y captura todos los mensajes de KDC a B, entonces es posible que Mallory cause un **ataque de repetición (ataque *replay*)**, y, por consiguiente, un ataque de Denegación de Servicio (DoS) sin necesidad de que éste derive K_{AB} o K_{BT}

KDC: modelos y protocolos – Modelo Simple *Freshnesses*

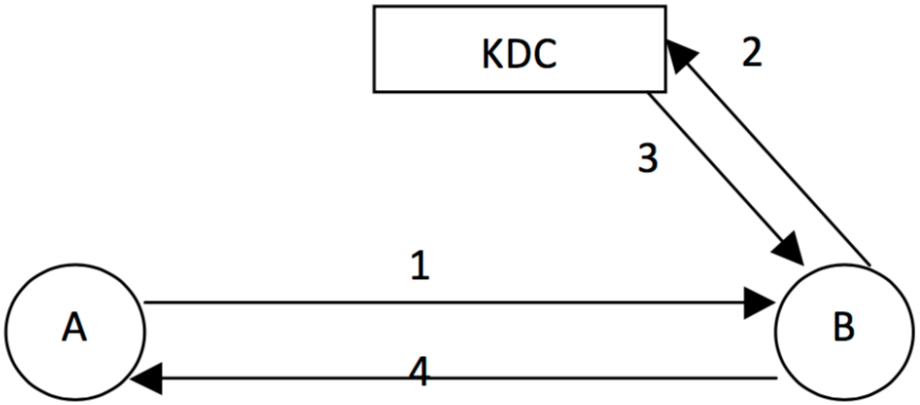
- Para resolver el problema anterior, se pueden hacer uso de alguno de los mecanismos existentes:
  - **Marca de tiempo**: incluir en cada mensaje una marca de tiempo (un sello de tiempo) de forma que pueda descartar mensajes obsoletos
    - Problema: los relojes nunca están perfectamente sincronizados en toda una red
  - **Nonce / núnico**: incluir un número aleatorio único para cada mensaje enviado, de forma que cada parte de la comunicación debe siempre recordar todos los núnicos enviados o recibidos, y rechazar cualquier mensaje que contenga un núnico previamente usado
    - Problema: si una de las partes pierde la lista de nonce / núnicos, es suceptible a ataques *replays*
  - **Combinación de ambas** estrategias para limitar el tiempo que pueden recordarse los núnicos, pero el protocolo se volverá más complicado

KDC: modelos y protocolos – Modelos Genéricos

- Modelo **PULL** para la distribución de claves:

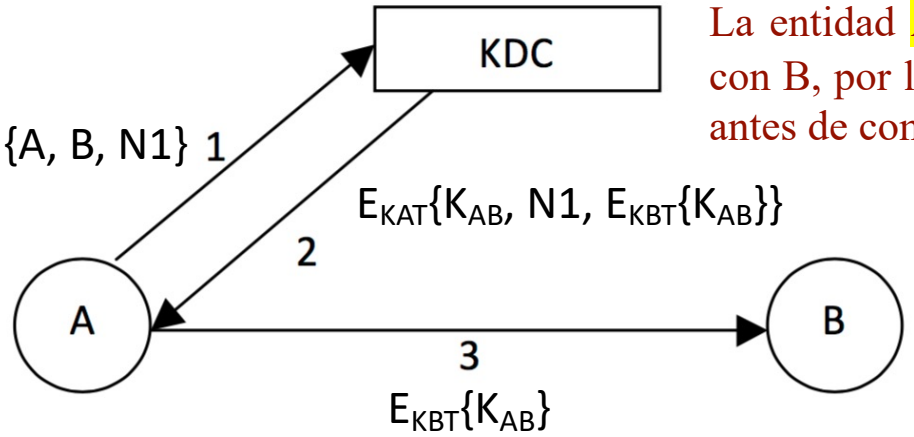


- Modelo **PUSH** para la distribución de claves:



KDC: modelos y protocolos

- Modelo **PULL** para la distribución de claves:

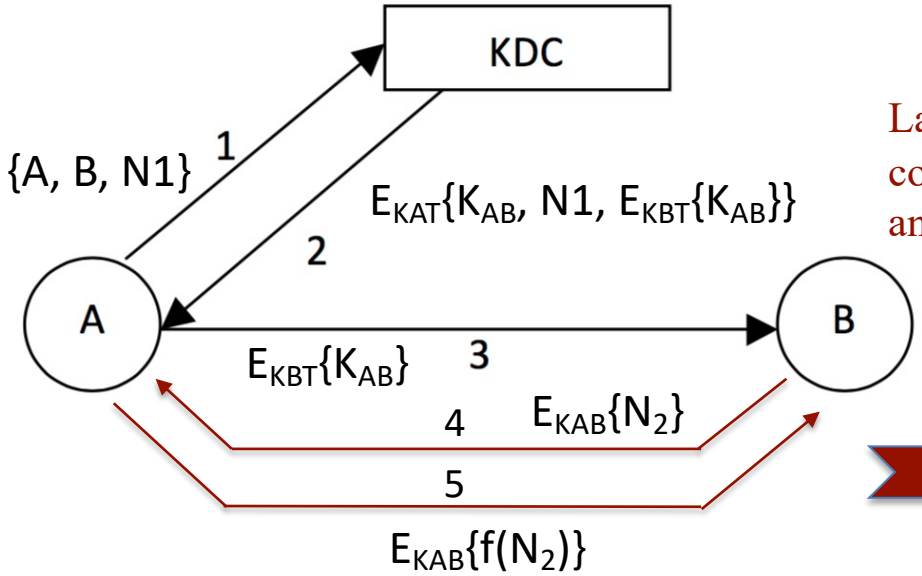


La entidad A desea tener comunicación segura con B, por lo que contacta con el KDC primero antes de comunicarse con B

  - Funcionamiento general:
    - **Paso 1**: A solicita una clave de sesión K_{AB} al KDC
      - el mensaje incluye la identidad de A, la identidad de B y un valor **N1 (sello de tiempo, valor aleatorio)**
    - **Paso 2**: El KDC le contesta a A con un mensaje cifrado mediante la clave maestra K_{AT}, de manera que solamente A puede leer dicho mensaje y con ello, sabe, además, que el KDC es el único que pudo haberlo generado
      - el mensaje contiene la clave K_{AB}, N1, y un mensaje cifrado para B con el K_{AB}
    - **Paso 3**: A obtiene la información recibida y reenvía el mensaje a B para que pueda obtener el K_{AB} también

KDC: modelos y protocolos

- Modelo **PULL** para la distribución de claves:



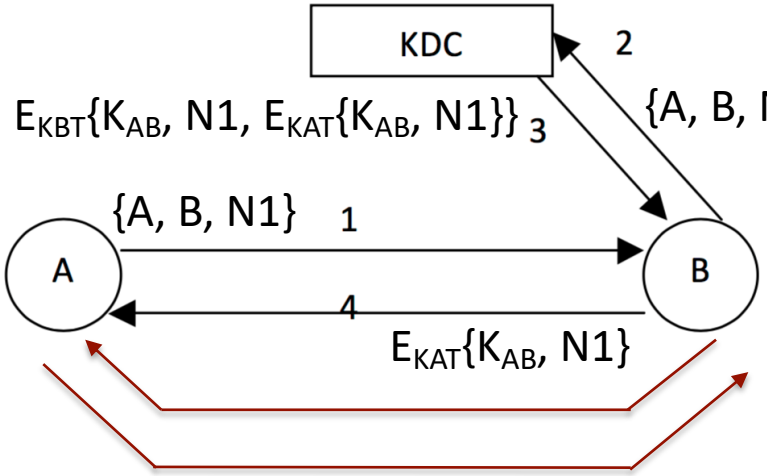
La entidad A desea tener comunicación segura con B, por lo que contacta con el KDC primero antes de comunicarse con B

desafío-respuesta
"challenge-response"

  - Funcionamiento general:
    - **Paso 4**: B utiliza la K_{AB} para cifrar un valor único N2 y se lo envía a A
    - **Paso 5**: A recibe el valor N2, le aplica una transformación f(N2), lo cifra con K_{AB} y lo transmite a B

KDC: modelos y protocolos

- Modelo **PUSH** para la distribución de claves:



La entidad A contacta primero con la entidad B a fin de que ésta última sea la encargada de solicitar al KDC la clave correspondiente

  - Funcionamiento general:
    - **Paso 1**: A solicita conexión segura con B a B
      - le manda, como mínimo, su identidad, la identidad de B y un nonce
    - **Paso 2**: B reenvía dicha solicitud a KDC para que éste genere la K_{AB}
    - **Paso 3**: KDC verifica las identidades y el *freshnesses* de los mensajes, genera la K_{AB}, y dicha información se reenvía a B cifrada con la correspondiente K_{xT}
    - **Paso 4**: B reenvía dicha solicitud a A para que obtenga K_{AB}
    - **Paso 5 (opcional)**: A establece un desafío y respuesta

- # Protocolo de **Needham-Schroeder**

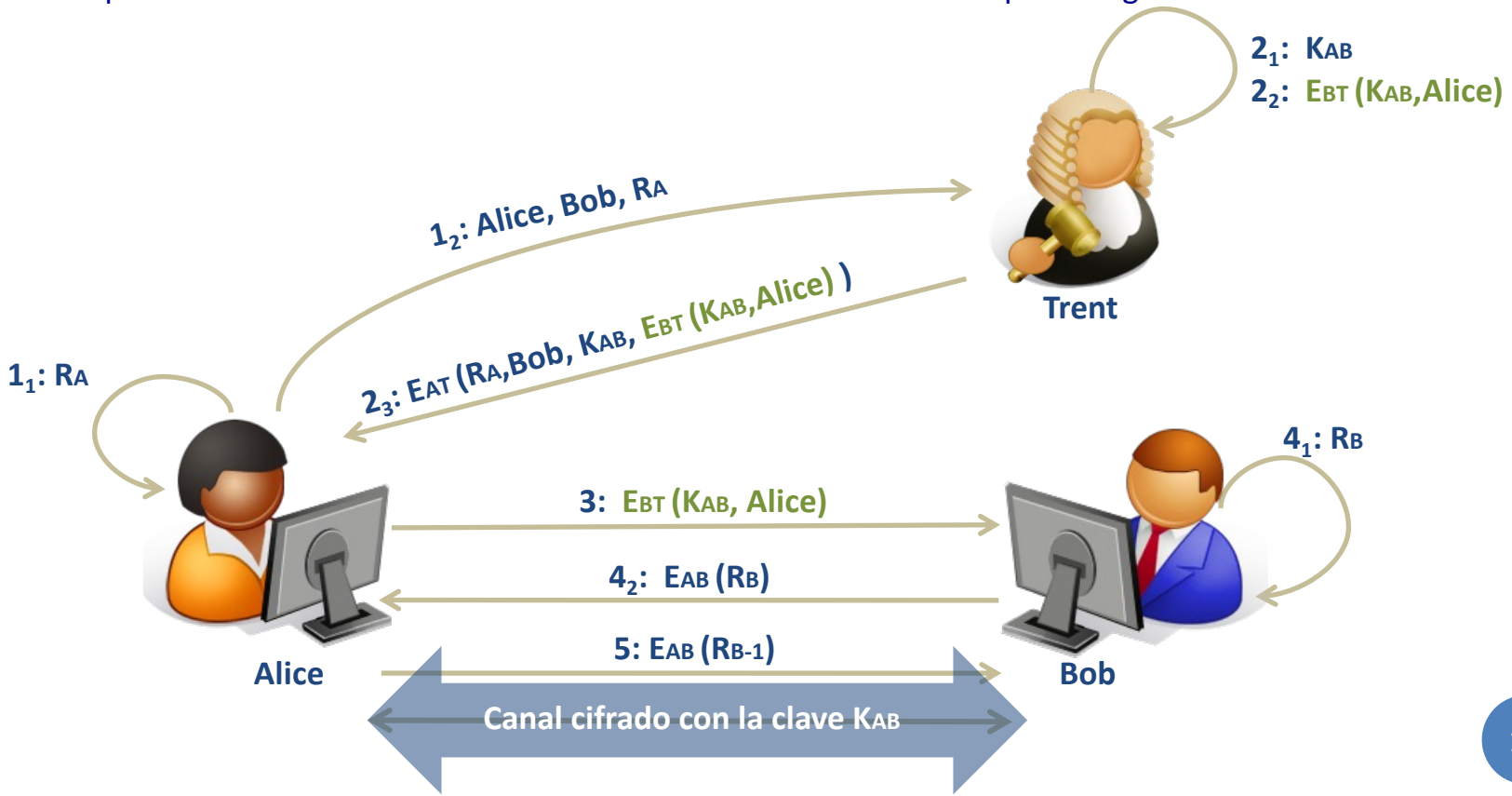
1: *Alice* genera el valor aleatorio (núnico) R_A <1_1>, y se lo envía a *Trent* <1_2>.

