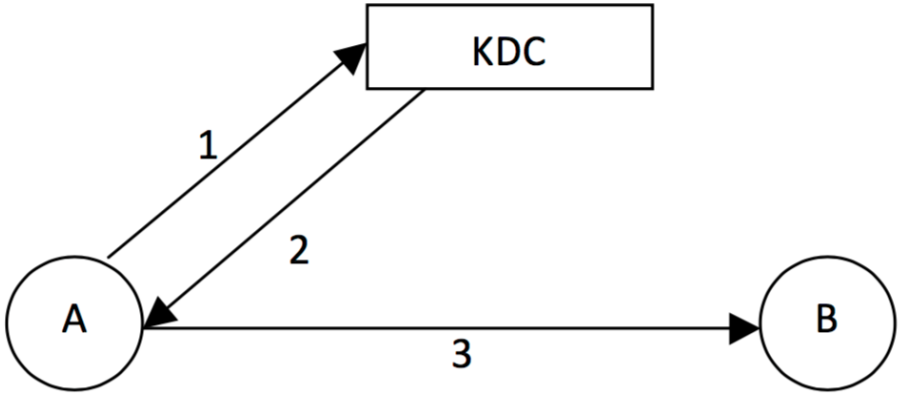
SEGURIDAD DE LA INFORMACIÓN

TEMA 3 – PARTE 2

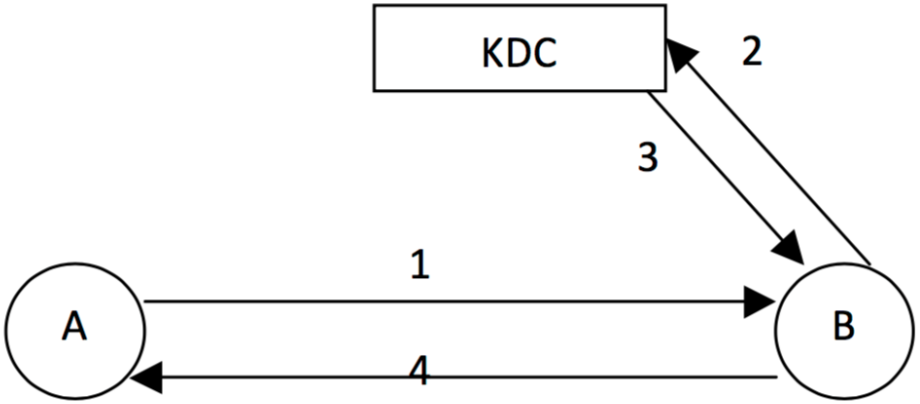
ESQUEMAS, PROTOCOLOS Y MECANISMOS DE SOPORTE
(A LA SEGURIDAD DE APLICACIONES Y DE REDES)

Recordatorio…

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



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



- # Protocolo de **Needham-Schroeder**

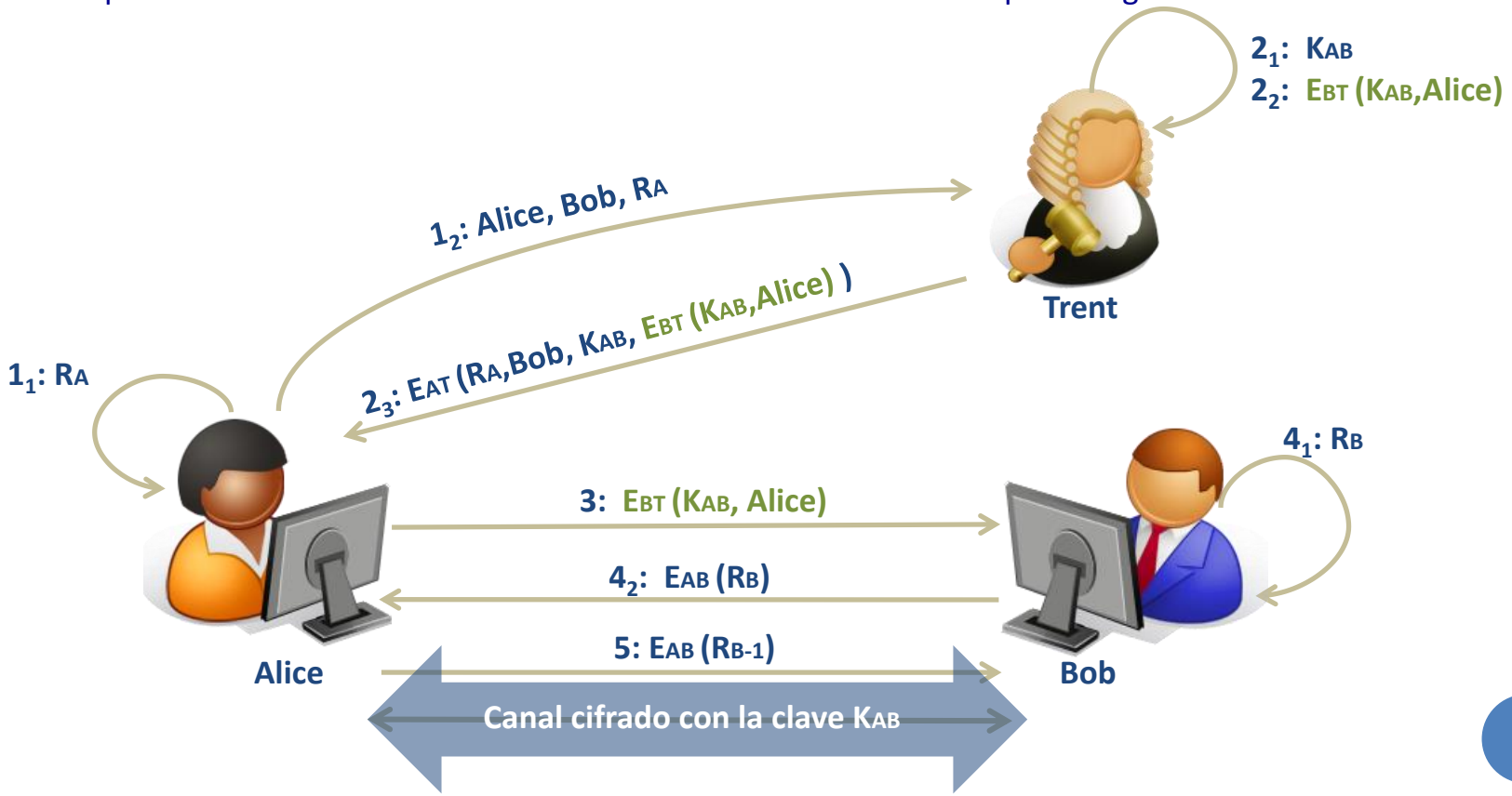
1: *Alice* genera el valor aleatorio R_A <1_1>, y se lo envía a *Trent* <1_2>.

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. $A \rightarrow S : A, B, \boxed{N_A}$ → **freshness**
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"

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

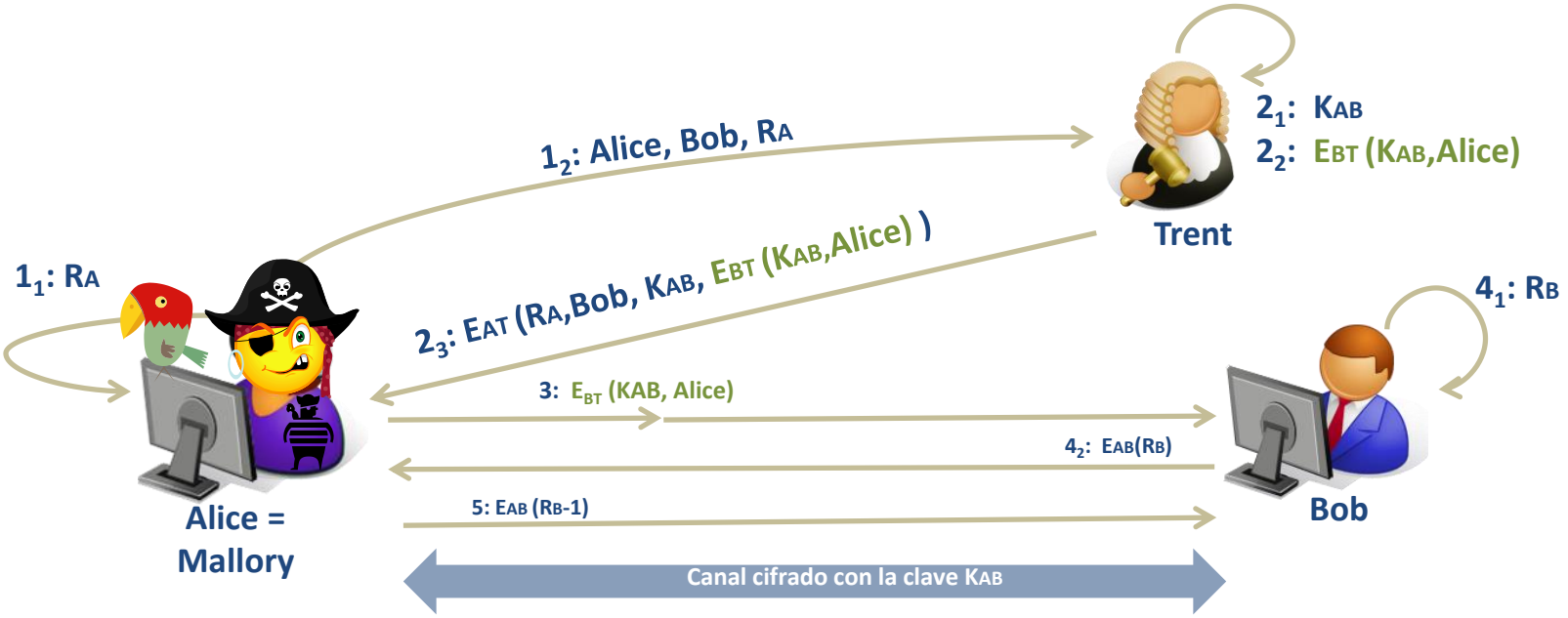
- Este protocolo tiene varios **fallos de seguridad**, que fueron descubiertos años después de su funcionamiento

