User Authentication Protocols

Week 5

User Authentication

- *The process of verifying an identity claimed by a system entity*
- Fundamental system security building block
  - Basis of access control & user accountability
- Has two steps:
  - Identification – provide claimed identity
  - Authentication – verify validity of claim
- *User authentication ≠ message authentication*

User Authentication: How ?

- Based on something the individual
  - Knows - e.g. password, PIN
  - Possesses - e.g. key, token, smartcard
  - Is (static biometrics): fingerprint, retina
  - Does (dynamic biometrics): voice, handwriting
- Can use alone or combined
- All can provide user authentication
- *All have issues*

Authentication Protocols

- Convince parties of each others identity
  - Also exchange session keys
- May be one-way or two-way (mutual)

Key issues:

1. Confidentiality
   - Protect session keys
   - Prior keys or secrets need to exist
2. Timeliness
   - Prevent replay attacks

Replay Attacks

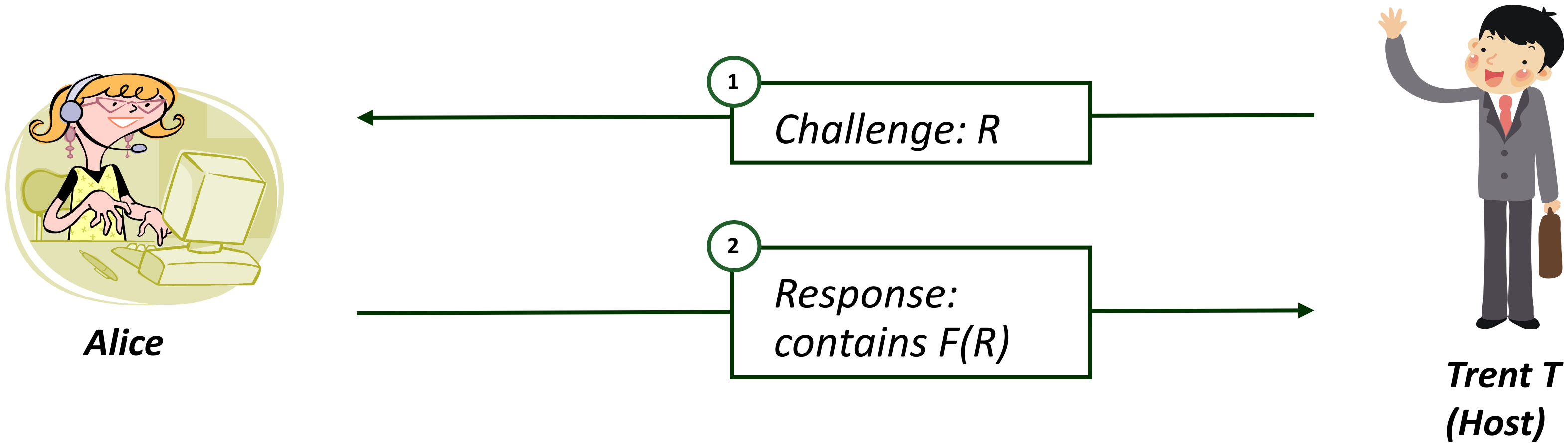
- Valid signed message is copied and later re-sent
- Simple replay
  - Copy message; replay later
- Repetition that can be logged
  - Replay timestamped message within validity interval
- Repetition that cannot be detected
  - Suppress original message
- Backward replay without modification
  - Send the replay message back to its sender

Replay Attacks: Countermeasures

- Sequence numbers
  - Attach sequence number *seqno* to message
  - Accept message if *seqno* follows previous value
  - Not always practical
- Timestamps
  - Message needs to contain *timestamp*
  - Accept message if timestamp is within validity window
  - Need synchronized clocks

Countermeasures (cont'd)

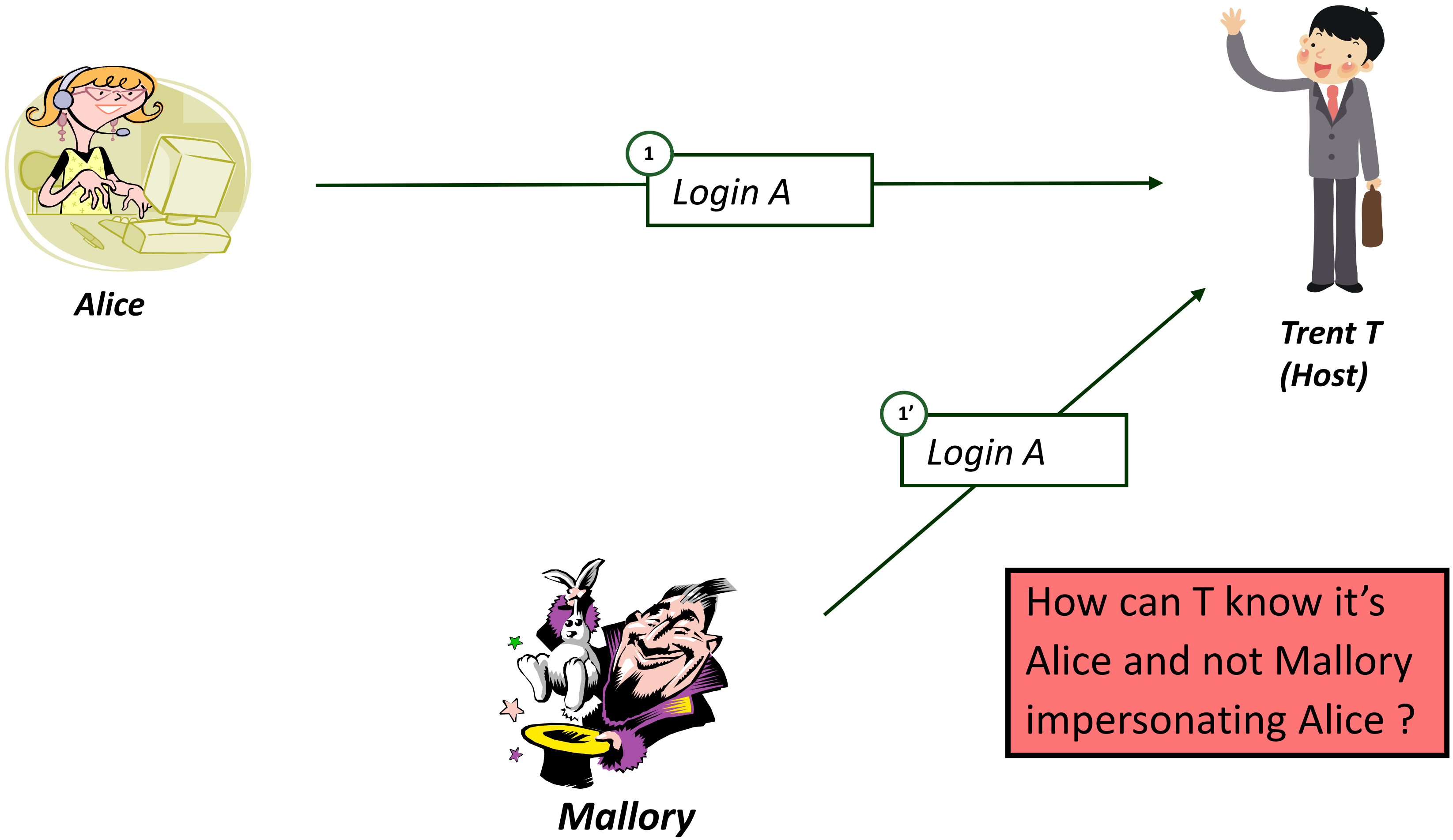
- Challenge/response
  - Ensures message *freshness*
  - Challenger sends random nonce R
  - Responder's message needs contain a function of R