2: *Trent* genera la clave de sesión K_{AB} <2_1>, y se la envía a *Alice*, junto a un mensaje para *Bob* <2_3>.

3: *Alice* envía a *Bob* el mensaje que ella ha recibido de *Trent*.

4: *Bob* genera valor aleatorio R_B y se lo envía a *Alice* usando la clave K_{AB} (proceso de "**challenge-response**").

5: Alice responde a Bob cifrando el resultado de restar 1 al valor aleatorio que éste generó.



Needham-Schroeder

- Diseño formalizado:

1. $A \rightarrow S : A, B, N_A$ → **freshness** - pero sin cifrar su valor ☹
2. $S \rightarrow A : \{N_A, B, K_{A,B}, \{K_{A,B}, A\}_{K_{B,S}}\}_{K_{A,S}}$
3. $A \rightarrow B : \{K_{A,B}, A\}_{K_{B,S}}$
4. $B \rightarrow A : \{N_B\}_{K_{A,B}}$
5. $A \rightarrow B : \{N_B - 1\}_{K_{A,B}}$

→ **desafío-respuesta "challenge-response"**

- pero se aplica una función muy obvia *f(n) = n - 1* ☹

N_A: nonce/valor aleatorio
S: KDC
A: Alice
B: Bob
$K_{A,B}$: clave secreta compartida

- Este protocolo tiene más **fallos de seguridad**, que fueron descubiertos años después de su funcionamiento

10 minutos – tic, tac, tic, tac …

- Este protocolo tiene más **fallos de seguridad**, que fueron descubiertos años después de su funcionamiento

  - **Ataque 1**: Mallory, el atacante, puede suplantar la identidad de Alice si éste consigue derivar la clave K_{AT}

    … obvio

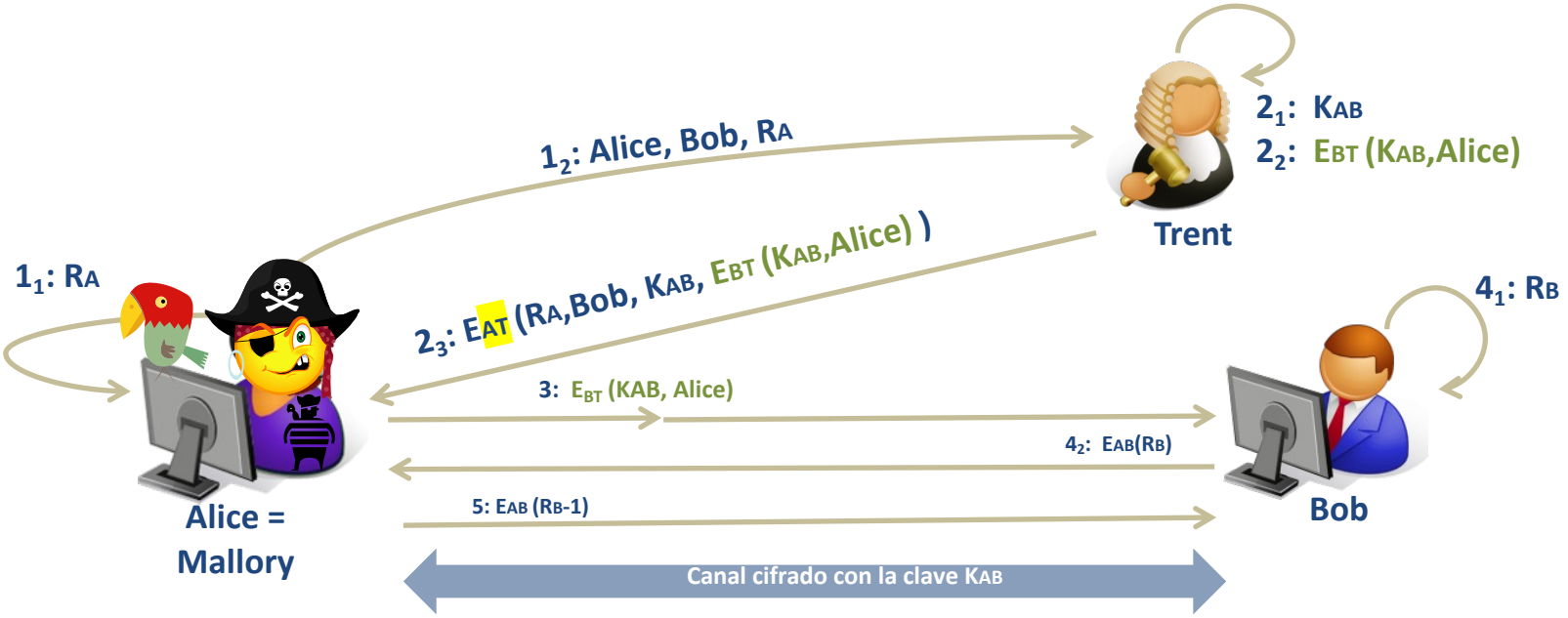
  - **Ataque 2**: Mallory puede suplantar la identidad de Bob si éste consigue derivar la clave K_{BT}

    … obvio

  - **Ataque 3**: Mallory puede producir un ataque de DoS debido a un ataque de repetición, especialmente en las últimas fases del protocolo

    … ¿ dónde ?

- # Ataque 1

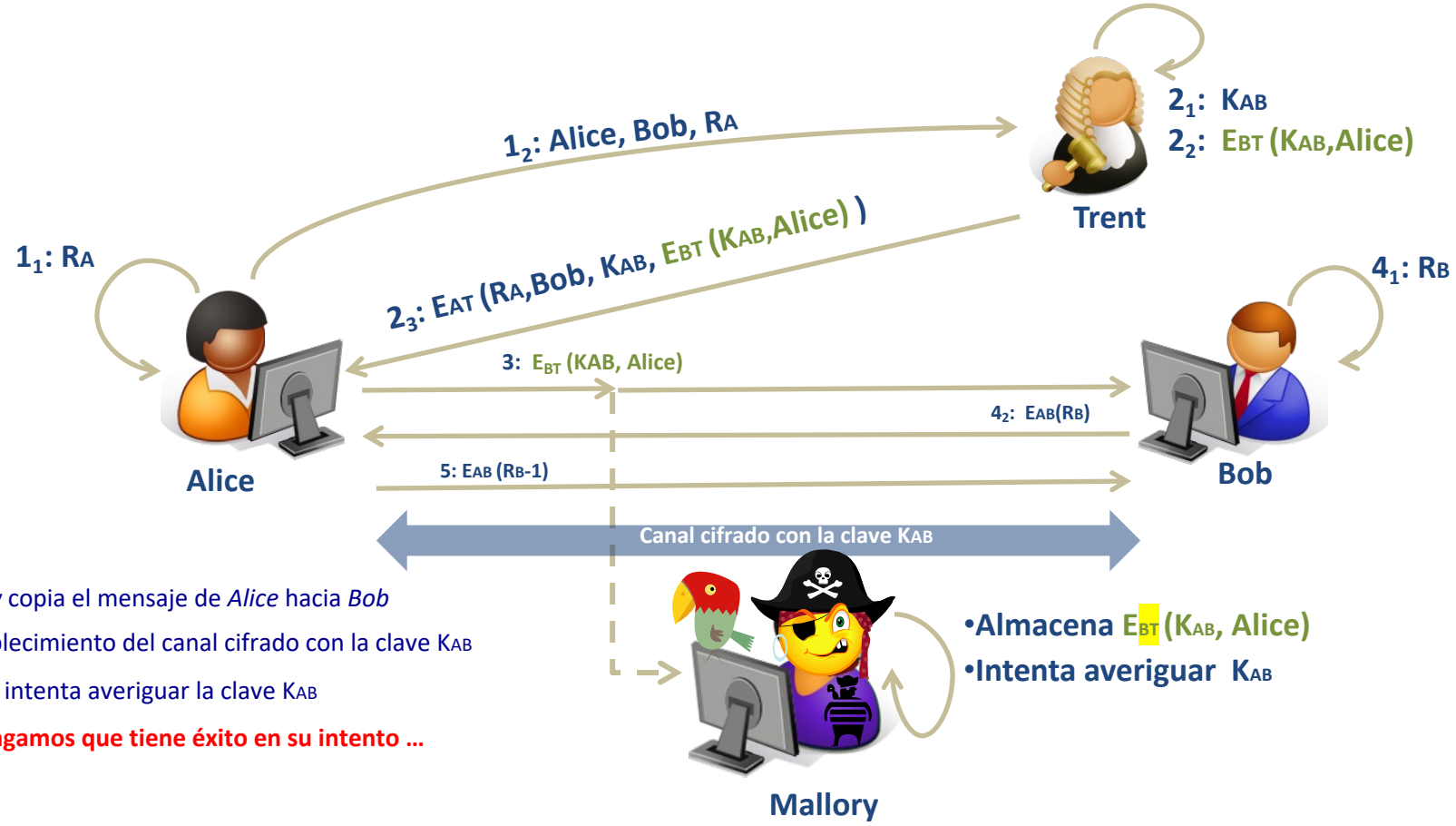


0: *Mallory* copia cualquier mensaje de *Alice* hacia *Trent* en el pasado y deriva la clave K_{AT} . A partir de aquí, todos los mensajes quedan comprometidos

supongamos que tiene éxito en su intento …

- **Almacena E_{BT} (K_{AB}, Alice)**
- **Intenta averiguar K_{AB}**

- # Ataque 2



3: *Mallory* copia el mensaje de *Alice* hacia *Bob*

4, 5: Establecimiento del canal cifrado con la clave K_{AB}

6: *Mallory* intenta averiguar la clave K_{AB}

supongamos que tiene éxito en su intento …

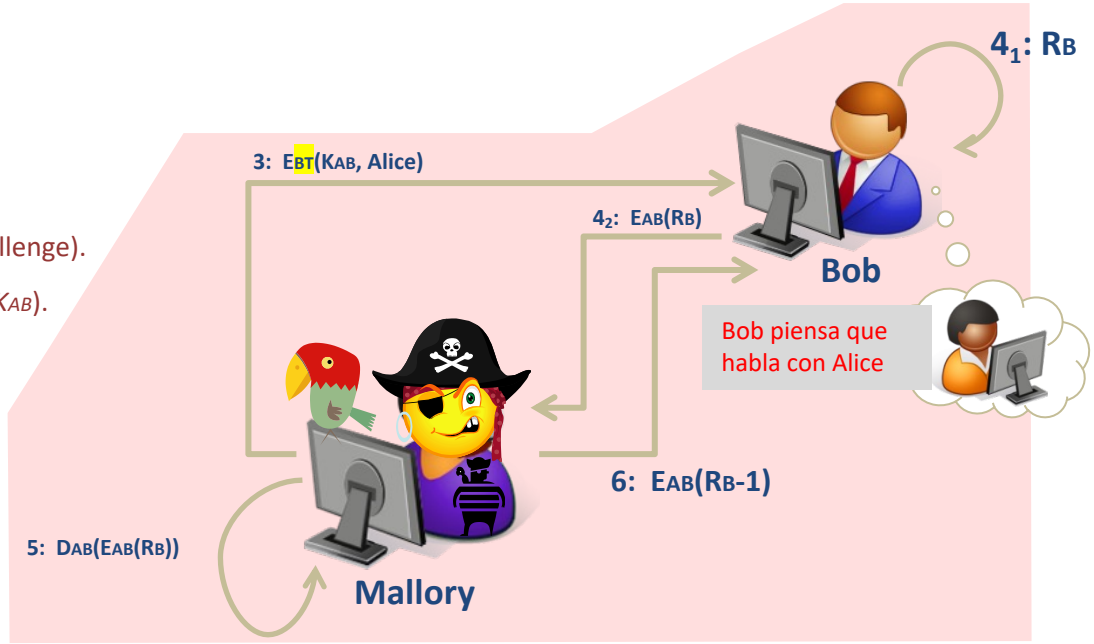
- continuación…

3: *Mallory* envía el mensaje E_{BT} (K_{AB}, Alice) a *Bob*.