10 minutos – tic, tac, tic, tac …

- Este protocolo tiene varios **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}
  - **Ataque 2**: Mallory puede suplantar la identidad de Bob si éste consigue derivar la clave K_{BT}
  - **Ataque 3**: Mallory puede producir un ataque de DoS debido a un ataque de repetición, especialmente en las últimas fases del protocolo

- # Ataque 1

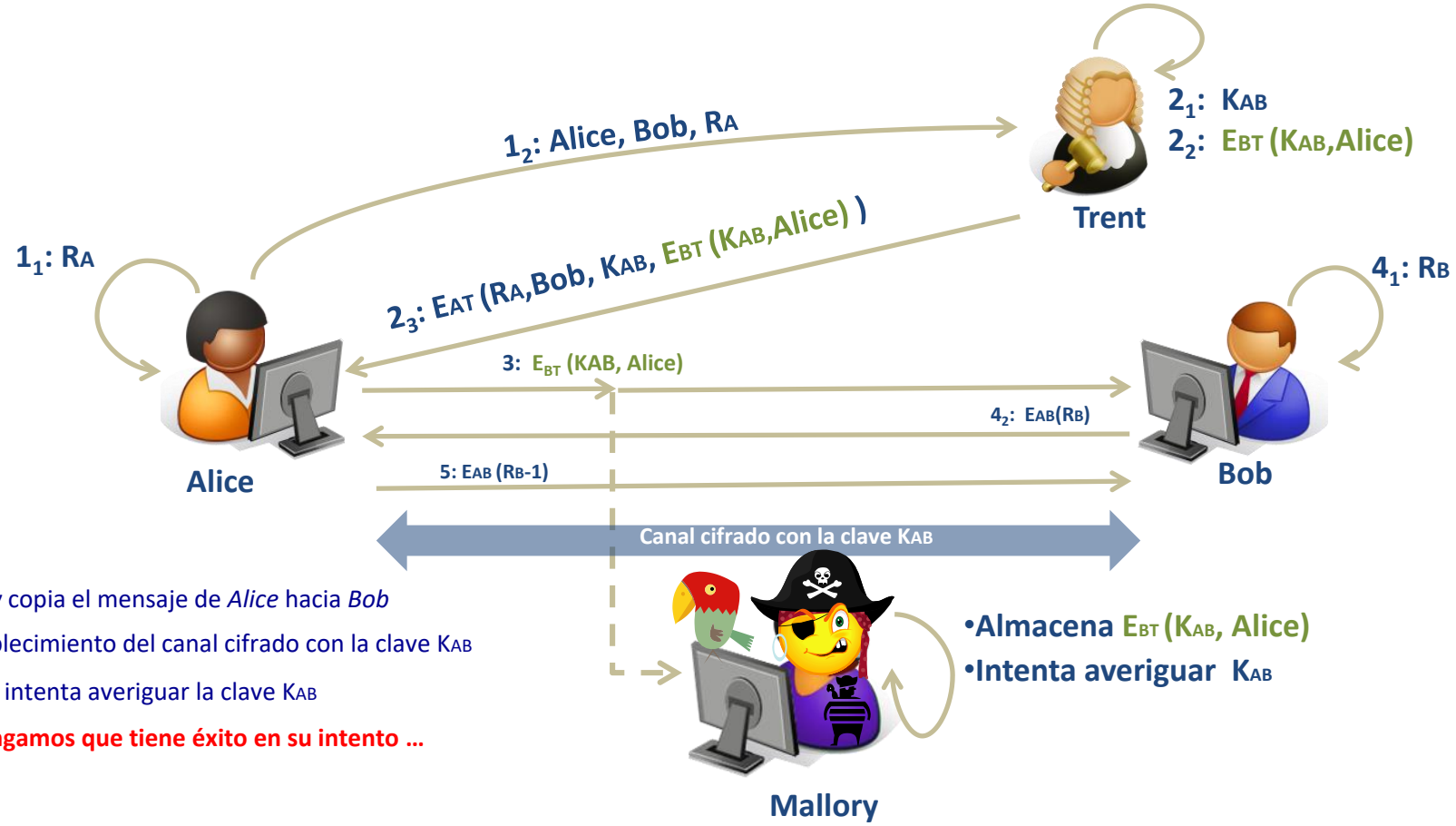


0: *Mallory* copia cualquier mensaje de *Alice* hacia *Trent* en el pasado y deriva la clave K_{AT} . A partir de aquí, todos los mensajes quedan comprometidos