Authentication

- One-way authentication
- Mutual: two-way authentication
  - Using symmetric key crypto
  - Using public-key crypto

One-Way Authentication



Authentication Approaches

- Password
  - Host stores Alice’s password
  - Alice sends password
  - Host verifies password
- Problem:
  - Trent stores all passwords in clear
  - Whoever breaks into Trent can steal passwords
- Solutions
  - One-Way Functions
  - Dictionary Attacks and Salts

Authentication Using Hashes

- Roger Needham and Mike Guy
  - T does not need to know password
  - *Only differentiate between valid and invalid ones*

Alice

1 *Login A, pwd*

Problem ?

Password file

User ID	H(pwd)
Alice	H_A

Trent T (Host)

T: Compare H(pwd) to H_A

Password Vulnerabilities

- One-way hashes are vulnerable
- *Which password is better ?*
  - Barney
  - 9(hH/A.
- Which one is easier to remember ?
- Dictionary attack
  - Compile list of most probable passwords
  - Apply hash function to each
  - Compare against the password file
  - *If match, password has been found !*

Defending with Salts!

Salt: per user random value

Alice

1 *Login A, pwd*

Trent T (Host)

Password file

User ID	salt	H(salt, pwd)
Alice	s	H_A

$H(s, pwd) == H_A$

Example: Linux

- Passwords stored in /etc/shadow
  - Root readable only
- carbunar:6lGHQQKZn$8.eJLvAaJiDTFAauGVbFlmn AcjIKyLtH6GiO0mVgra8weKJ1igU2BmgdDQAalynFQ0 QuezQr7mDTWEPD7sDrW
- $6: hash algorithm
  - $1 = MD5 hashing algorithm.
  - $2 =Blowfish Algorithm is in use.
  - $2a=eksblowfish Algorithm
  - $5 =SHA-256 Algorithm
  - $6 =SHA-512 Algorithm

Example: Linux

- Passwords stored in /etc/shadow
  - Root readable only
- carbunar:6lGHQQKZn$8.eJLvAaJiDTFAauGVbFlmn AcjIKyLtH6GiO0mVgra8weKJ1igU2BmgdDQAalynFQ0 QuezQr7mDTWEPD7sDrW
- salt
- hash

The Goal of Salts

- Ensure that attacker cannot use the same dictionary to break all passwords

- Instead, attacker has to do a per-user dictionary + computation …

Improved Dictionary Attack [D. Klein]

1. Copy the password file
2. For each user A with salt s and hash H_A
   1. Collect dictionary D_A of tentative passwords
   2. Hash all items in D_A using salt s
   3. Compare result against H_A
3. *If match exists, found password*

- 40% of passwords were guessed on average system !

Building the Dictionary

1. Name, initials, account name
   - Example: Daniel V. Klein, account – klone
   - klone0, klone1, …, dvk, dklein, DKlein, dvklein, etc
2. Words from databases
   - Men and women names, nicknames (also famous)
   - Places
   - Variations of the above (capitalizations, plurals, etc)
3. Foreign language words
4. Word pairs

Conclusions

- Never use your personal information
- Do not use words (dictionary)
- Use combination of words and characters
- Do not use same passwords for all systems
- Change your password frequently
- Use passphrases
- Example:
  - ”My Password is not easy to crack”
  - mpine2C.

SKEY: Authentication for Machines

Use hash-chains

Alice

Trent T
(Host)

① *Init, A, x_{100}* →

② *Login, A, x_{99}* →

③ *Login, A, x_{98}* →

Generate R

Compute
$x_1 = H(R)$
$x_2 = H^2(R) = H(H(R))$
$x_3 = H^3(R) = H(H(H(R)))$
…
$x_{100} = H^{100}(R)$

Store x_{100}

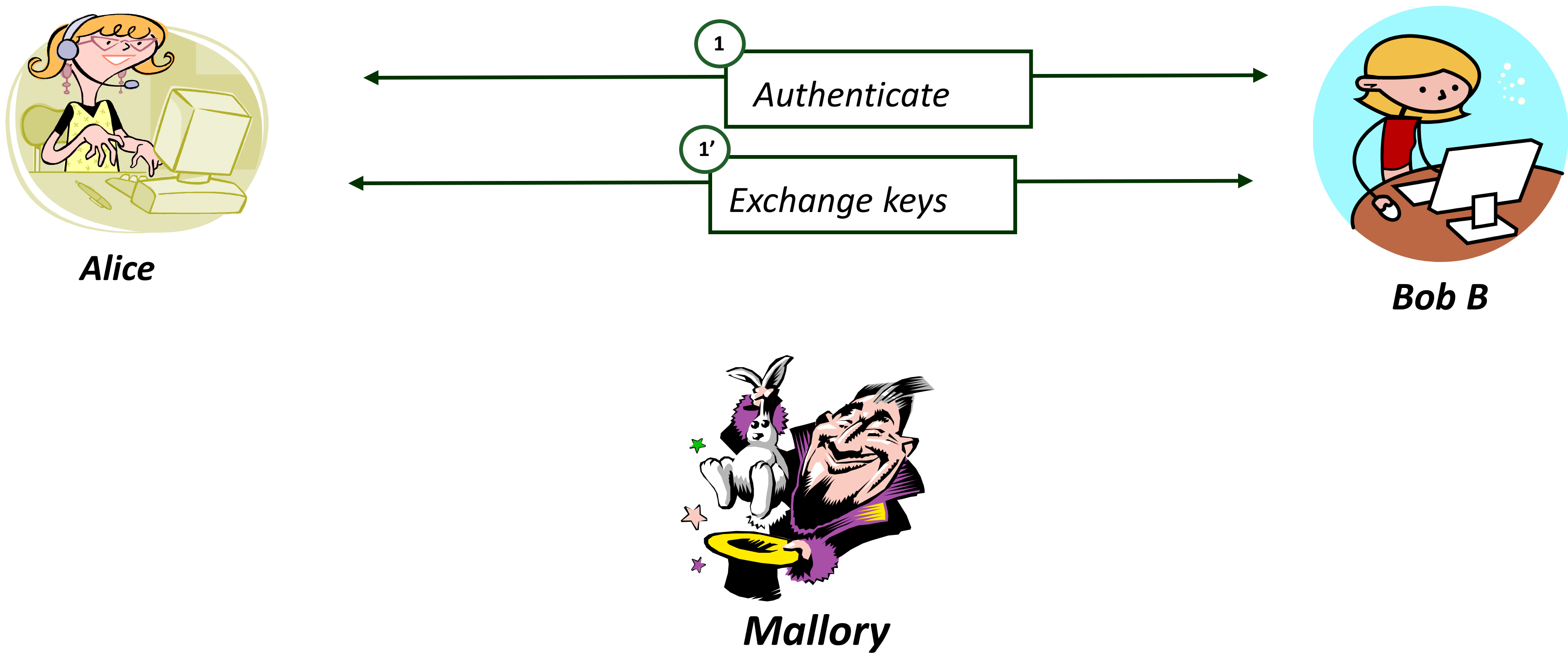
Compare
$H(x_{99})$ to x_{100}

Discard x_{100} Store x_{99}

Authentication

- One-way authentication
- Mutual: two-way authentication
  - Using symmetric key crypto
  - Using public-key crypto

What is Mutual Authentication ?