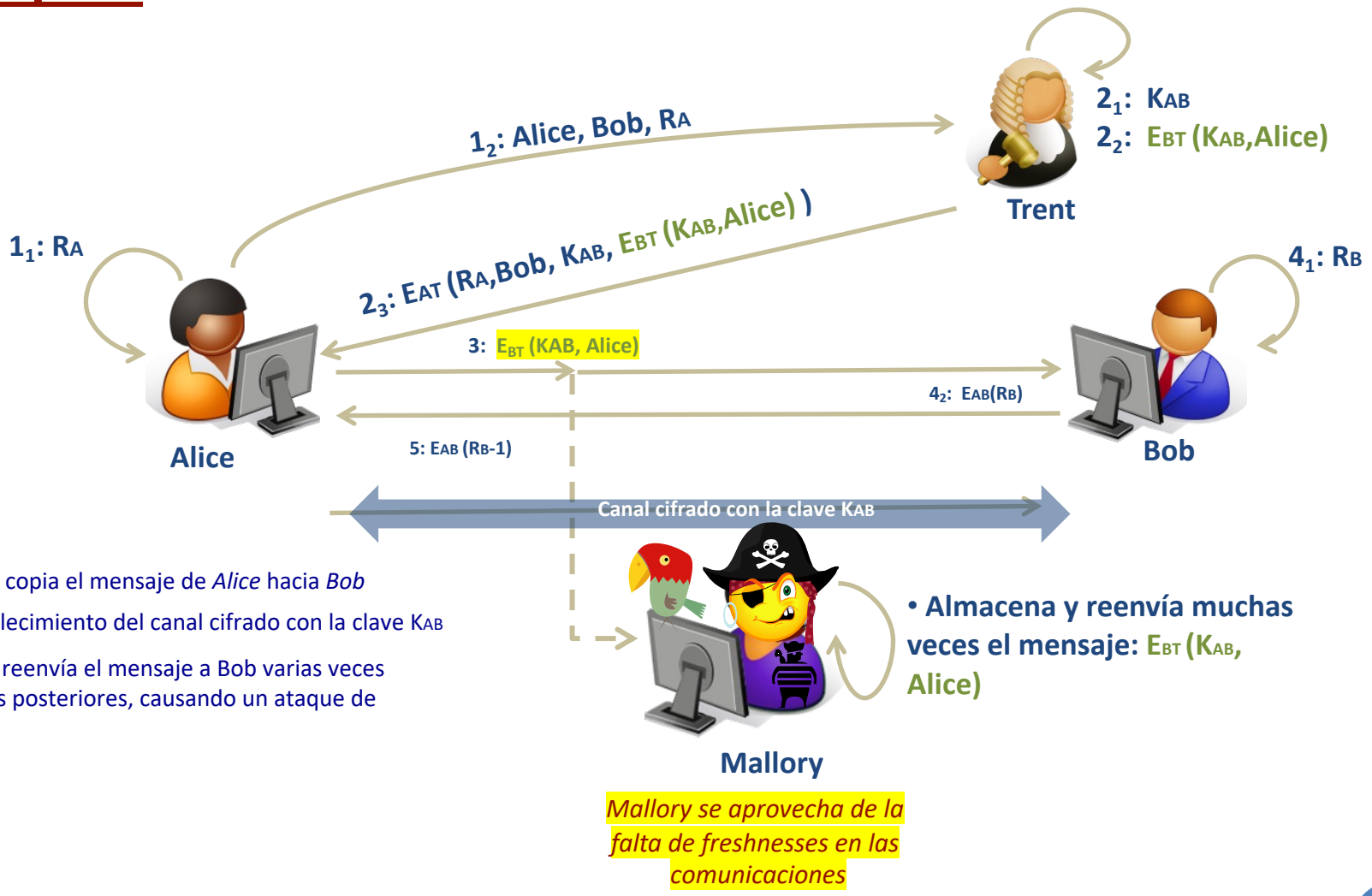
4: *Bob* responde a *Mallory* con un valor aleatorio (challenge).

5: *Mallory* descifra el valor aleatorio (porque conoce K_{AB}).

6: Mallory responde al challenge de Bob, y Bob piensa que habla con Alice.



- # Ataque 3



3: *Mallory* copia el mensaje de *Alice* hacia *Bob*

4, 5: Establecimiento del canal cifrado con la clave K_{AB}

6: *Mallory* reenvía el mensaje a Bob varias veces en tiempos posteriores, causando un ataque de repetición

Mallory se aprovecha de la falta de freshnesses en las comunicaciones

- Metiendo en el mensaje 3 un nonce:

1. $A \rightarrow S : A, B, N_A$
2. $S \rightarrow A : \{N_A, B, K_{A,B}, \{K_{A,B}, A\}_{K_{B,S}}\}_{K_{A,S}}$
3. $A \rightarrow B : \{K_{A,B}, A\}_{K_{B,S}}$
4. $B \rightarrow A : \{N_B\}_{K_{A,B}}$
5. $A \rightarrow B : \{N_B - 1\}_{K_{A,B}}$

Problema: la frescura de los mensajes solo se encuentra en los mensajes 1 y 2, pero no en el resto de mensajes

- **Solución**:

1. $A \rightarrow S : A, B, N_A$
2. $S \rightarrow A : \{N_A, B, K_{A,B}, \{K_{A,B}, A\}_{K_{B,S}}\}_{K_{A,S}}$
3. $A \rightarrow B : \{K_{A,B}, A\}_{K_{B,S}}$
4. $B \rightarrow A : \{N_B\}_{K_{A,B}}$
5. $A \rightarrow B : \{N_B - 1\}_{K_{A,B}}$

Solución: extender el uso del nonce en el resto de transacciones

- Protocolo **Amended Needham Schroeder protocol**
  - soluciona el fallo del anterior **Needham-Schroeder (en relación a los ataques de repetición)**

$A \rightarrow B : A$

$B \rightarrow A : E_{KBT}\{A, N_{b0}\}$

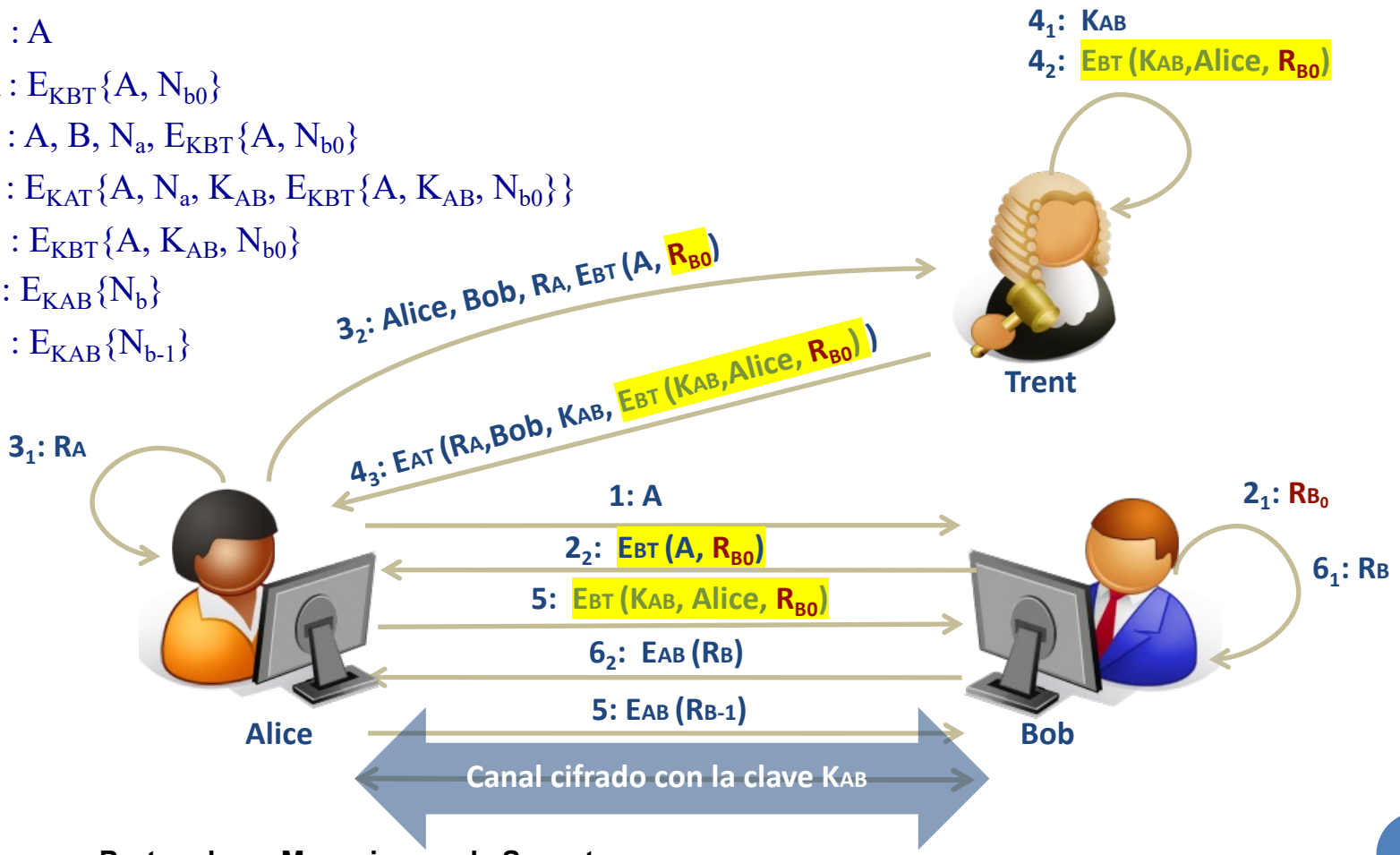
$A \rightarrow T : A, B, N_a, E_{KBT}\{A, N_{b0}\}$