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 …

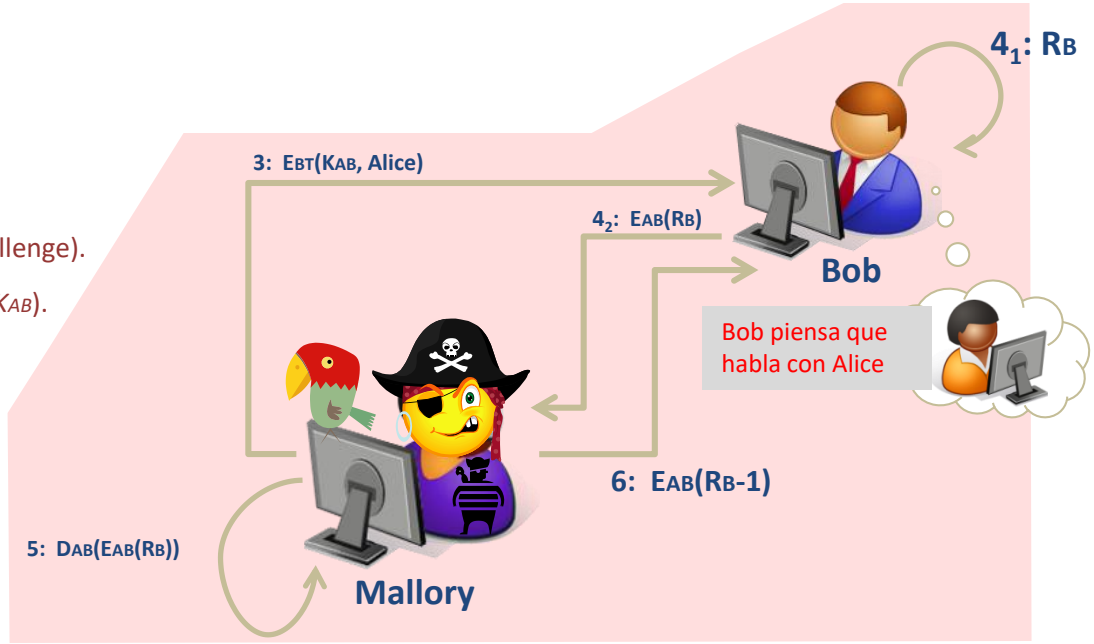
- continuación…

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

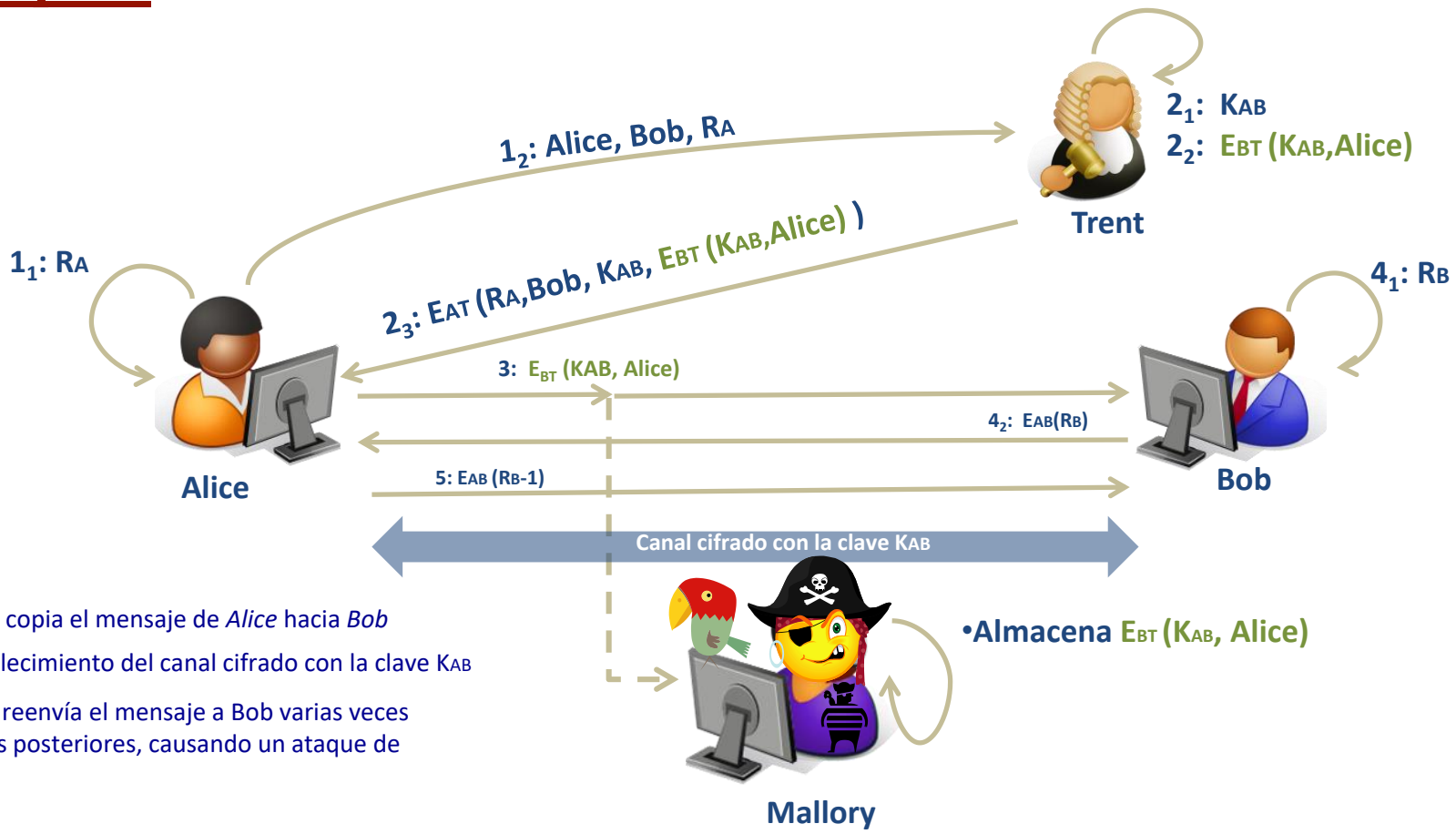
4: *Bob* responde a *Mallory* con un valor aleatorio (challenge).

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

- 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: el freshness 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)**

$A \rightarrow B : A$

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

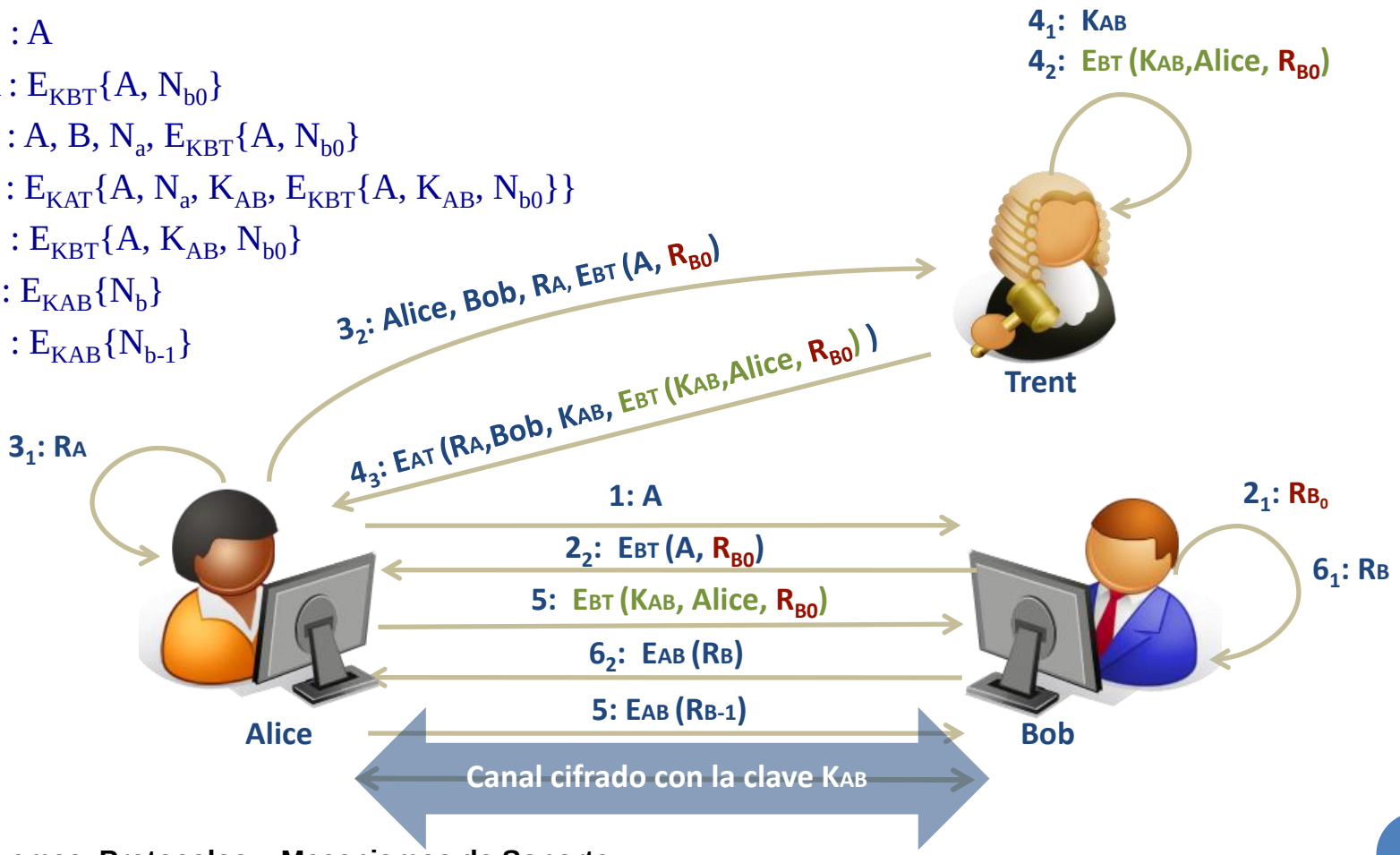
$A \rightarrow T : A, B, N_a, E_{KBT}\{A, N_{b0}\}$

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

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

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

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



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

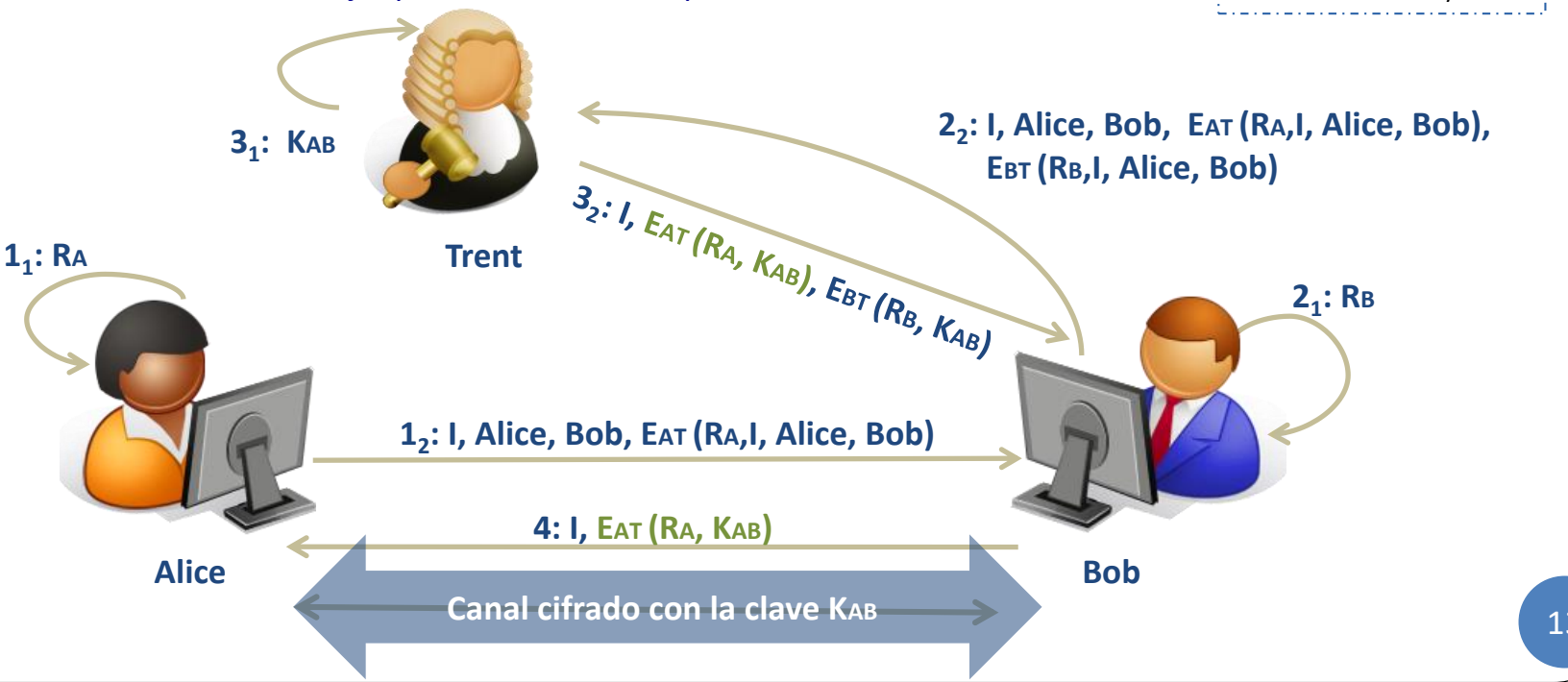
1: *Alice* genera el valor aleatorio R_A <1_1> y se lo envía hacia *Bob*, dentro de un mensaje cifrado con la clave que comparte con *Trent* <1_2>.