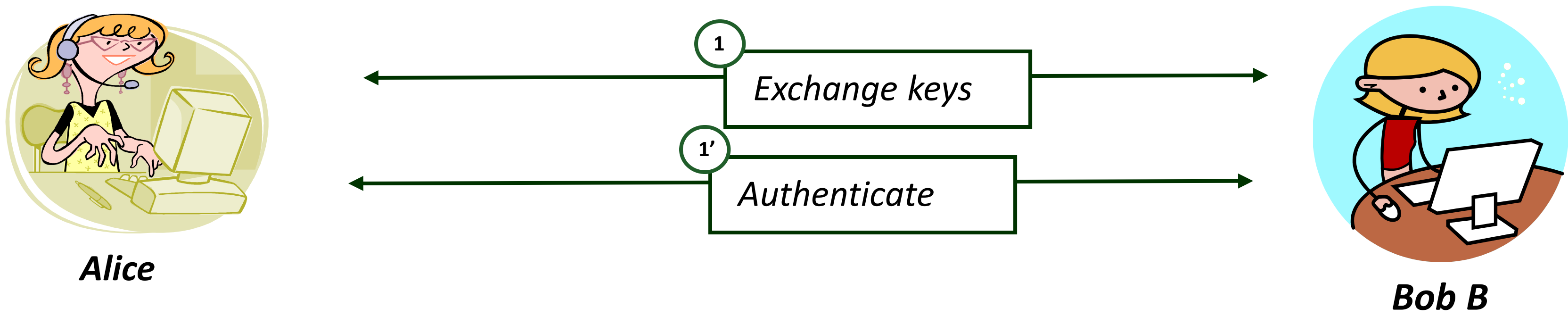


Make sure they don’t talk to Mallory !

Authentication

- One-way authentication
- Mutual: two-way authentication
  - Using symmetric key crypto
  - Using public-key crypto

Using Symmetric Keys



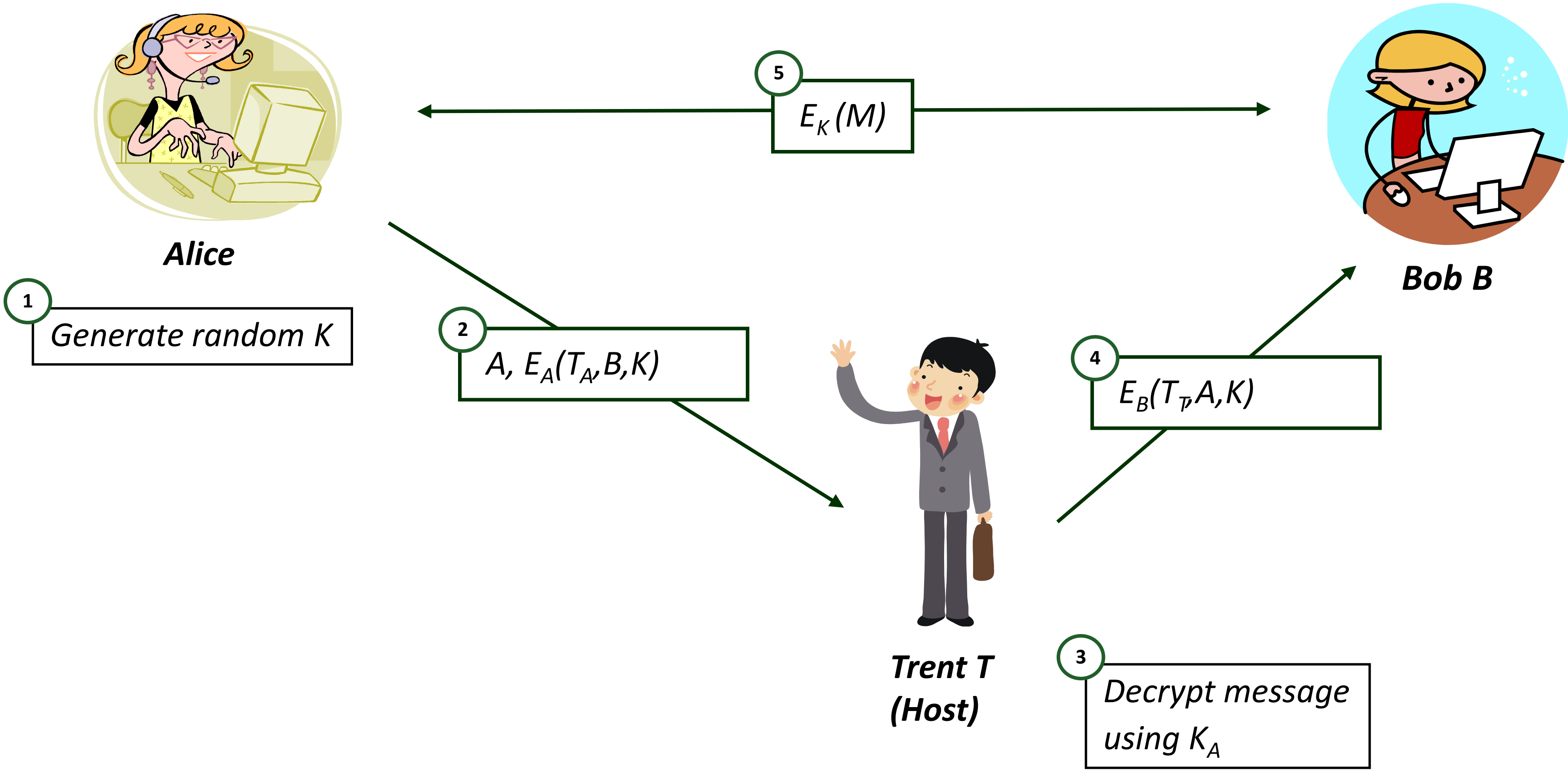
Trent T (Host)

Assume T shares a key with A (K_A) and B (K_B)