$T \rightarrow A : E_{KAT}\{A, N_a, K_{AB}, E_{KBT}\{A, K_{AB}, N_{b0}\}\}$

$A \rightarrow B : E_{KBT}\{A, K_{AB}, N_{b0}\}$

$B \rightarrow A : E_{KAB}\{N_b\}$

$A \rightarrow B : E_{KAB}\{N_{b-1}\}$



- Protocolo **Otway-Rees**
  - soluciona también el fallo del **Needham-Schroeder**, aunque con un diseño diferente

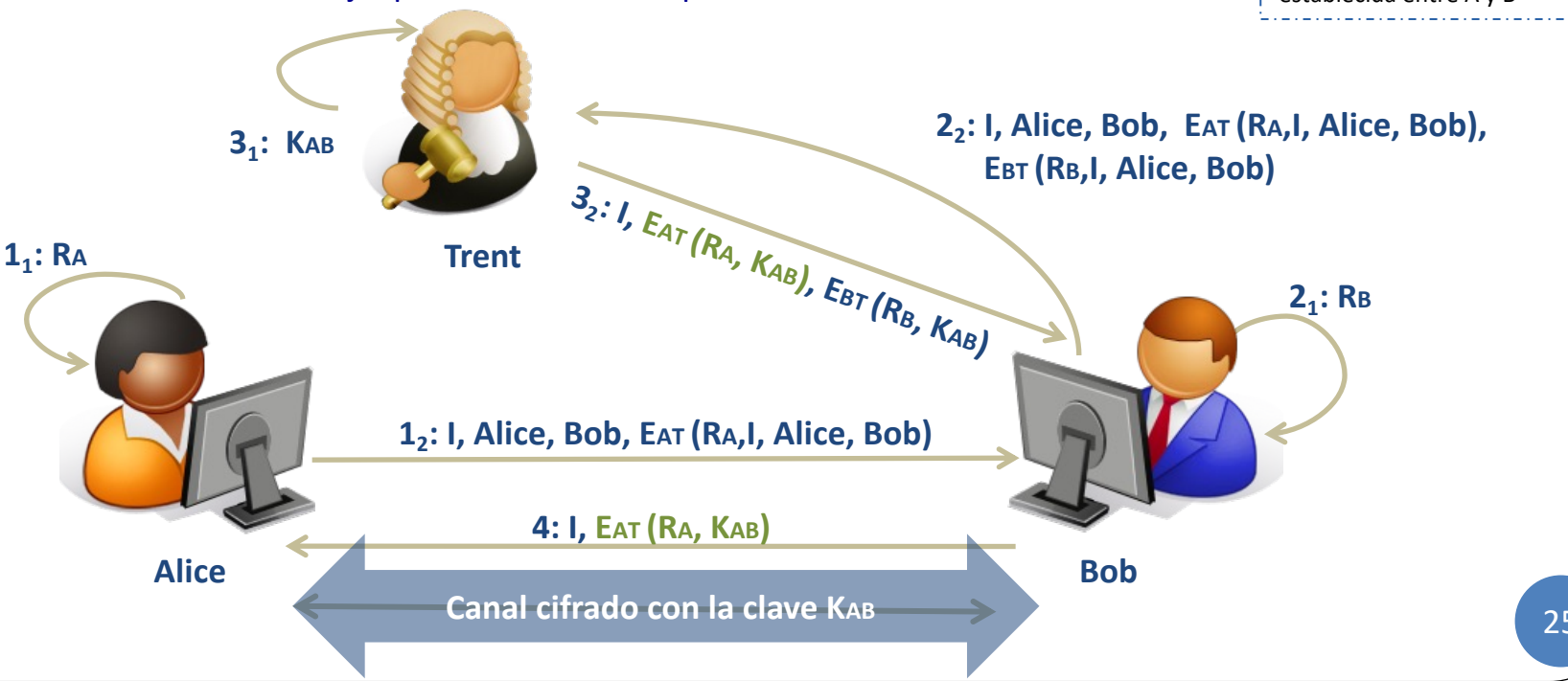
1: *Alice* genera el valor aleatorio R_A <1_1> y se lo envía hacia *Bob*, dentro de un mensaje cifrado con la clave que comparte con *Trent* <1_2>.

2: *Bob* genera un valor aleatorio R_B y se lo envía a *Trent* usando la clave que comparte con éste <2_2>. También le envía el mensaje que recibió de *Alice*.

3: *Trent* descifra el mensaje cifrado con la clave que comparte con *Alice*, genera la clave de sesión K_{AB} <3_1> y se la envía a *Bob* cifrada, junto a un mensaje para *Alice* <3_2>.

4: *Bob* envía a *Alice* el mensaje que recibió de *Trent* para ella.

I (Índice): I-ésima sesión establecida entre A y B



Otway-Rees

- Diseño formalizado:

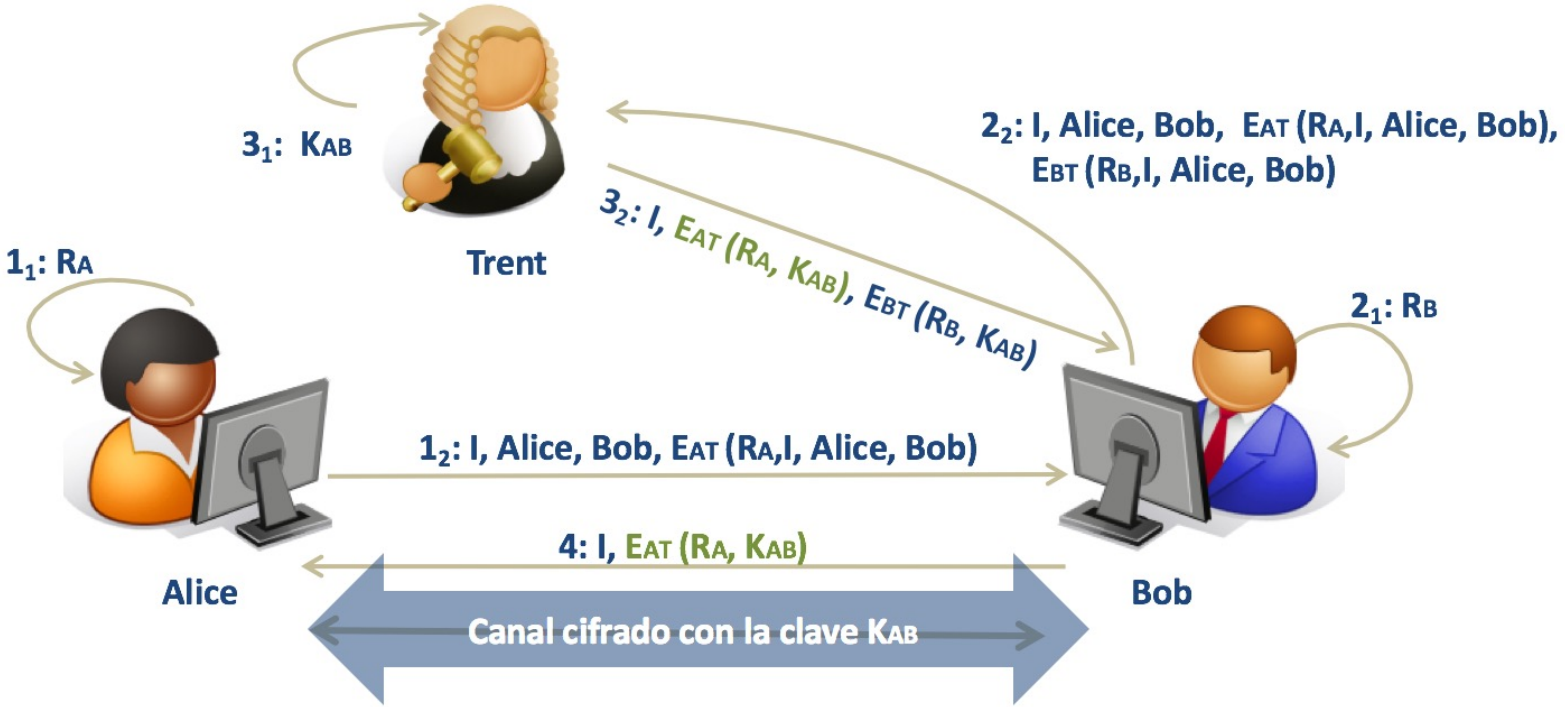
$A \rightarrow B : I, A, B, E_{KAT}\{N_a, I, A, B\}$

$B \rightarrow T : I, A, B, E_{KAT}\{N_a, I, A, B\}, E_{KBT}\{N_b, I, A, B\}$

$T \rightarrow B : I, E_{KAT}\{K_{AB}, N_a\}, E_{KBT}\{K_{AB}, N_b\}$

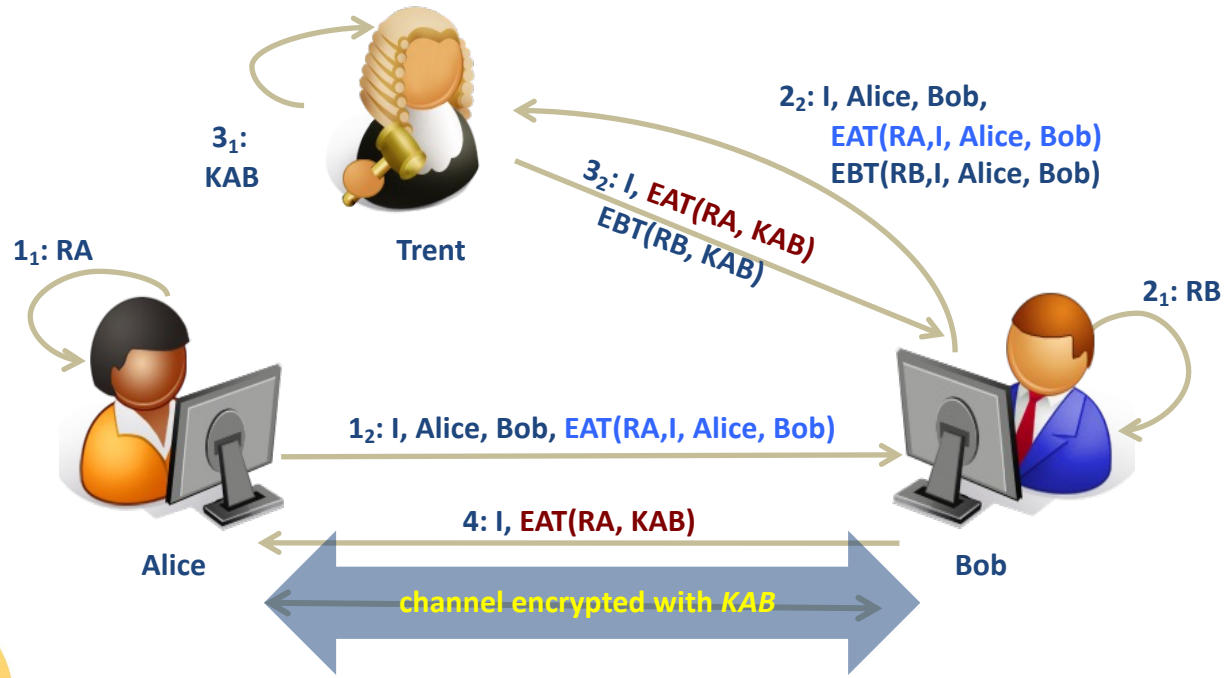
$B \rightarrow A : I, E_{KAT}\{K_{AB}, N_a\}$

Como se puede ver en el protocolo, Otway-Rees también intenta solucionar el problema del freshness en los mensajes