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

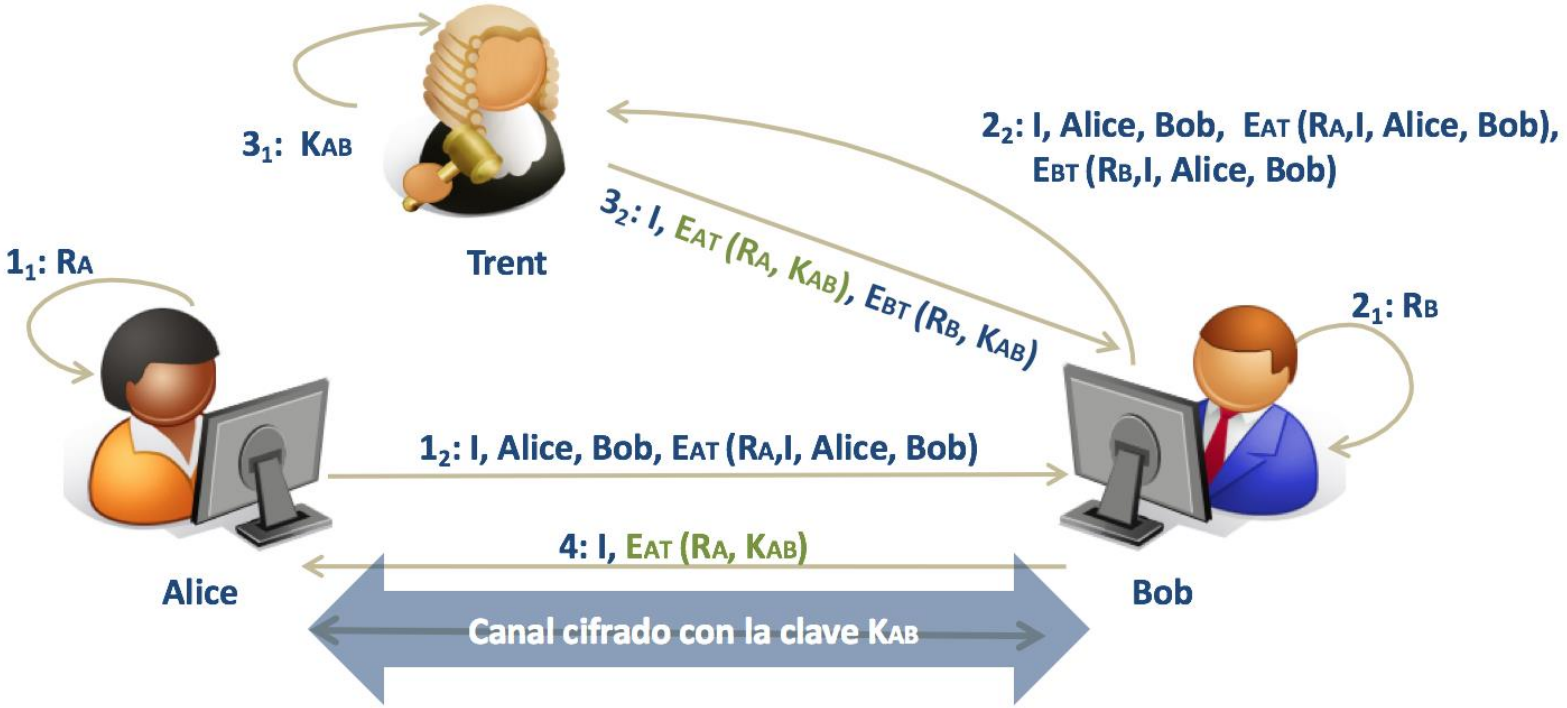
- Diseño formalizado:

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

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

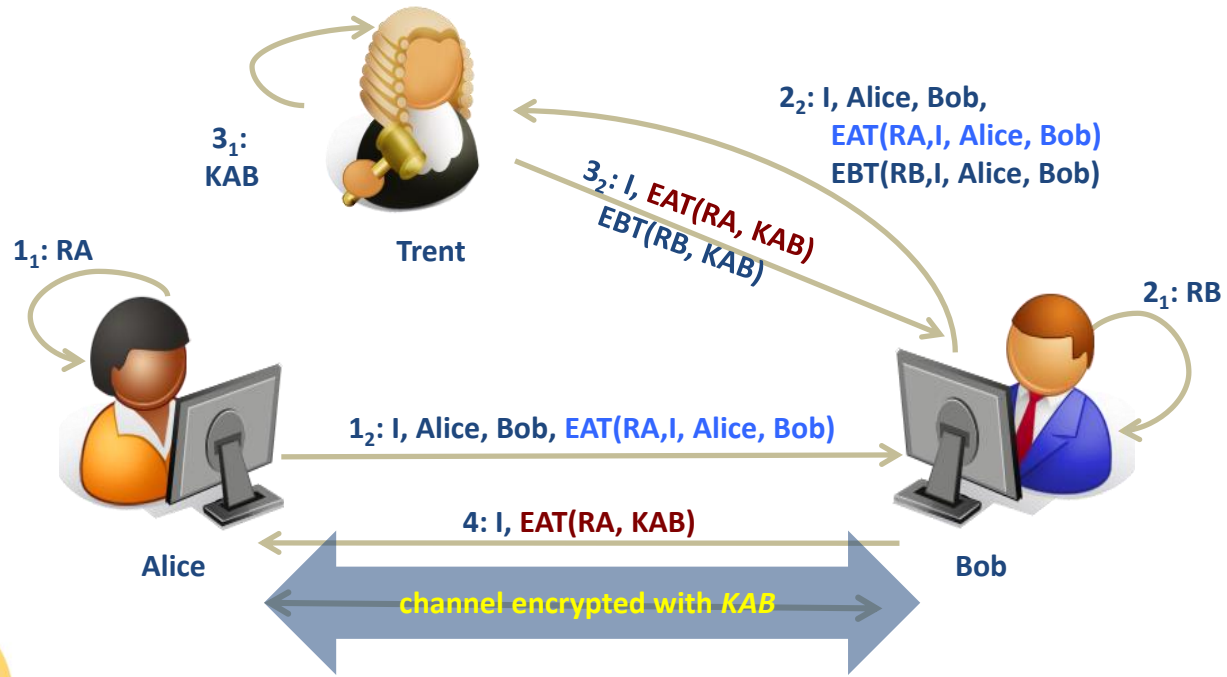
$T \rightarrow B : I, E_{KAT}\{K_{AB}, N_a\}, E_{KBT}\{K_{AB}, N_b\}$

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



Otway-Rees

- Sin embargo, el protocolo presenta dos vulnerabilidades importantes



10 minutos – tic, tac, tic, tac …

- Este protocolo también tiene un **fallo de seguridad**, y concretamente dos:
  - **Primer agujero de seguridad**:

Ojo: este ataque es factible si la longitud de K_{AB} coincide con la concatenación de I || Alice || Bob