$E_A(M)$:encryption with key shared by A and T

Wide-Mouth Frog

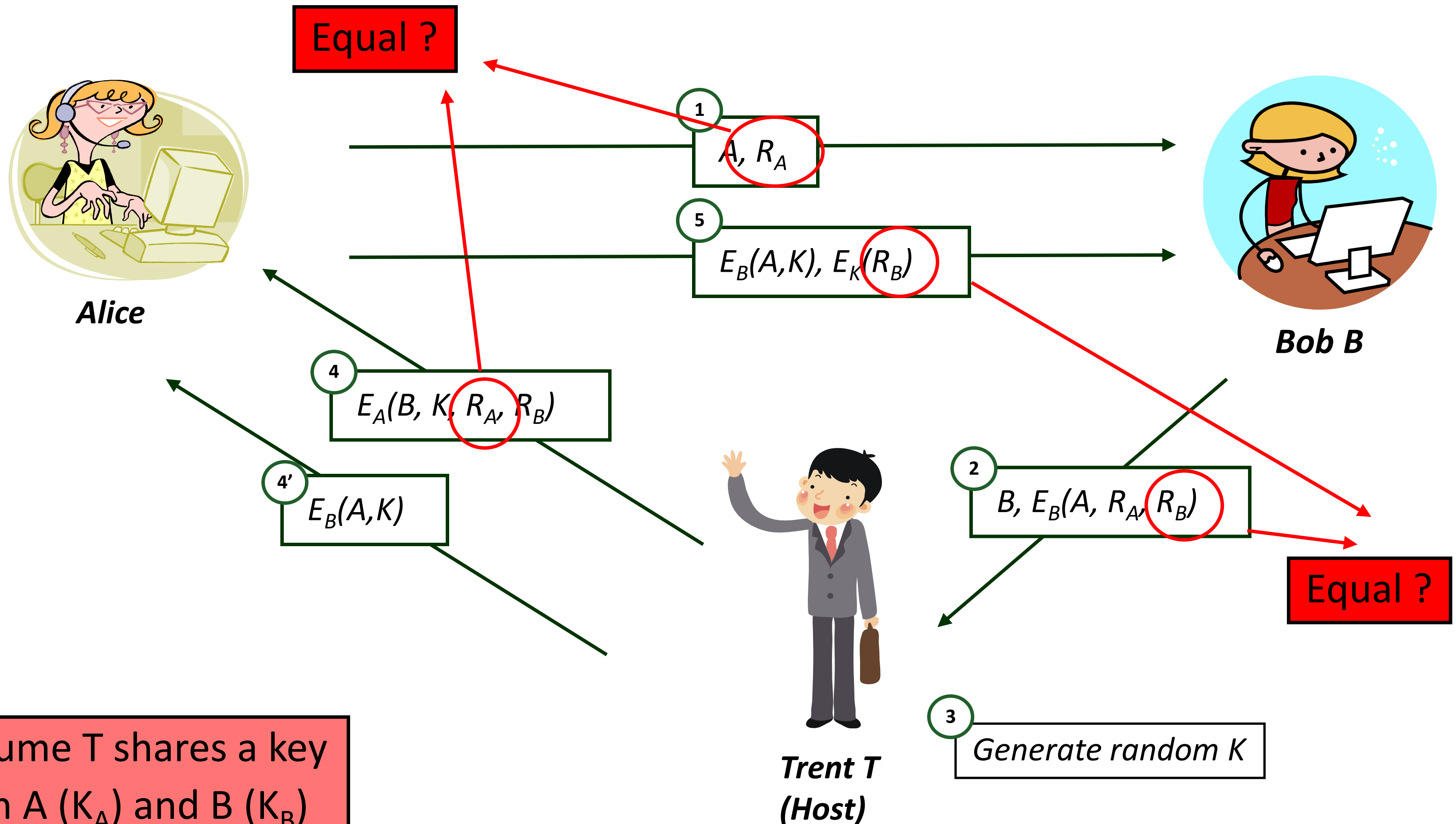
Simplest Authentication/Key Exchange



Wide-Mouth Frog Observations

- Alice and Bob trust each other because of Trent
- *Timestamps prevent replay attacks (Why ?)*
- Trent is single point of failure/bottleneck
- Assumption:
  - Alice is able to generate good random numbers

Yahalom

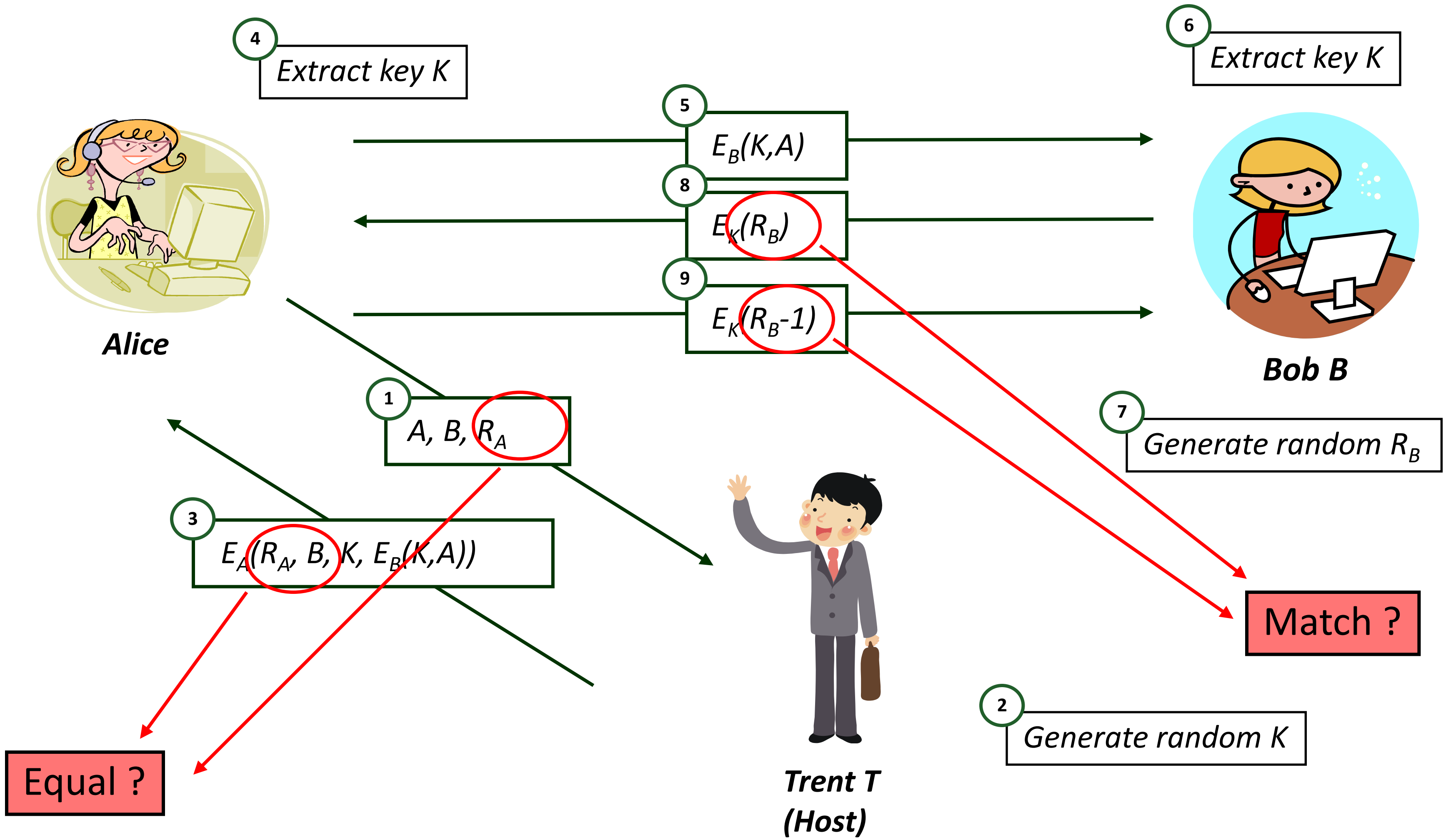


Assume T shares a key with A (K_A) and B (K_B)