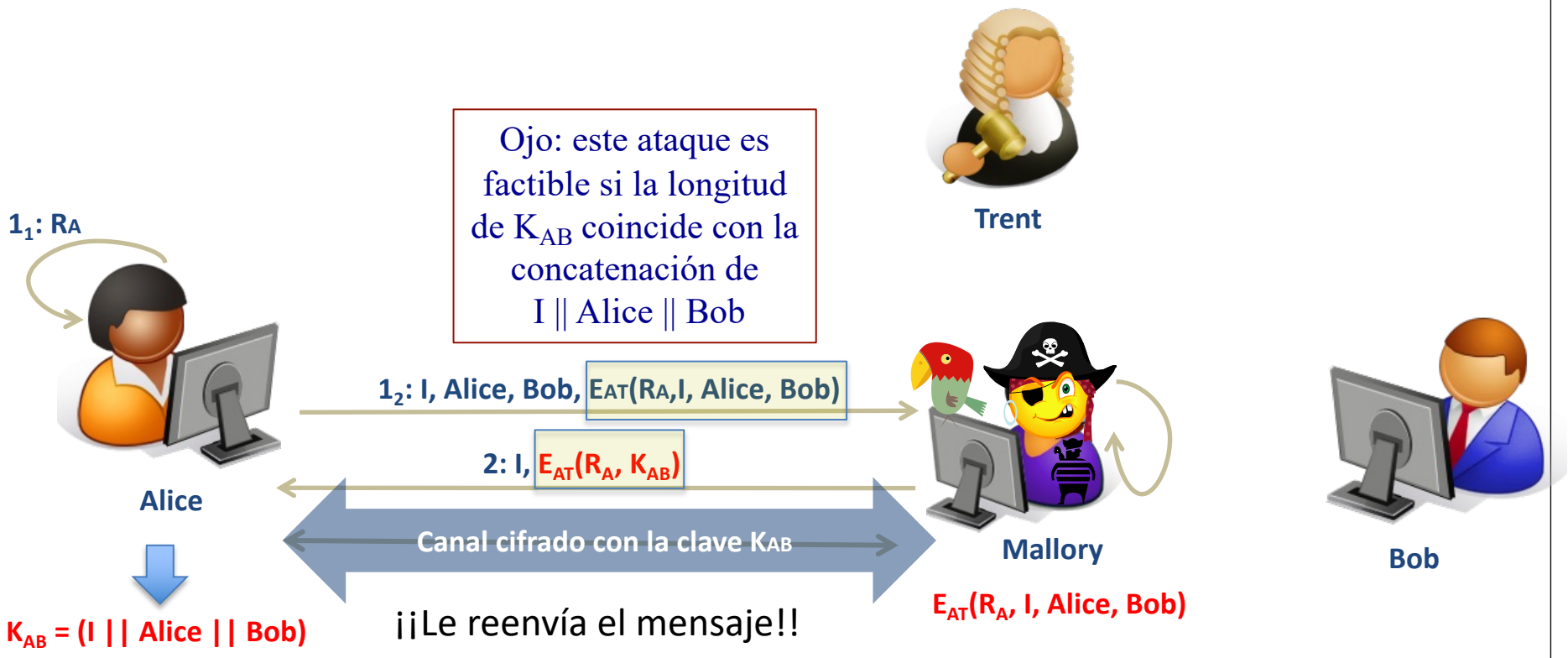
Otway-Rees

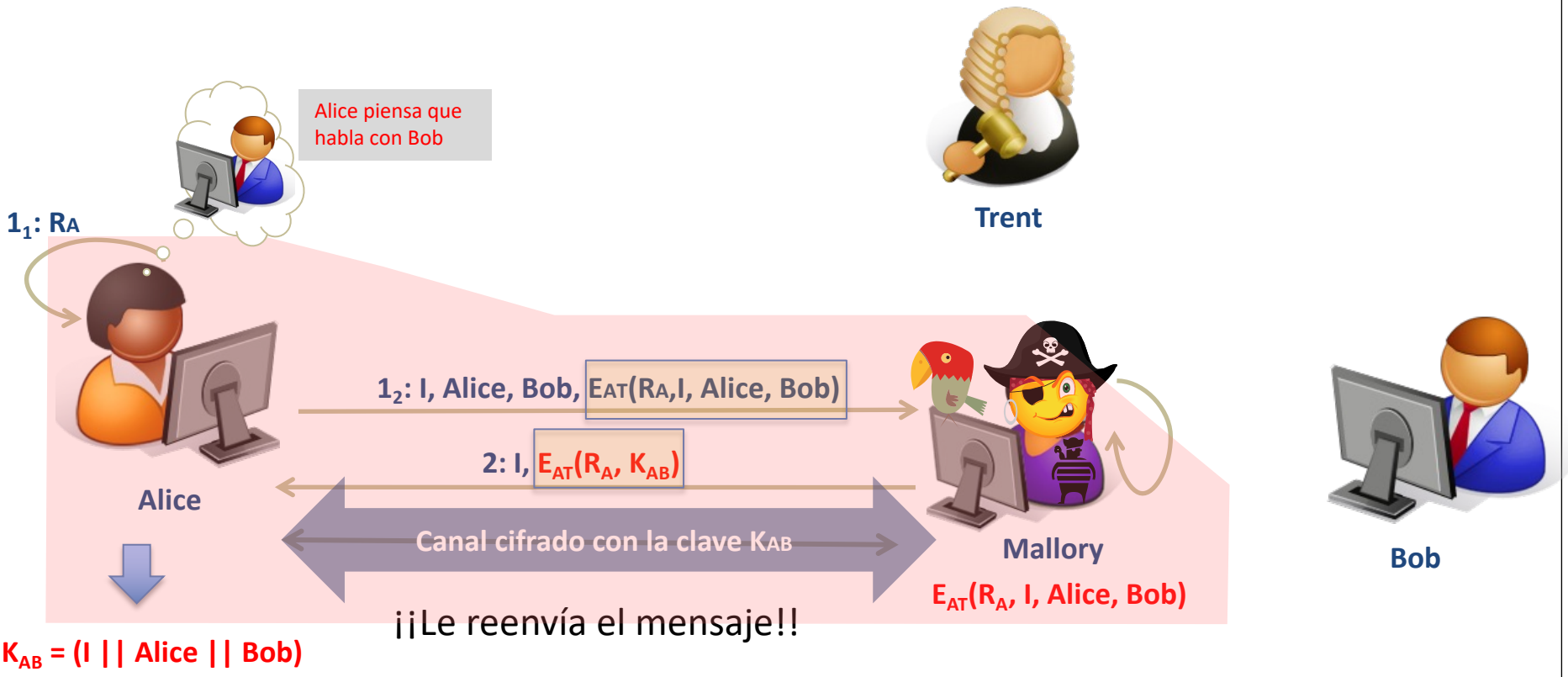
- Sin embargo, el protocolo presenta dos vulnerabilidades importantes



10 minutos – tic, tac, tic, tac …

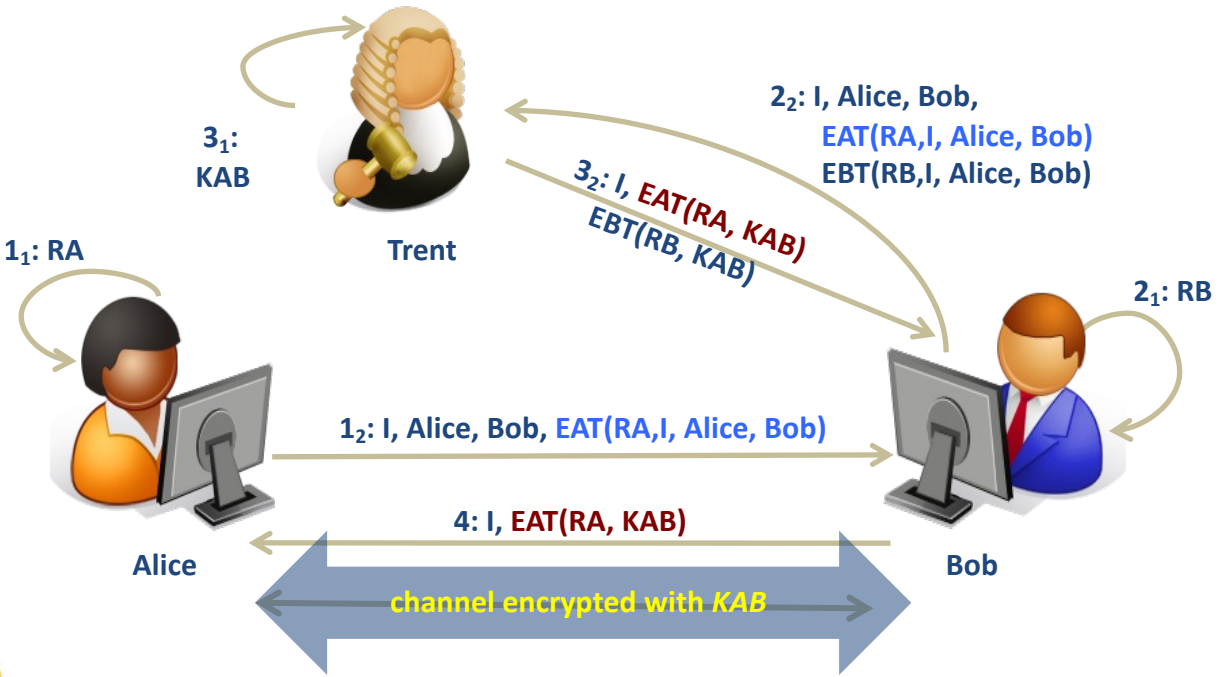
– **Primer agujero de seguridad**:




Alice piensa que habla con Bob
Trent
1₁: RA
1₂: I, Alice, Bob, EAT(RA,I, Alice, Bob)
2: I, EAT(RA, KAB)
Alice
Canal cifrado con la clave KAB
Mallory
EAT(RA, I, Alice, Bob)
Bob
¡¡Le reenvía el mensaje!!
KAB = (I || Alice || Bob)