Trent

1_1: R_A

Alice

1_2: I, Alice, Bob, $E_{AT}(R_A, I, Alice, Bob)$

2: I, $E_{AT}(R_A, K_{AB})$

Canal cifrado con la clave K_{AB}

Mallory

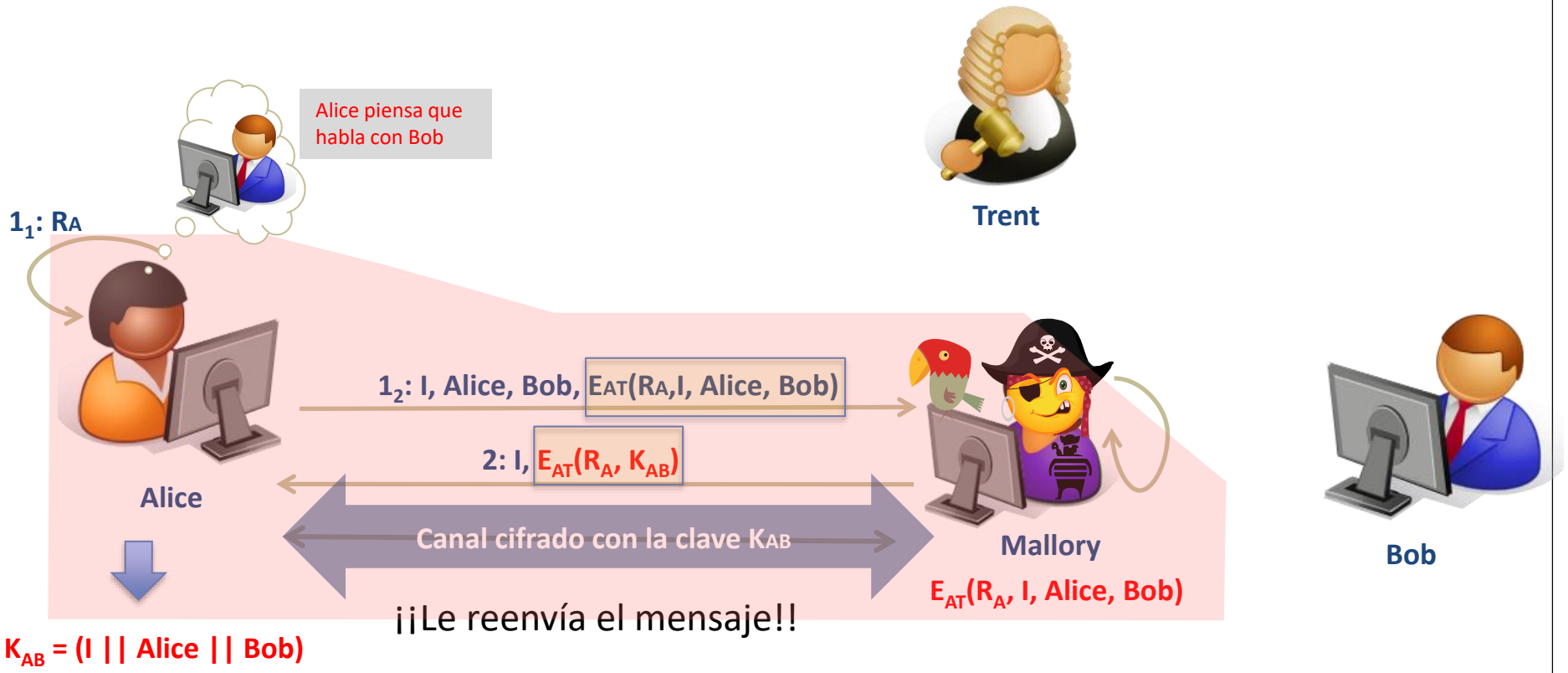
$E_{AT}(R_A, I, Alice, Bob)$

Bob

¡¡Le reenvía el mensaje!!

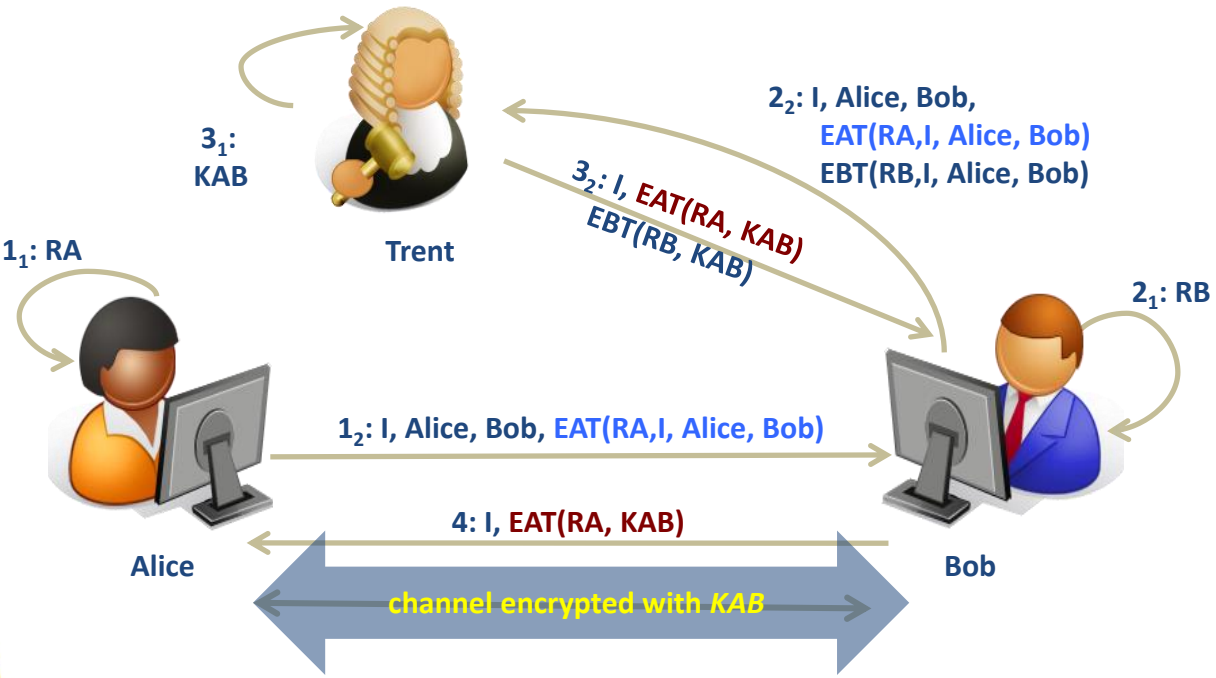
K_{AB} = (I || Alice || Bob)

- Otway-Rees también tiene **fallos de seguridad**, y concretamente dos:
  - Primer agujero de seguridad:



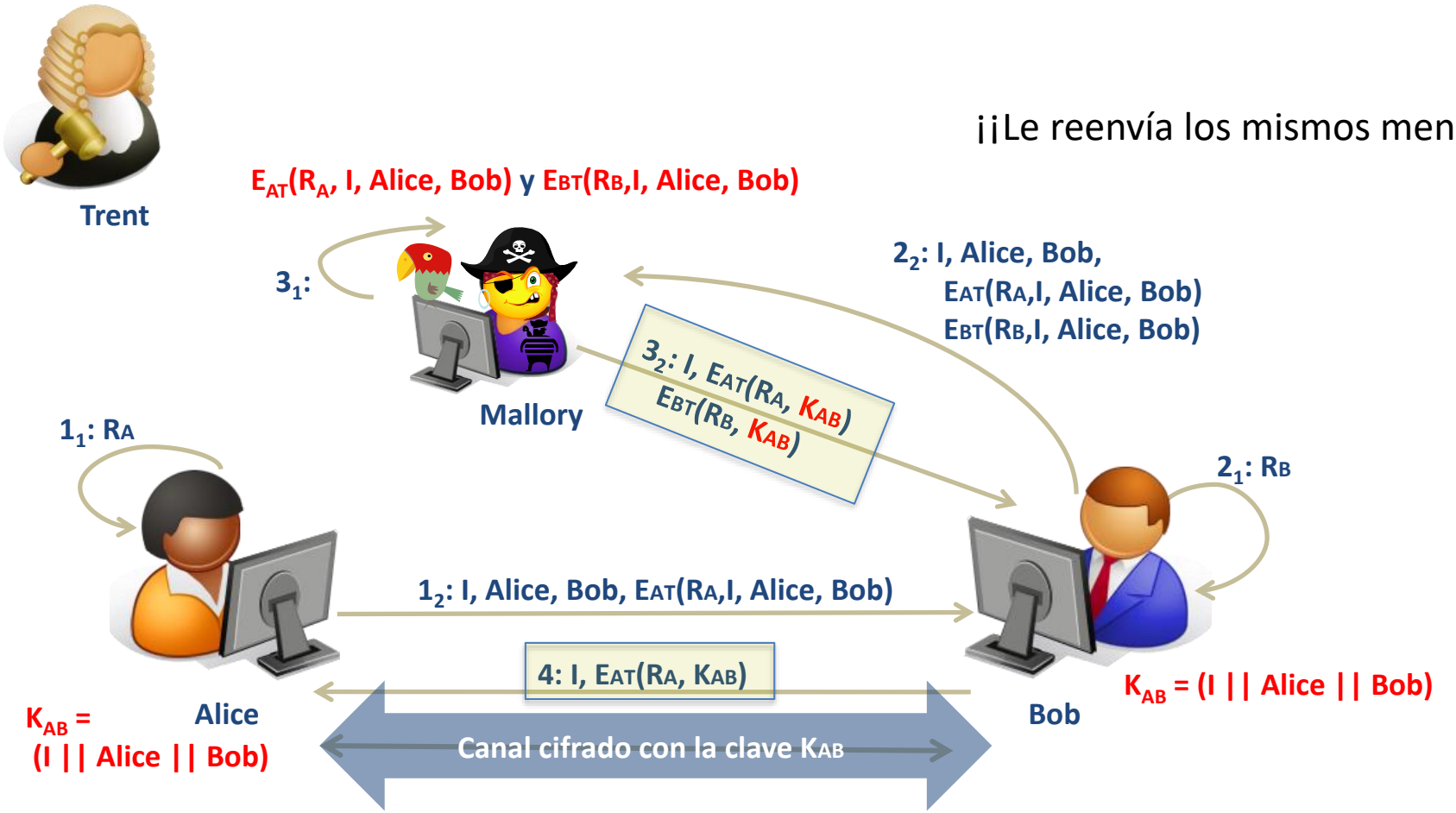
Otway-Rees

- Hay otro agujero de seguridad…



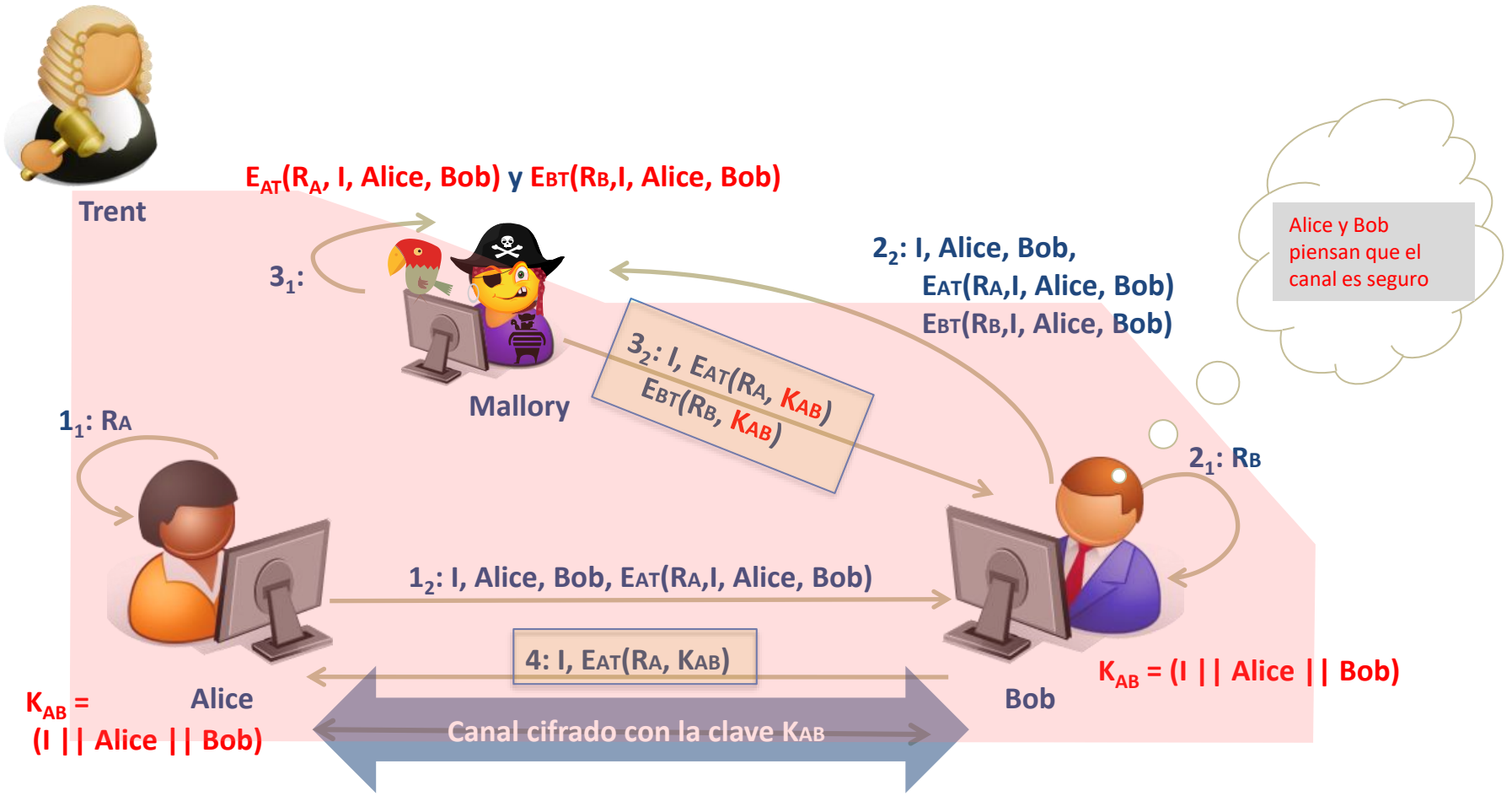
10 minutos – tic, tac, tic, tac …

– **Segundo agujero de seguridad**:



¡¡Le reenvía los mismos mensajes!!

– **Segundo agujero de seguridad**:



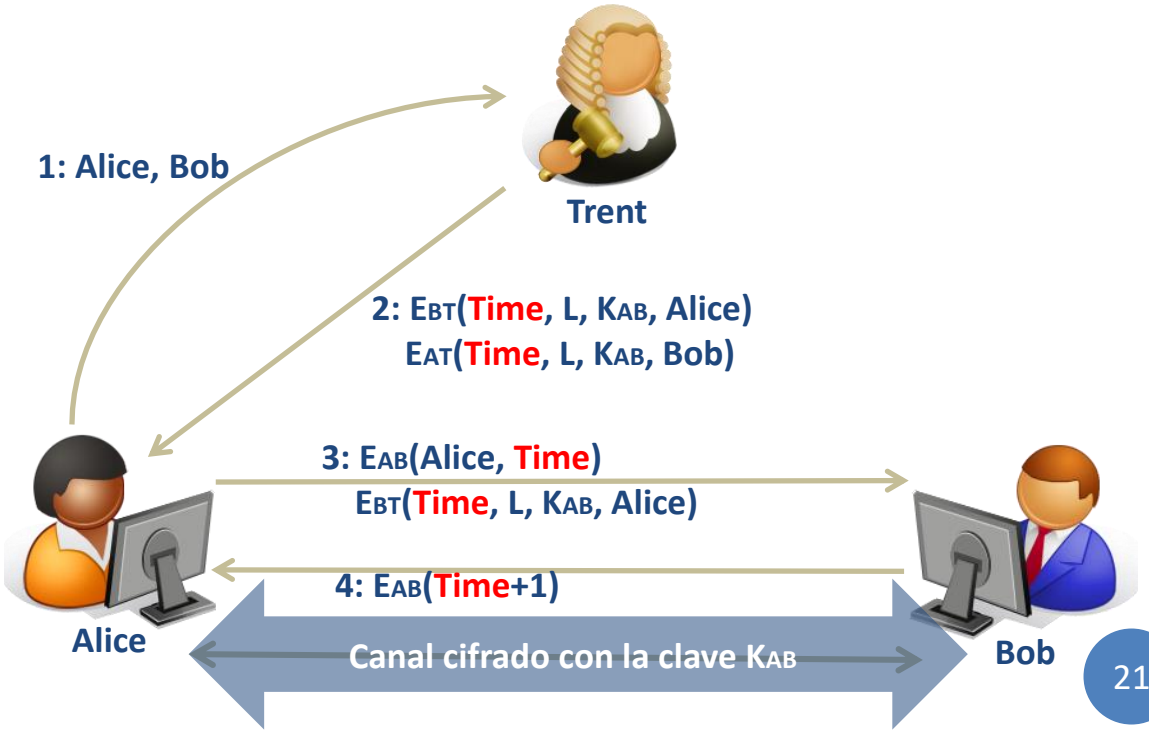
- Protocolo **Kerberos**

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*.

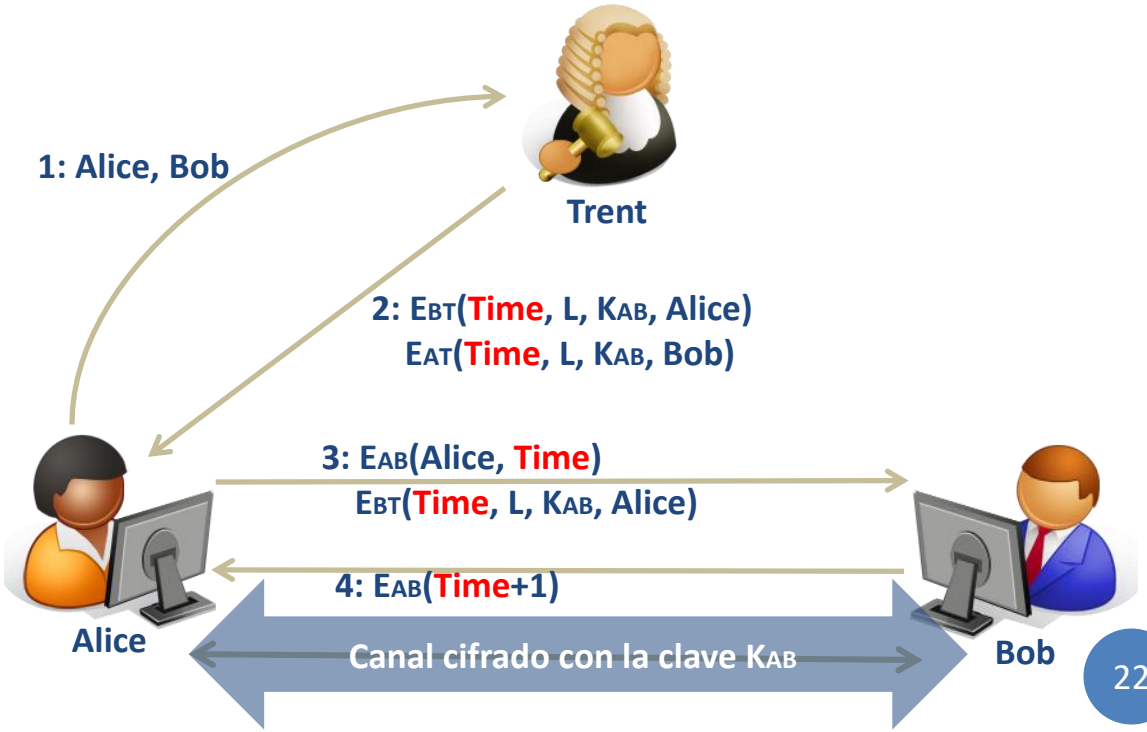
3: *Alice* obtiene K_{AB}, genera un mensaje con su identidad y el timestamp, y lo cifra con K_{AB} para enviárselo a *Bob*. *Alice* también envía a *Bob* el mensaje cifrado que recibió de *Trent*.

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 con el reloj de *Trent*.
En la práctica se sincronizan en el rango de unos pocos minutos.