Yahalom Observations

- This time the protocol is initiated by B (not T)
- T chooses the key K to be shared by A and B
- A and B trust each other
  - Because of R_A and R_B
  - Only T and B have access to R_B
- Problem in step 1 -- R_A is sent in clear
  - Can Mallory impersonate B ?
- No !
  - In step 4, T includes the identity of B - A will know who it is talking to

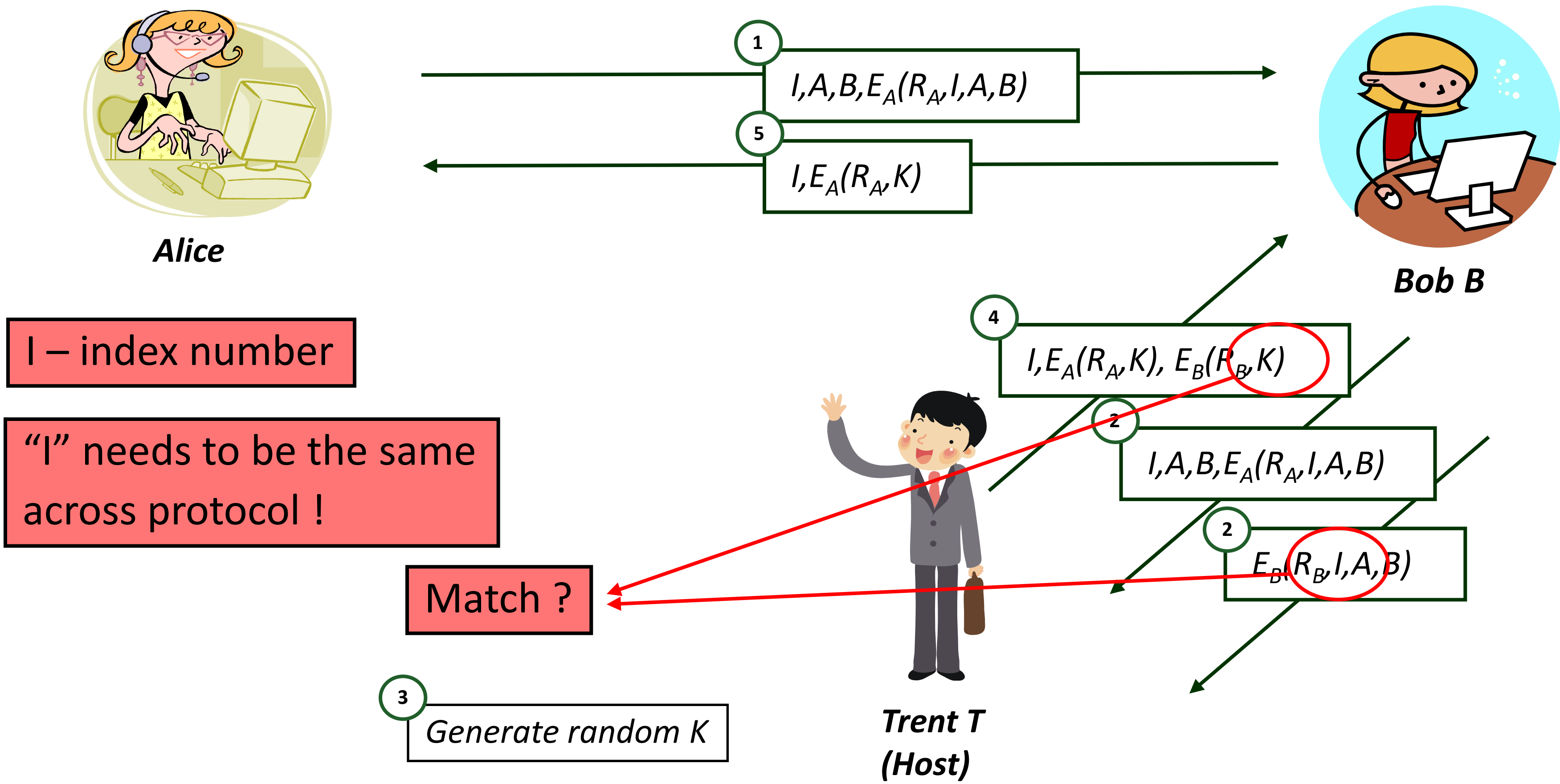
Needham-Schroeder



Needham-Schroeder Observations

- **What is the purpose of R_A ?**
  - For A to prevent replay attacks
  - Ensure it is talking to T
- **What is the purpose of R_B ?**
  - For B to prevent replay attacks
  - And ensure that it is talking to A
- Weakness
  - If Mallory gets hold of an old key K, it can impersonate A
- Solution: use timestamps