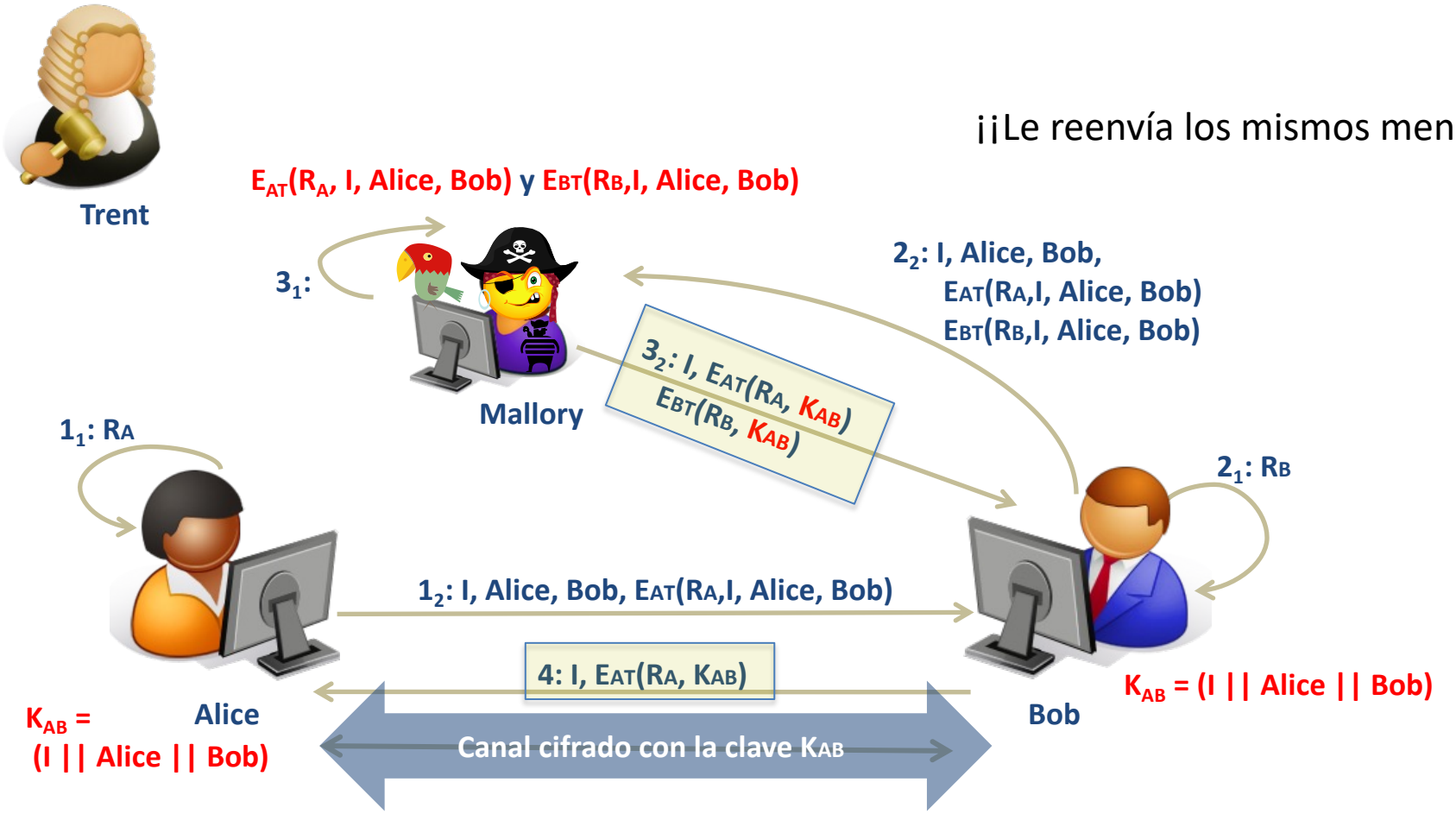
Otway-Rees

- ¿ Y el segundo agujero ?



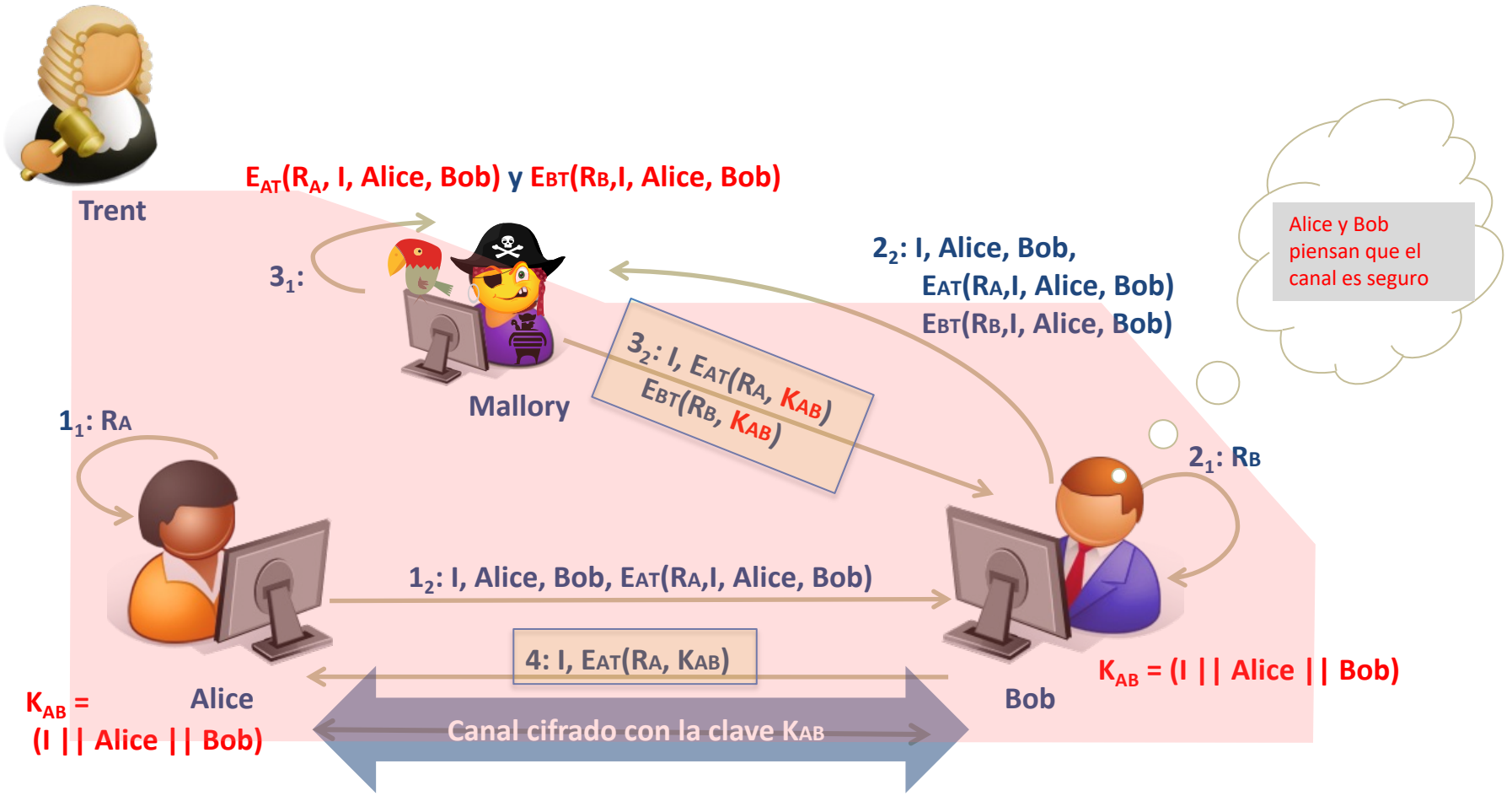
10 minutos – tic, tac, tic, tac …

– **Segundo agujero de seguridad**:



¡¡Le reenvía los mismos mensajes!!

– **Segundo agujero de seguridad**:



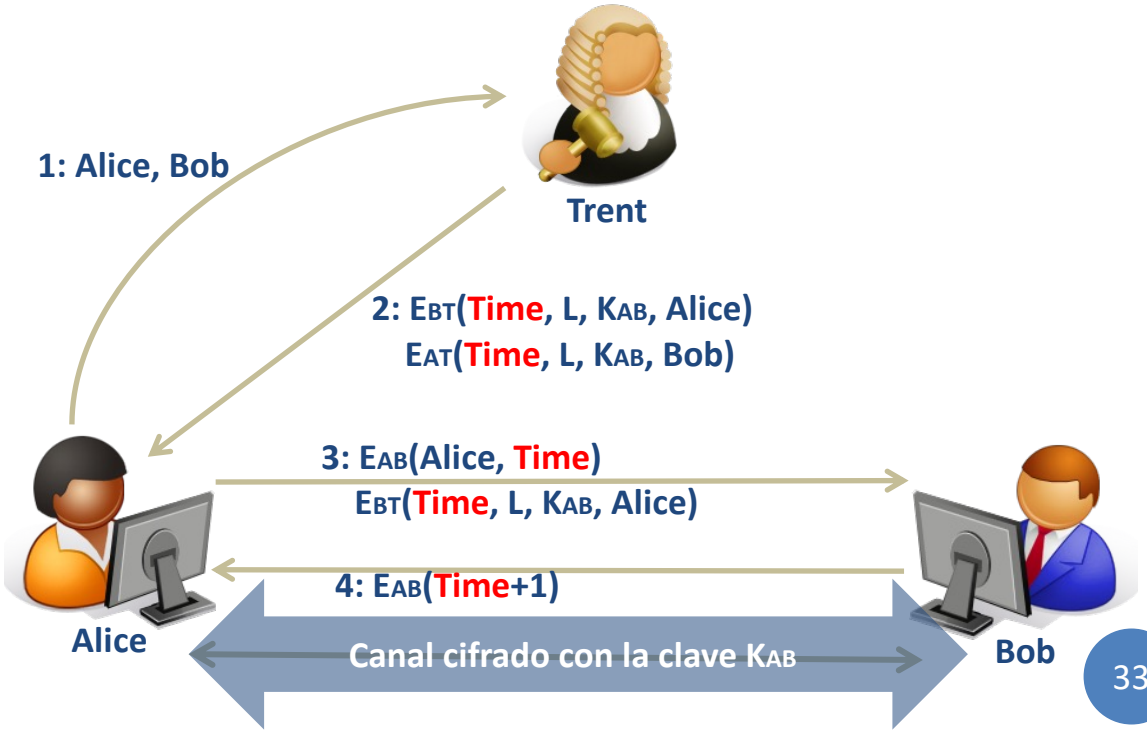
- Protocolo **Kerberos**

1: *Alice* envía un mensaje a *Trent* con su identidad y la identidad de *Bob*.

2: *Trent* genera un mensaje con un *timestamp* (*Time*), un *tiempo de vida* (*L*), una clave de sesión aleatoria, y la identidad de *Alice*. Lo cifra con la clave compartida con *Bob*. Prepara un mensaje similar para *Alice*.
Envía ambos mensajes cifrados a *Alice*.

3: *Alice* obtiene K_{AB}, genera un mensaje con su identidad y el timestamp, y lo cifra con K_{AB} para enviárselo a *Bob*. *Alice* también envía a *Bob* el mensaje cifrado que recibió de *Trent*.

4: *Bob* genera un mensaje que consta del timestamp más uno, lo cifra con K_{AB} y se lo envía a *Alice*.



- # Protocolo **Kerberos**

1: *Alice* envía un mensaje a *Trent* con su identidad y la identidad de *Bob*.

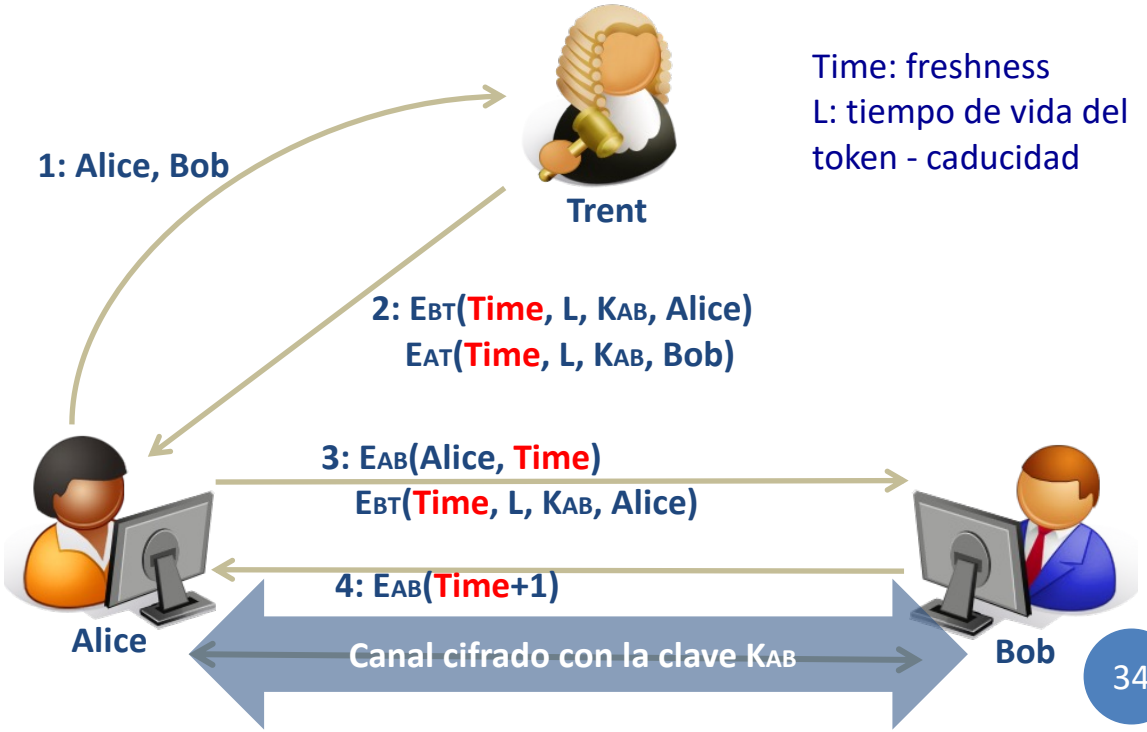
2: *Trent* genera un mensaje con un *timestamp* (*Time*), un *tiempo de vida* (*L*), una clave de sesión aleatoria, y la identidad de *Alice*. Lo cifra con la clave compartida con *Bob*. Prepara un mensaje similar para *Alice*.
Envía ambos mensajes cifrados a *Alice*.