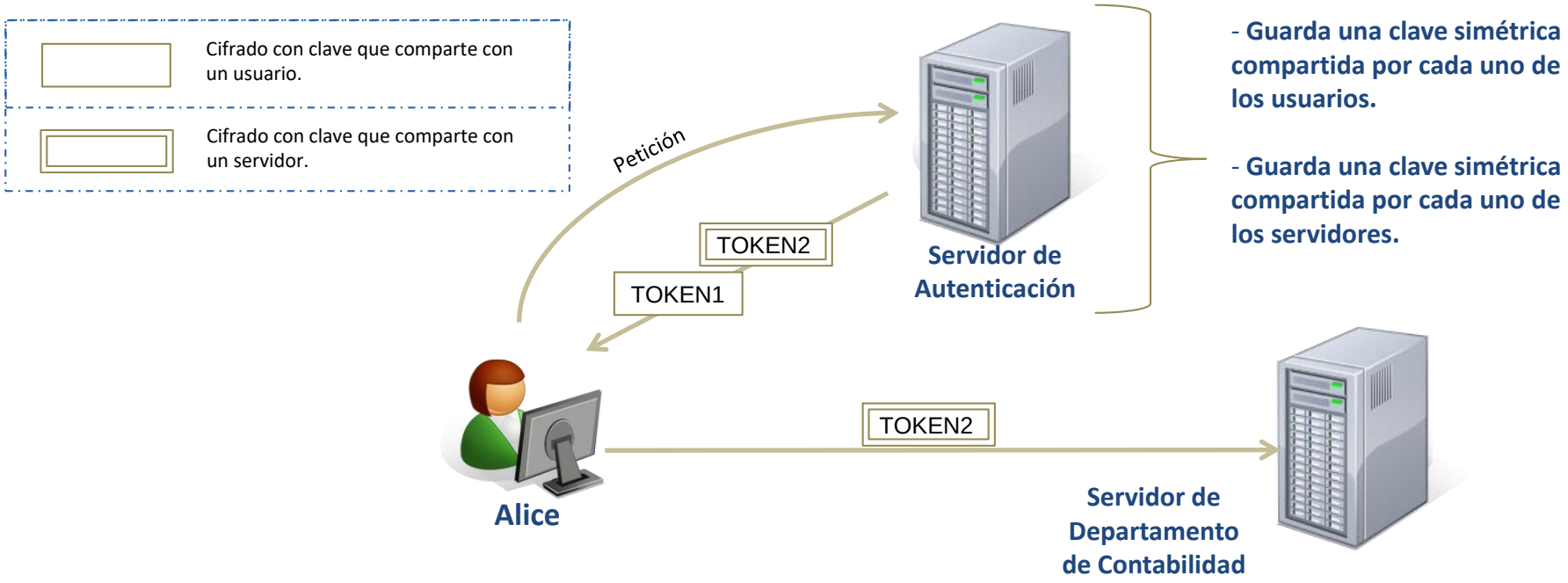
- Por fallos del sistema o por sabotaje, los relojes pueden desincronizarse (➔ ataque)

¿Dónde podría usarse?...

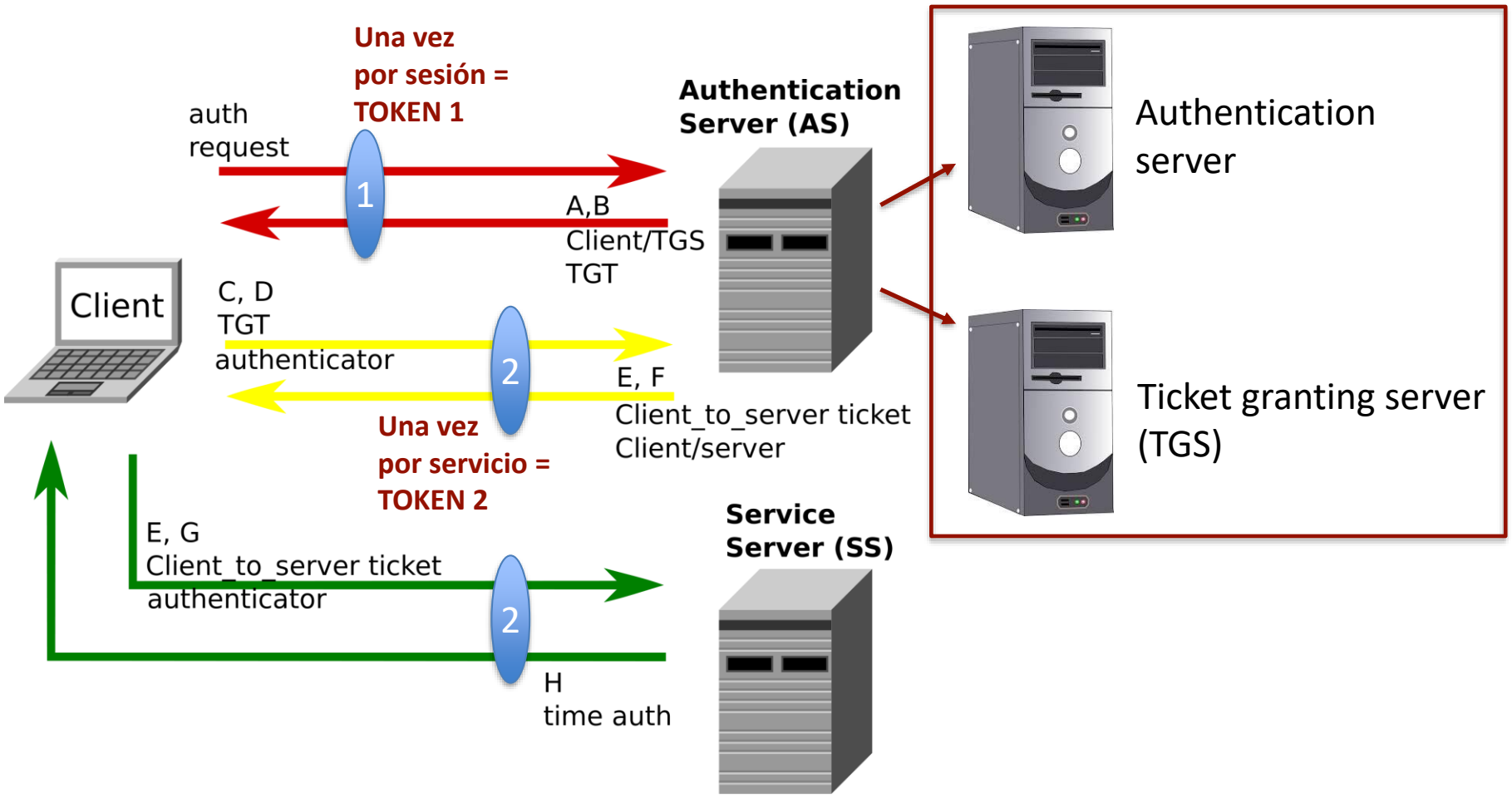


Kerberos

- 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
    - 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 estos protocolos, hay muchos otros que combinan múltiples tipos de estrategias (a nivel de modelos como de mecanismos de seguridad):

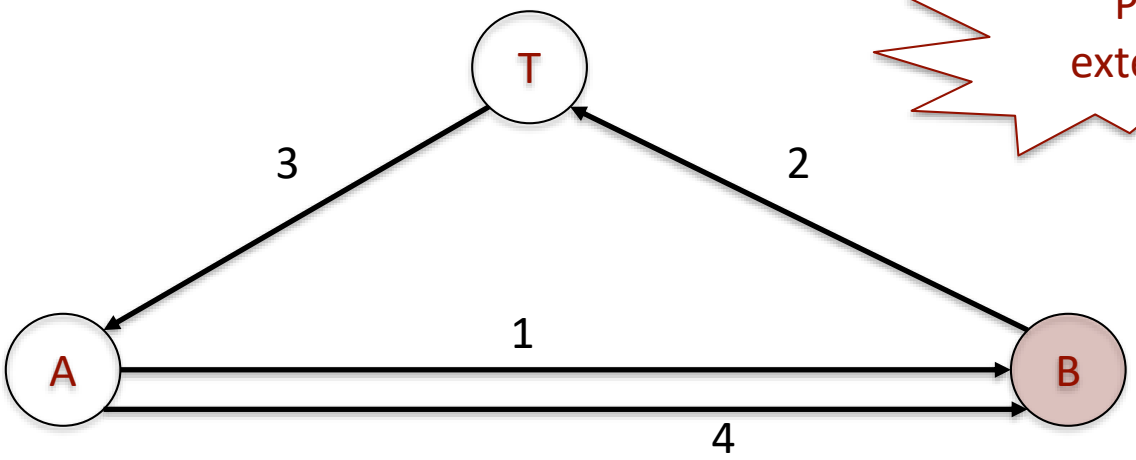
  - **Yahalom**
    - Objetivo: consiste que Trent genere la clave de sesión K_{AB} y lo envía directamente a Alice e indirectamente a Bob

$A \rightarrow B : A, N_a$

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

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

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



PUSH extendido

– **Neuman Stubblebine**

- Objetivo: consiste en combinar múltiples formas de verificar la autenticidad de las transacciones – time-stamps, nonces