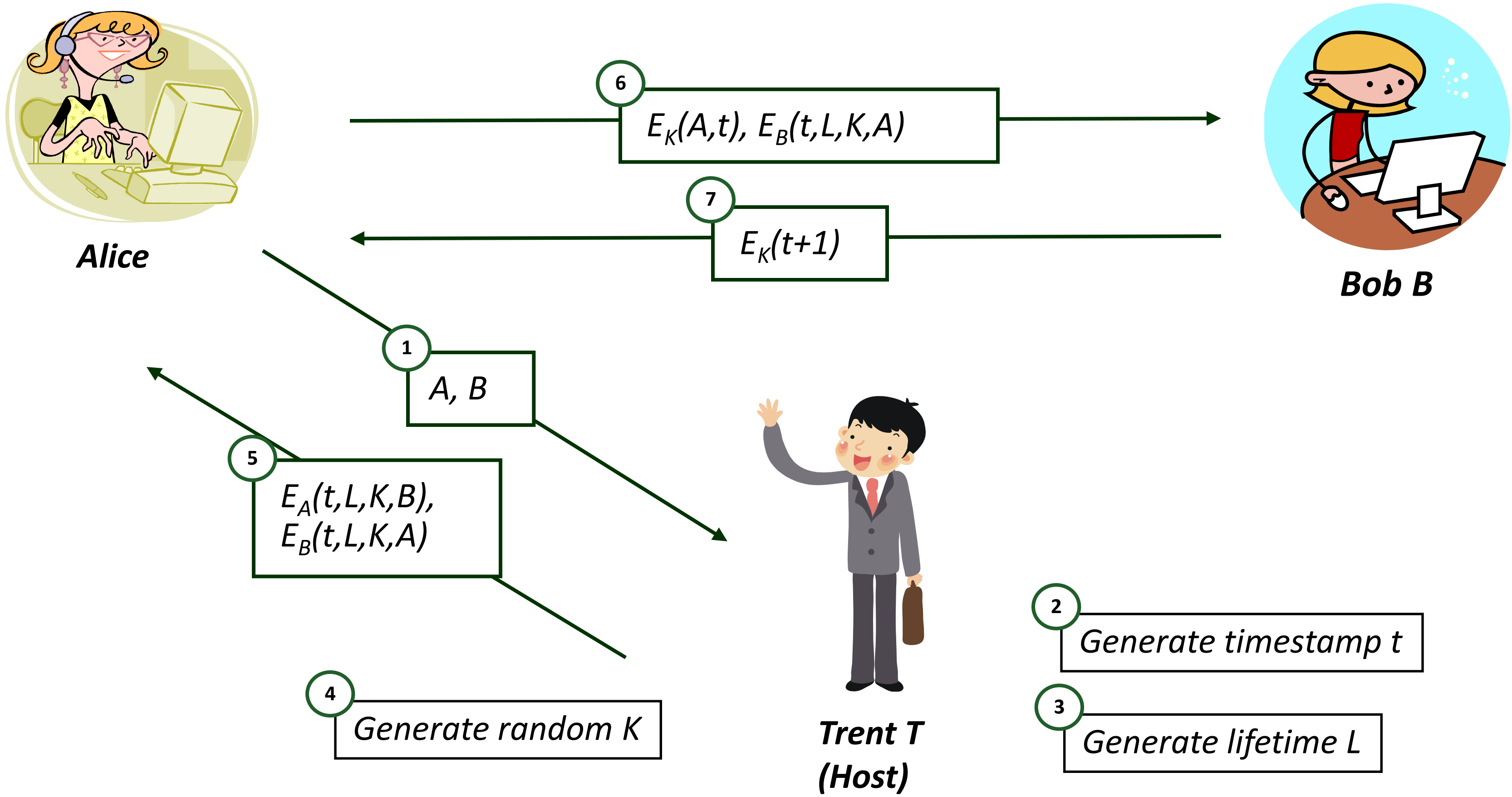
Otway-Rees



Alice
1
$I,A,B,E_A(R_A,I,A,B)$
5
$I,E_A(R_A,K)$
Bob B
I – index number
"I" needs to be the same across protocol !
4
$I,E_A(R_A,K), E_B(R_B,K)$
2
$I,A,B,E_A(R_A,I,A,B)$
2
$E_B(R_B,I,A,B)$
Match ?
3
Generate random K
Trent T
(Host)

Kerberos - Simplified

Kerberos 5: Variant of Needham-Schroeder



Kerberos Observations

- **What is the goal of the timestamp and lifetime ?**
  - To prevent replay attacks
  - The messages are valid only in [t,t+L]
- Major assumption:
  - The clocks are synchronized !
  - Not trivial (see Lamport's clocks)
- **In practice**
  - Use time servers
  - Sync within a few minutes

Authentication

- One-way authentication
- Mutual: two-way authentication
  - Using symmetric key crypto
  - Using public-key crypto

Authentication with Public Keys

Alice

Bob B

Trent T (Host)

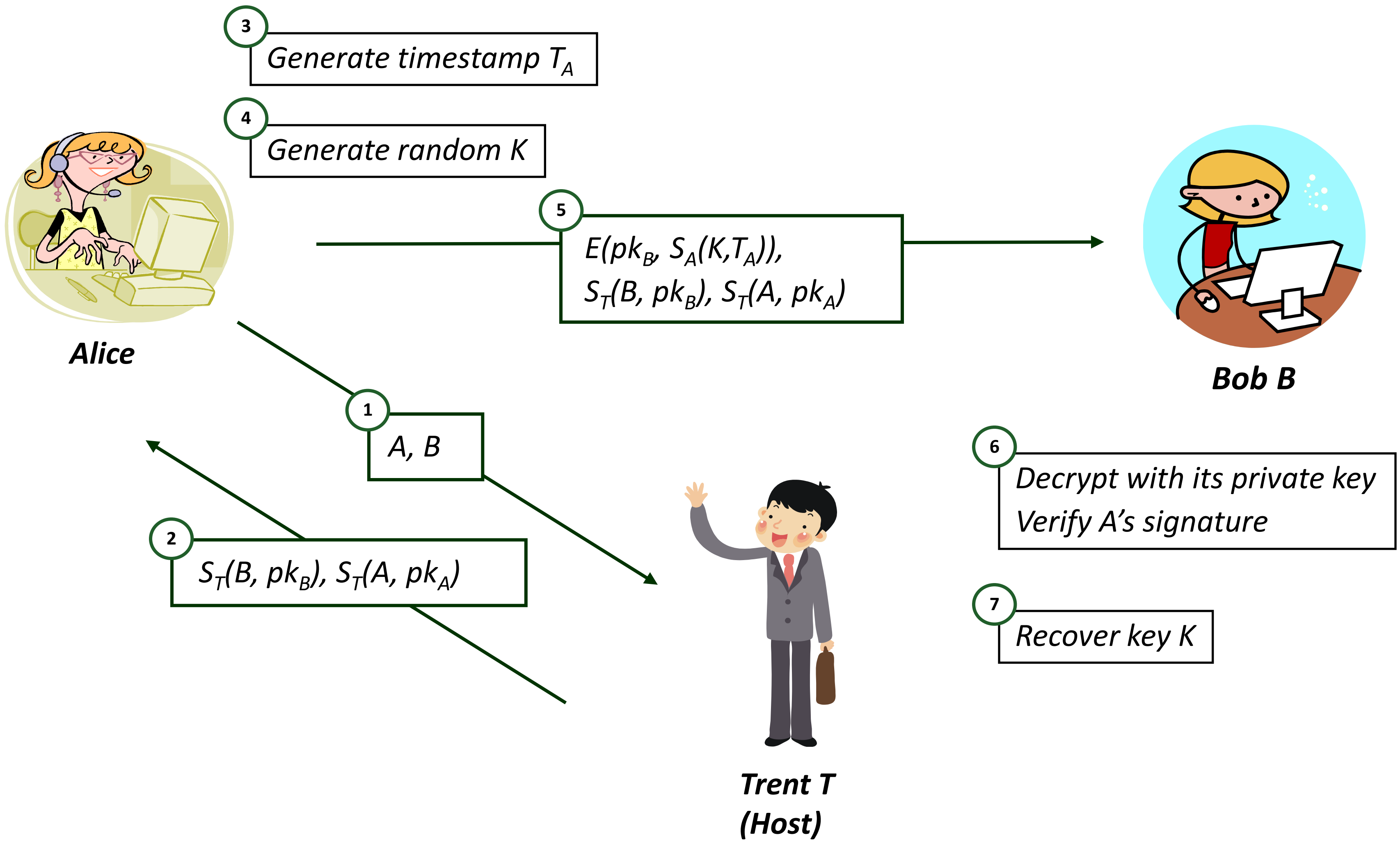
Assume T has a database of public keys for each participant