3: *Alice* obtiene K_{AB}, genera un mensaje con su identidad y el timestamp, y lo cifra con K_{AB} para enviárselo a *Bob*. *Alice* también envía a *Bob* el mensaje cifrado que recibió de *Trent*.

4: *Bob* genera un mensaje que consta del timestamp más uno, lo cifra con K_{AB} y se lo envía a *Alice*.

- Asume que los relojes de todos los sistemas están sincronizados
- En la práctica se sincronizan en el rango de unos pocos minutos

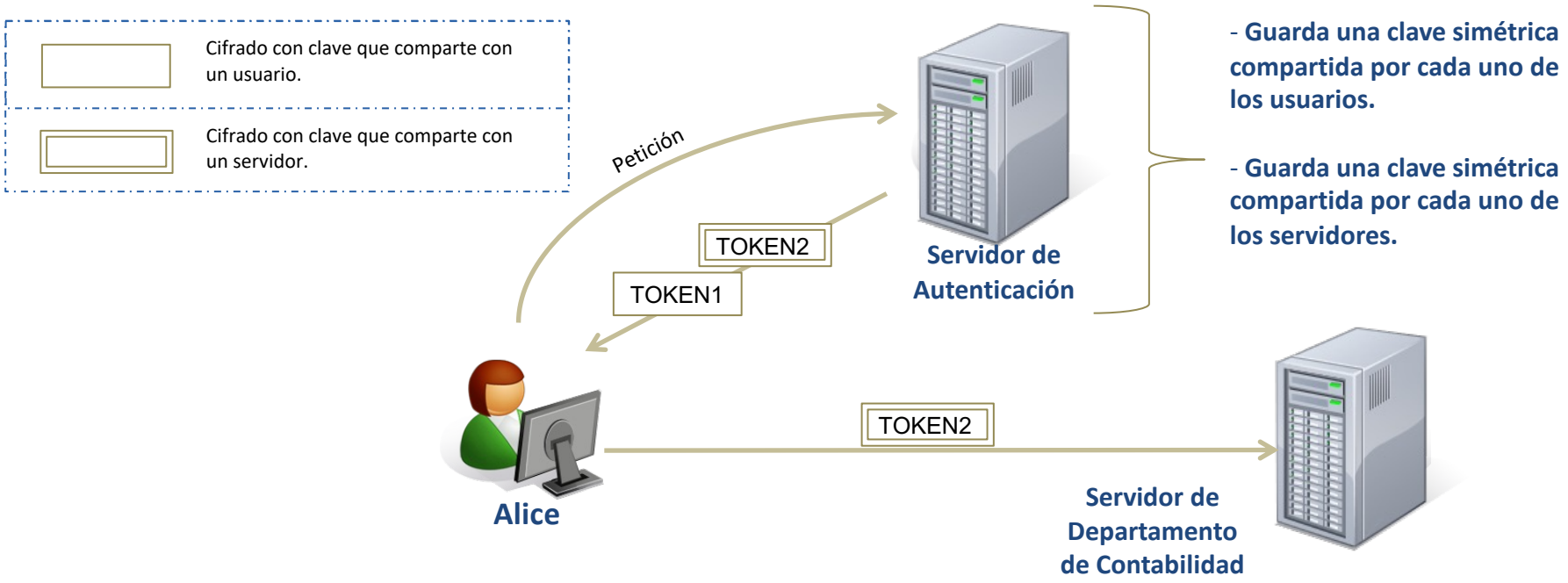
- Por fallos del sistema o por sabotaje, los relojes pueden desincronizarse (➔ ataque de denegación de servicio)



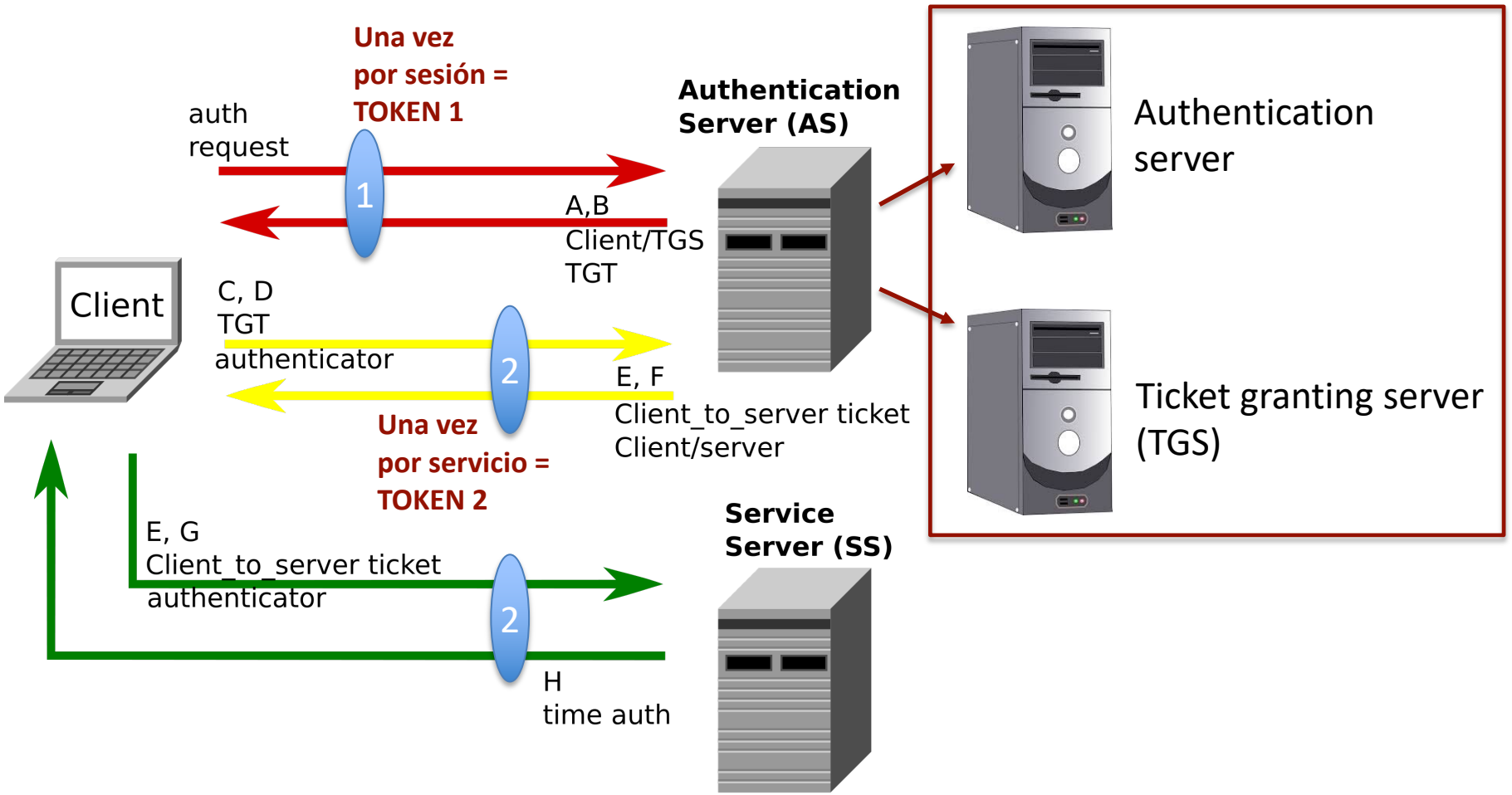
Time: freshness
L: tiempo de vida del token - caducidad

Kerberos

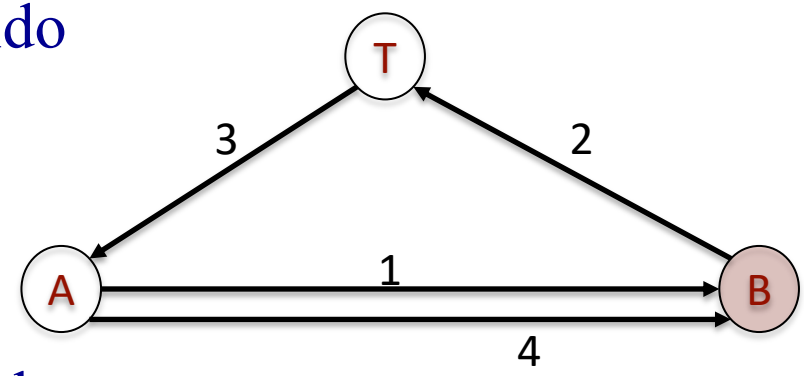
- Ejemplo de escenario de uso de Kerberos
  - Campus universitario, empresa, …
  - Se usa Kerberos para evitar que cada usuario tenga una cuenta en cada servidor con el que va a contactar, y un tiempo límite de uso
    - en el escenario de abajo *Alice* accede al Servidor del Departamento de Contabilidad sin tener una cuenta en ese servidor



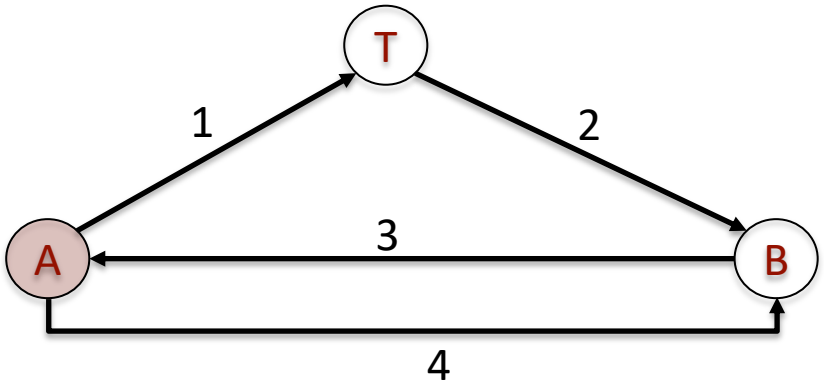
Kerberos


Una vez
por sesión =
TOKEN 1
auth
request
1
A,B
Client/TGS
TGT
Authentication
Server (AS)
Authentication
server
Client
C, D
TGT
authenticator
2
E, F
Client_to_server ticket
Client/server
Ticket granting server
(TGS)
Una vez
por servicio =
TOKEN 2
E, G
Client_to_server ticket
authenticator
2
H
time auth
Service
Server (SS)

- Aparte de los protocolos anteriores, hay otros que combinan múltiples tipos de estrategias (a nivel de modelos como de mecanismos de seguridad):
  - PUSH extendido