$A \rightarrow B : A, N_a$

$B \rightarrow T : B, E_{KBT}\{A, N_a, \text{time-stamp}_b\}, N_b$

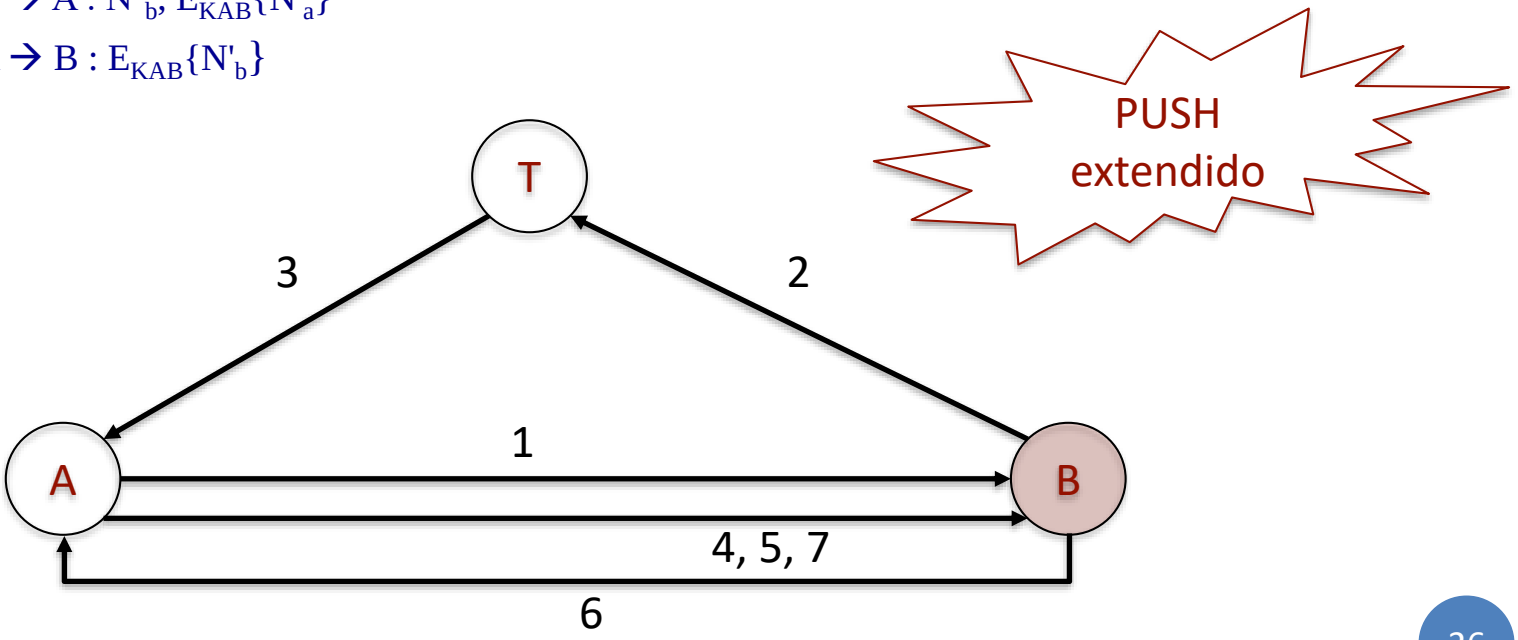
$T \rightarrow A : E_{KAT}\{B, K_{AB}, N_a, \text{time-stamp}_b\}, E_{KBT}\{A, K_{AB}, \text{time-stamp}_b\}, N_b$

$A \rightarrow B : E_{KBT}\{A, K_{AB}, \text{time-stamp}_b\}, E_{KAB}\{N_b\}$

$A \rightarrow B : N'_a, E_{KBT}\{A, K_{AB}, \text{time-stamp}_b\}$

$B \rightarrow A : N'_b, E_{KAB}\{N'_a\}$

$A \rightarrow B : E_{KAB}\{N'_b\}$



– **Kao Chow**

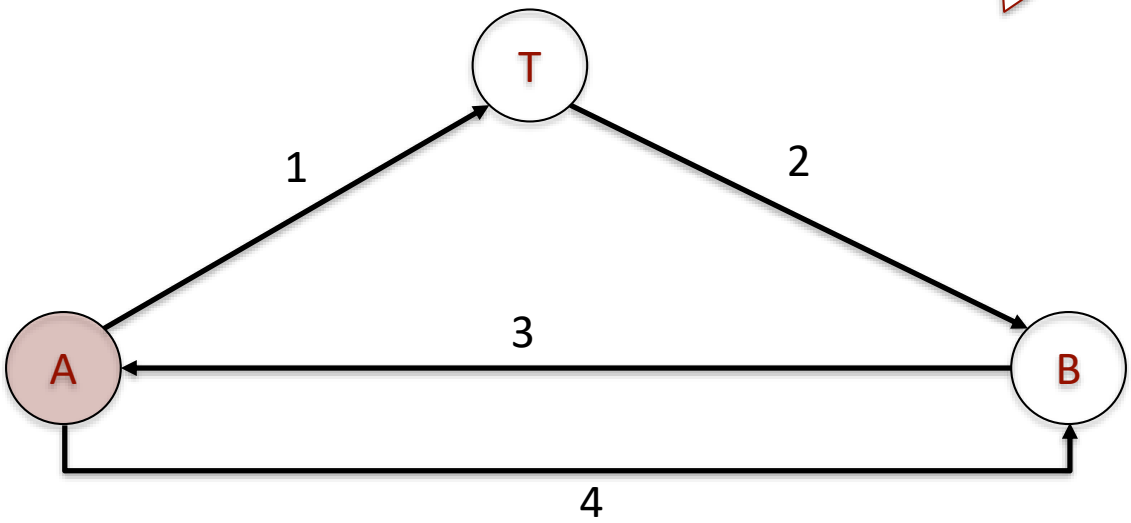
$A \rightarrow T : A, B, N_a$

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

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

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

PULL extendido



- Hemos visto que existen distintos protocolos para solucionar el mismo tipo de problema de administración/intercambio de claves
- El protocolo a elegir depende bastante de la arquitectura de comunicaciones subyacente, o lo que es lo mismo, de las respuestas a preguntas como:
  - ¿Es necesario minimizar el tamaño de los mensajes?
  - ¿Es necesario minimizar el número de mensajes?
  - ¿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?
  - etc.

- 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;

1. el KDC representa un **único punto de fallos** (o ataques)
   - si queda inutilizado nadie puede establecer comunicaciones seguras dentro de la red;

2. 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.

- Caso real de uso por desarrolladores en Information Security Stack Exchange Q&A:
  - Question: Today I was reading notes about cryptography and I came across a problem that exists in Symmetric Key encryption: How to share the secret key across the network
    - 1st Method: Make use of the trusted KDC (Key Distribution Center)
    - 2nd Method: Encrypt the key using Public Key technique and exchange the key (e.g., SSL)

    My question is:
    - which technique is prove to be more effective?
    - Why has SSL implemented method 2? Is it because it's more secure?

Which technique is prove to be more effective?

- Answer 1:

  KDC is suitable for smaller infrastructures where you place explicit trust into each person or node doing encryption. Each time Alice wants to encrypt a message to Bob, she has to ask KDC for a temporary key to use for encryption. The KDC will also need to provide that temporary key to Bob so he can decrypt the message. You don't want to just give Alice Bob's encryption key, as then she would be able to read all encrypted messages sent to Bob (the nature of symmetric encryption is such that the same key both encrypts and decrypts the message).

  As you can imagine, this does not scale beyond local infrastructures, nor is very fault-tolerant. If the Internet relied on KDCs in order to do encryption, it would be easy for attackers to DoS them and kill all commerce on the web -- or at least make it unreliable enough not to bother. This is why browsers went with Public Key cryptography and a framework of mutually trusted certificate authorities (CAs). This has its own set of trade-offs. One, you have to trust that the CAs know what they are doing -- a trust they have repeatedly violated. Two, PKI relies on the hope that we'll never find a fast way to factor the product of two very large prime numbers -- a problem not present in symmetric cryptography. If the likes of Grigory Perelman one day find a way to quickly solve that problem, anyone will be able to obtain the private decryption key from the public encryption key. If that happens, we'll be all in big trouble and will have to come up with some other way for two untrusted parties to exchange encryption keys.

  So, to answer your question:

  - **cryptographically, KDCs and symmetric keys are stronger than asymmetric public-private keys, as there is no danger that one day some crazy mathematician will find a way to quickly factor products of large primes**.
  - On the other hand, **KDCs have inherent problems with key distribution, reliability and ongoing trust** that can't be easily solved and therefore KDCs **are not suitable beyond local installations where such trust is easy to assure**.

Why has SSL implemented method 2? Is it because it's more secure?

- Answer 2:

  A key distribution centre is a central system which distributes the keys to the user. Its **central nature** implies that:

  – Everybody talks to the KDC, so the KDC is easily overwhelmed in big networks.

  – The KDC has the power to betray everybody in the system.

  As such, a KDC does not scale well, and requires an existing, ***online* infrastructure**.

  On the other hand, **SSL uses asymmetric cryptography and public key distribution with certificates**; even there, there *are* entities who can betray many people (the *certification authorities*) but at least they can operate offline, which avoids scalability issues and makes them easier to protect against external subversion.

  Thus, SSL uses asymmetric cryptography **because it is easier to apply generically (especially worldwide), and also easier to keep secure**.