pK_A: A's public key

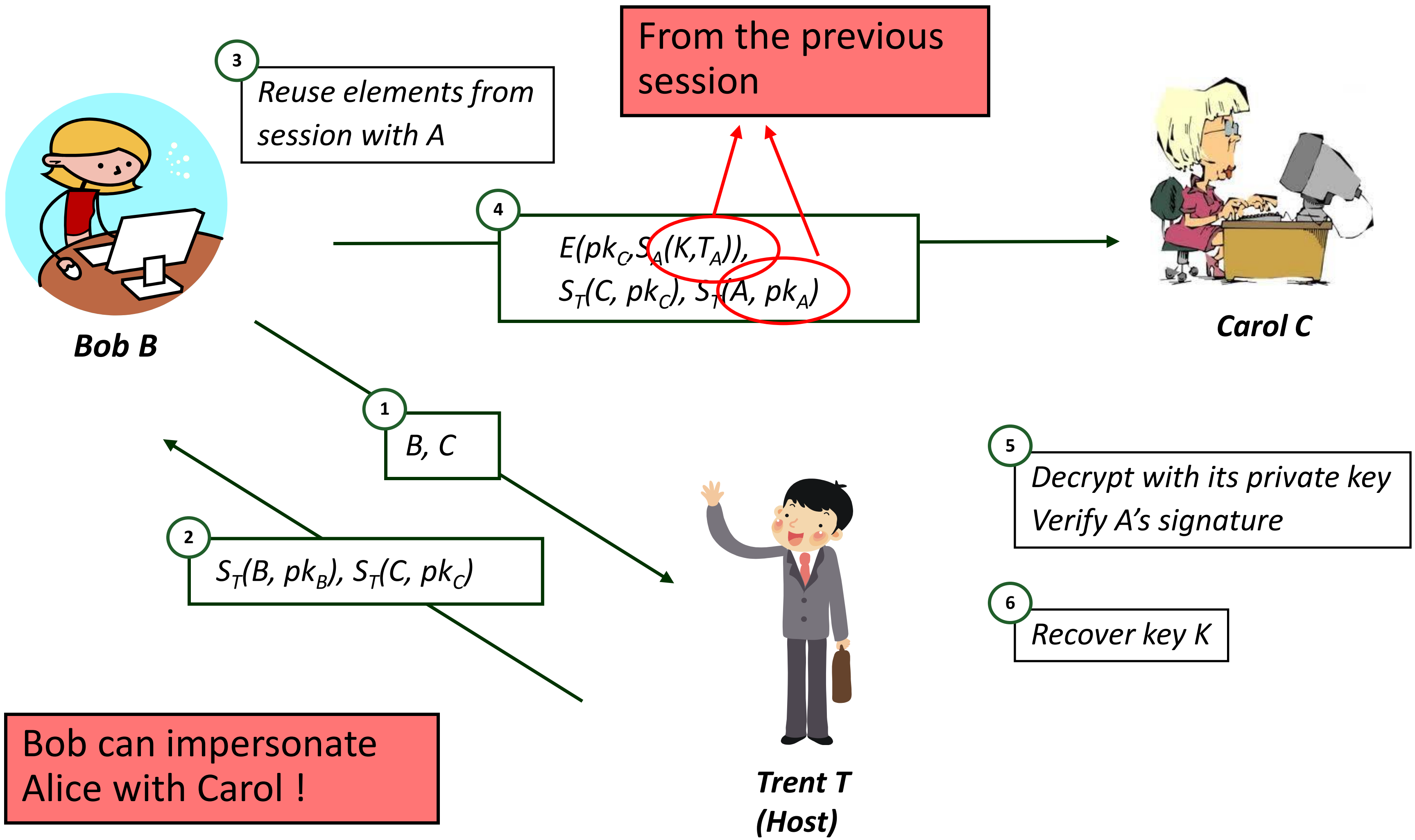
$E(pk_A, M)$: encryption with A's public key

$S_A(M)$: signature with A's private key

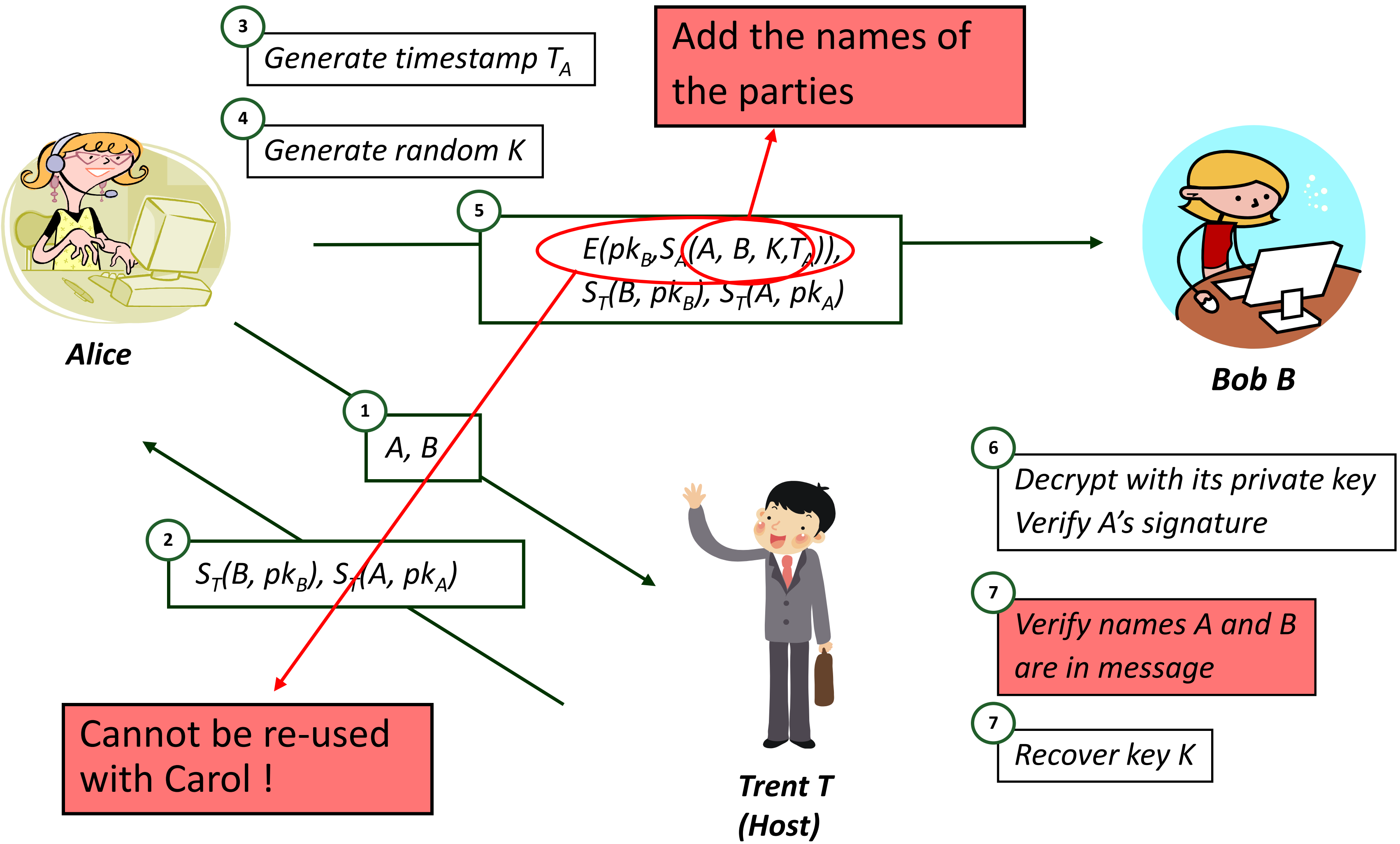
Denning-Sacco



Attacking Denning-Sacco !