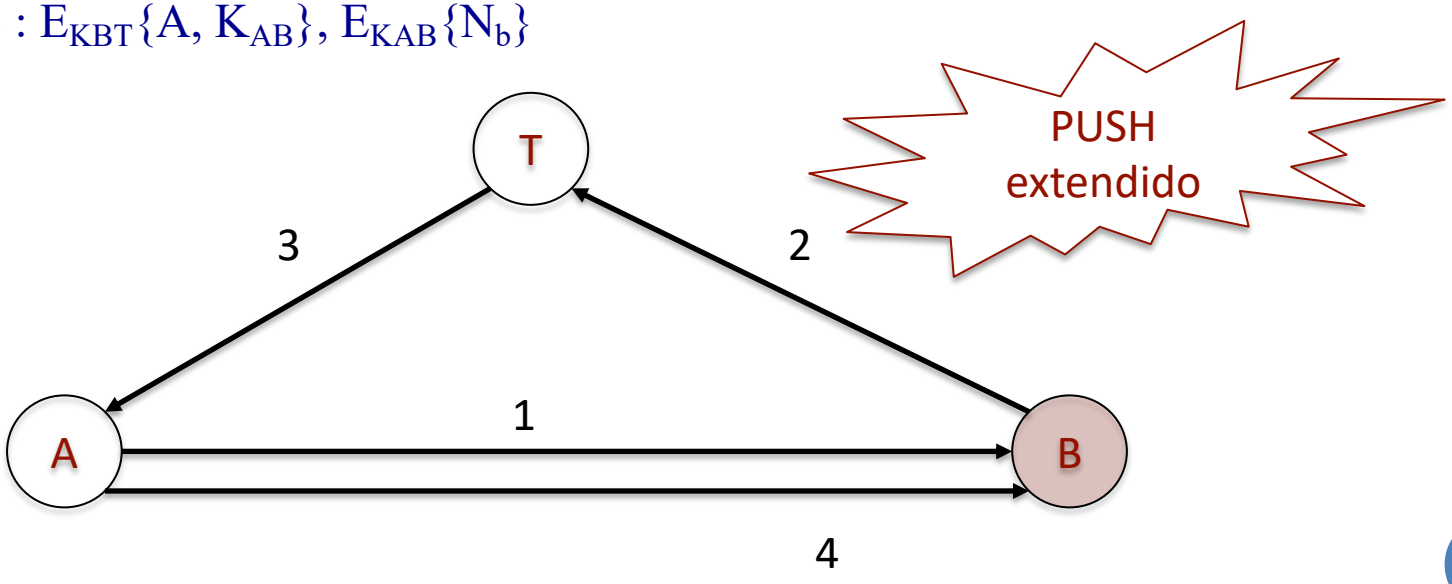


  - PULL extendido



- **Yahalom**
  - Objetivo: permitir a Trent generar la clave de sesión K_{AB} y enviar dicha clave a Alice (de manera directa) y a B (de manera indirecta)

$A \rightarrow B : A, N_a$
$B \rightarrow T : B, E_{KBT}\{A, N_a, N_b\}$
$T \rightarrow A : E_{KAT}\{B, K_{AB}, N_a, N_b\}, E_{KBT}\{A, K_{AB}\}$
$A \rightarrow B : E_{KBT}\{A, K_{AB}\}, E_{KAB}\{N_b\}$



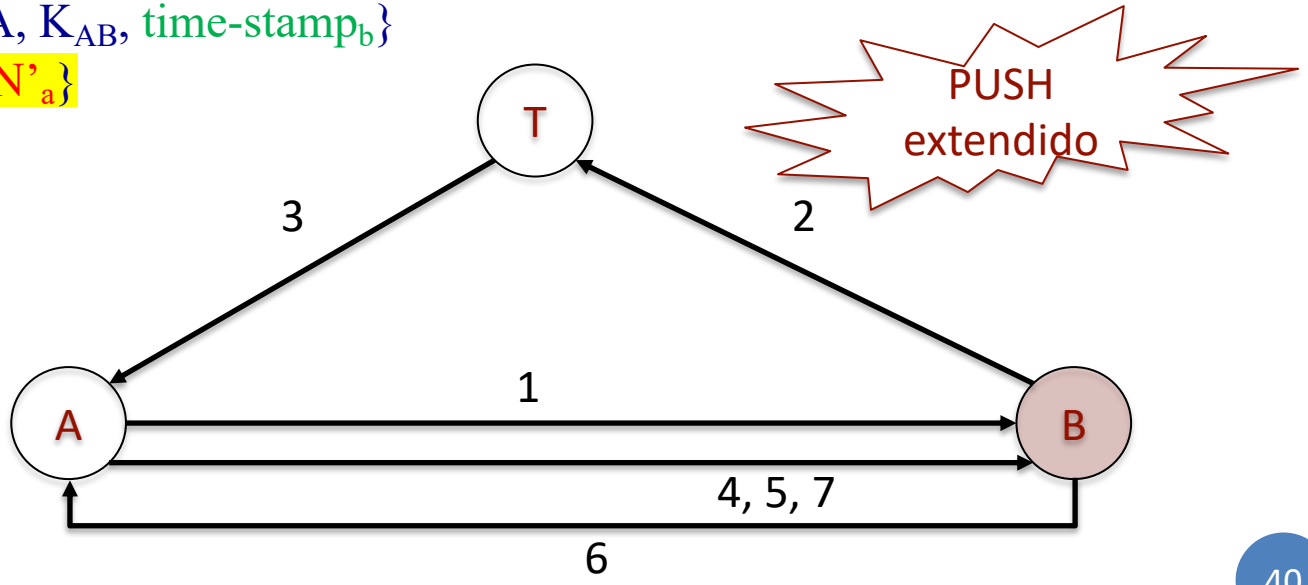
- Dado el protocolo anterior:

$A \rightarrow B : A, N_a$
$B \rightarrow T : B, E_{KBT}\{A, N_a, N_b\}$
$T \rightarrow A : E_{KAT}\{B, K_{AB}, N_a, N_b), E_{KBT}\{A, K_{AB}\}$
$A \rightarrow B : E_{KBT}\{A, K_{AB}\}, E_{KAB}\{N_b\}$

– ¿Hay desafío y respuesta?

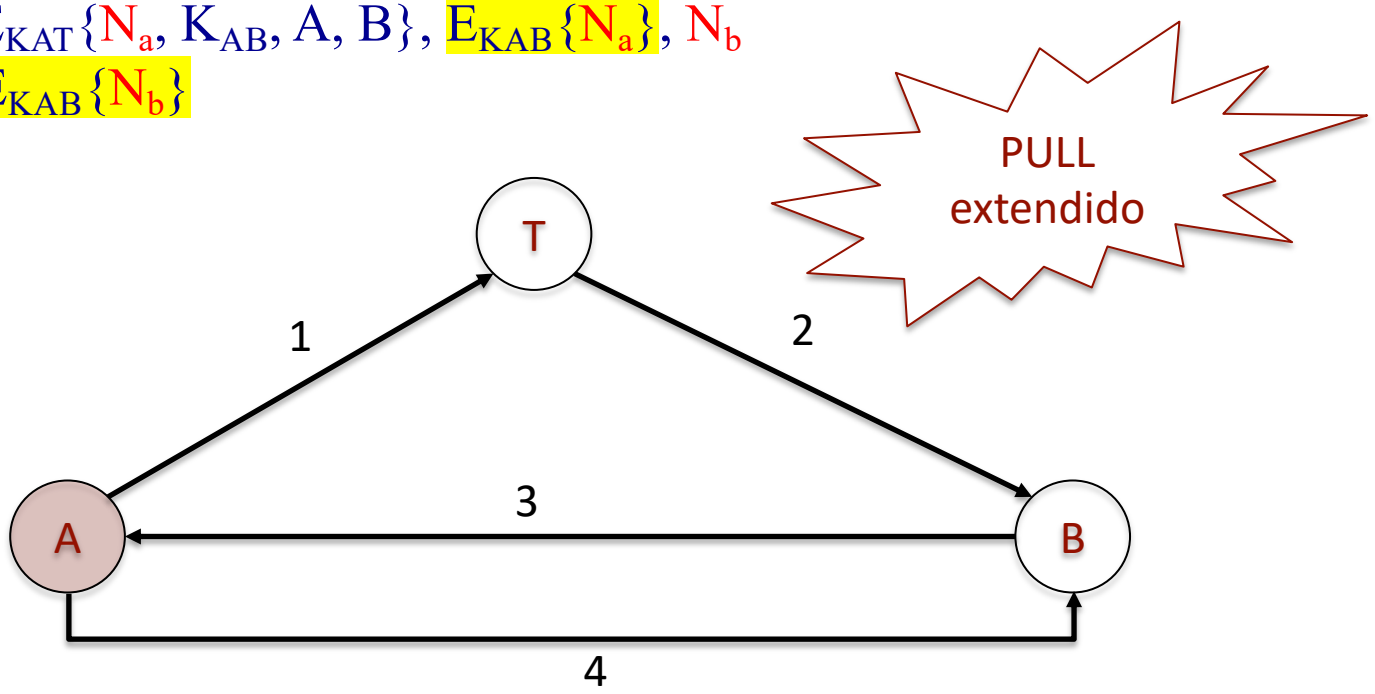
- **Neuman Stubblebine**
  - Objetivo: combinar formas para verificar la frescura de las transacciones – *time-stamps, nonces*

$A \rightarrow B : A, N_a$
$B \rightarrow T : B, EK_{BT}\{A, N_a, \text{time-stamp}_b\}, N_b$
$T \rightarrow A : EK_{AT}\{B, K_{AB}, N_a, \text{time-stamp}_b\}, EK_{BT}\{A, K_{AB}, \text{time-stamp}_b\}, N_b$
$A \rightarrow B : EK_{BT}\{A, K_{AB}, \text{time-stamp}_b\}, EK_{AB}\{N_b\}$
$A \rightarrow B : N'_a, EK_{BT}\{A, K_{AB}, \text{time-stamp}_b\}$
$B \rightarrow A : N'_b, EK_{AB}\{N'_a\}$
$A \rightarrow B : EK_{AB}\{N'_b\}$



- **Kao Chow**

$A \rightarrow T : A, B, N_a$
$T \rightarrow B : E_{KAT}\{N_a, K_{AB}, A, B\}, E_{KBT}\{N_a, K_{AB}, A, B\}$
$B \rightarrow A : E_{KAT}\{N_a, K_{AB}, A, B\}, E_{KAB}\{N_a\}, N_b$
$A \rightarrow B : E_{KAB}\{N_b\}$



- Hemos visto que existen diversos protocolos para solucionar el problema de la administración/intercambio de claves
  - El protocolo a elegir depende del escenario y de lo que se quiere proteger
    - ¿Se quiere aprovechar los canales de comunicación?
    - ¿Quién ha de contactar primero con el KDC: Alice o Bob?
    - ¿Debe el KDC contactar directamente con ambos o es suficiente que lo haga con sólo uno de ellos?
    - ¿Se quiere proteger las comunicaciones de posibles ataques replay?
    - ¿Se quiere desafío y respuesta?
    - ….

- Usar un KDC no está exento de potenciales problemas:
  1. el KDC posee suficiente **información para suplantar** a cualquier usuario
     - y si un intruso llega hasta él todos los documentos cifrados que circulan por la red se hacen vulnerables
  2. el KDC representa un **único punto de fallos** (o ataques)
     - si queda inutilizado, nadie puede establecer comunicaciones seguras dentro de la red
  3. el **rendimiento** de todo el sistema **puede bajar** cuando el KDC se convierte en un cuello de botella
     - lo cual no es difícil ya que todos los usuarios necesitan comunicar con él de forma frecuente con objeto de obtener las claves