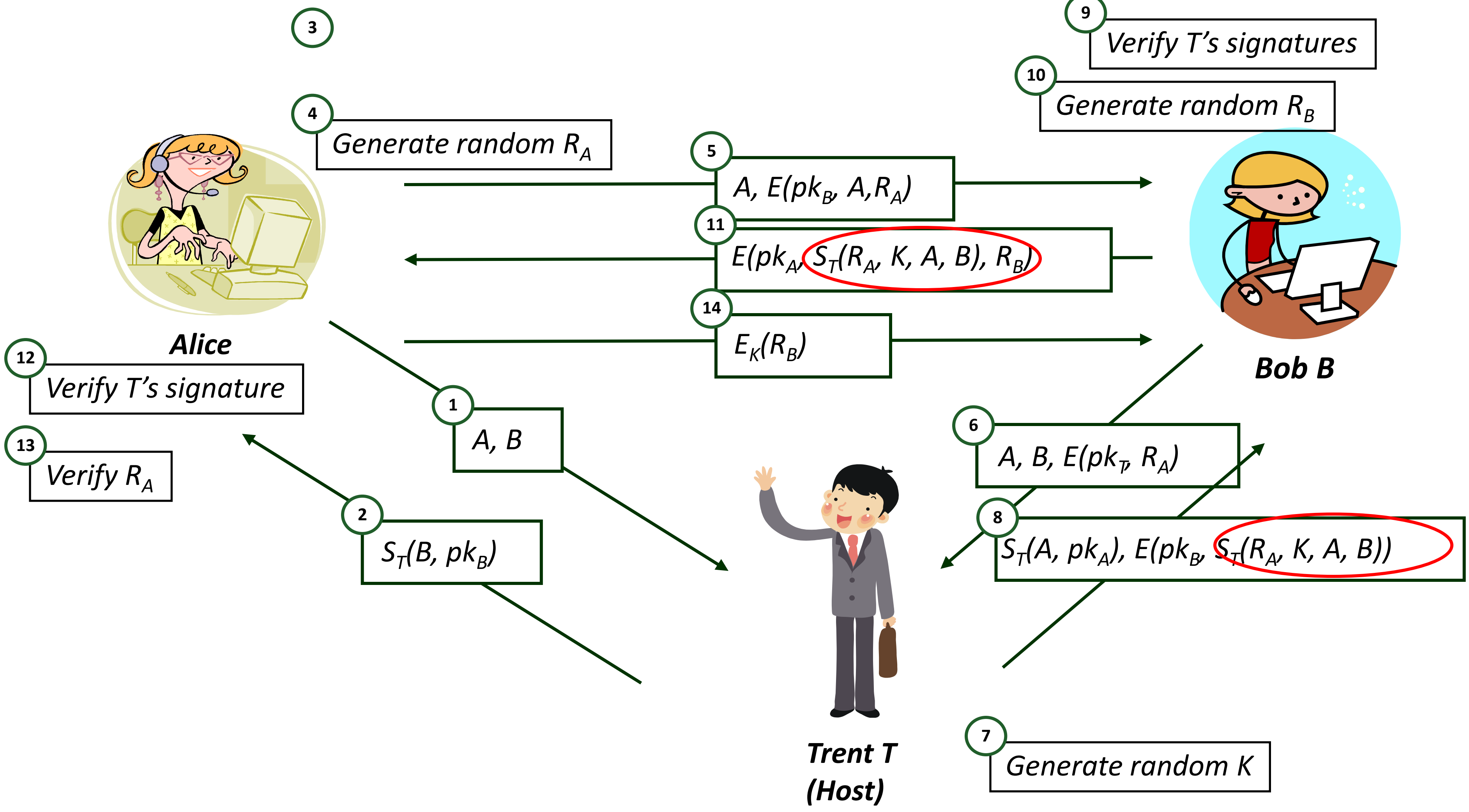
Denning-Sacco Fix



Denning-Sacco Lessons

- *Better be prudent than efficient*
- Include more rather than less information
- Timestamps, random nonces, names of participants

Woo-Lam



3
9
Verify T's signatures
10
Generate random R_B
4
Generate random R_A
5
$A, E(pk_B, A, R_A)$
11
$E(pk_A, S_T(R_A, K, A, B), R_B)$
14
$E_K(R_B)$
Alice
Bob B
12
Verify T's signature
13
Verify R_A
1
A, B
6
$A, B, E(pk_T, R_A)$
8
$S_T(A, pk_A), E(pk_B, S_T(R_A, K, A, B))$
2
$S_T(B, pk_B)$
7
Generate random K
Trent T
(Host)

Oauth 2.0

The Problems

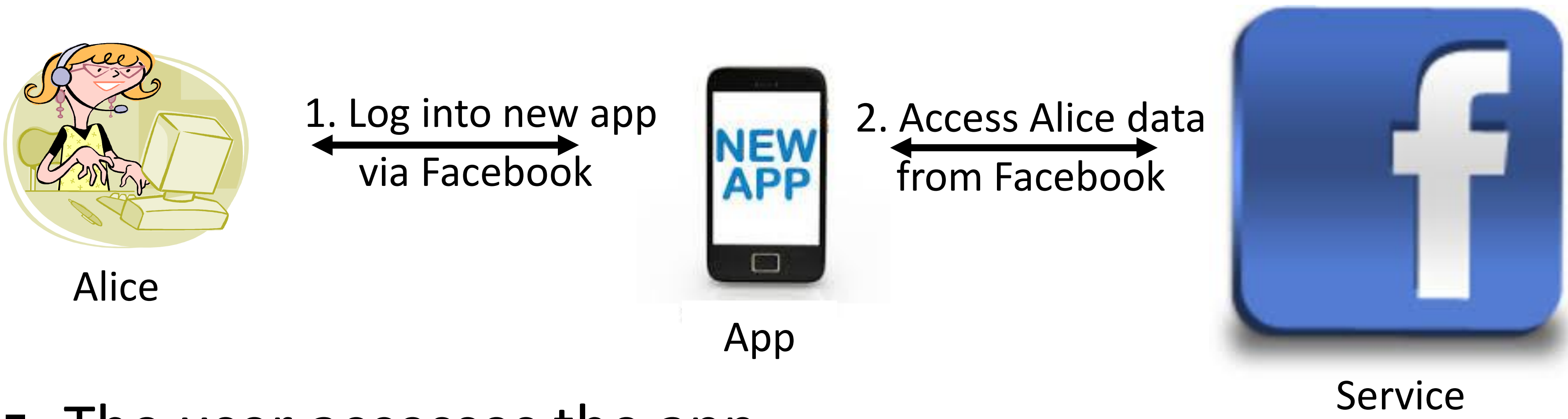
- User authentication is difficult
  - Passwords are hard to remember
  - Many of them, for many sites and apps
- Users cannot port their data from a site to another
- Examples:
  - Game would like to access user’s data from Facebook
  - Location based app would like to access user’s data from Foursquare application

OAuth 2.0

- Open authorization protocol
- Enable apps and websites to authenticate users with their credentials for other trusted sites (Facebook, Twitter …)
- Enables apps to access the user data of other systems
- Enable apps to call functions of other systems
  - Post in Facebook, Twitter

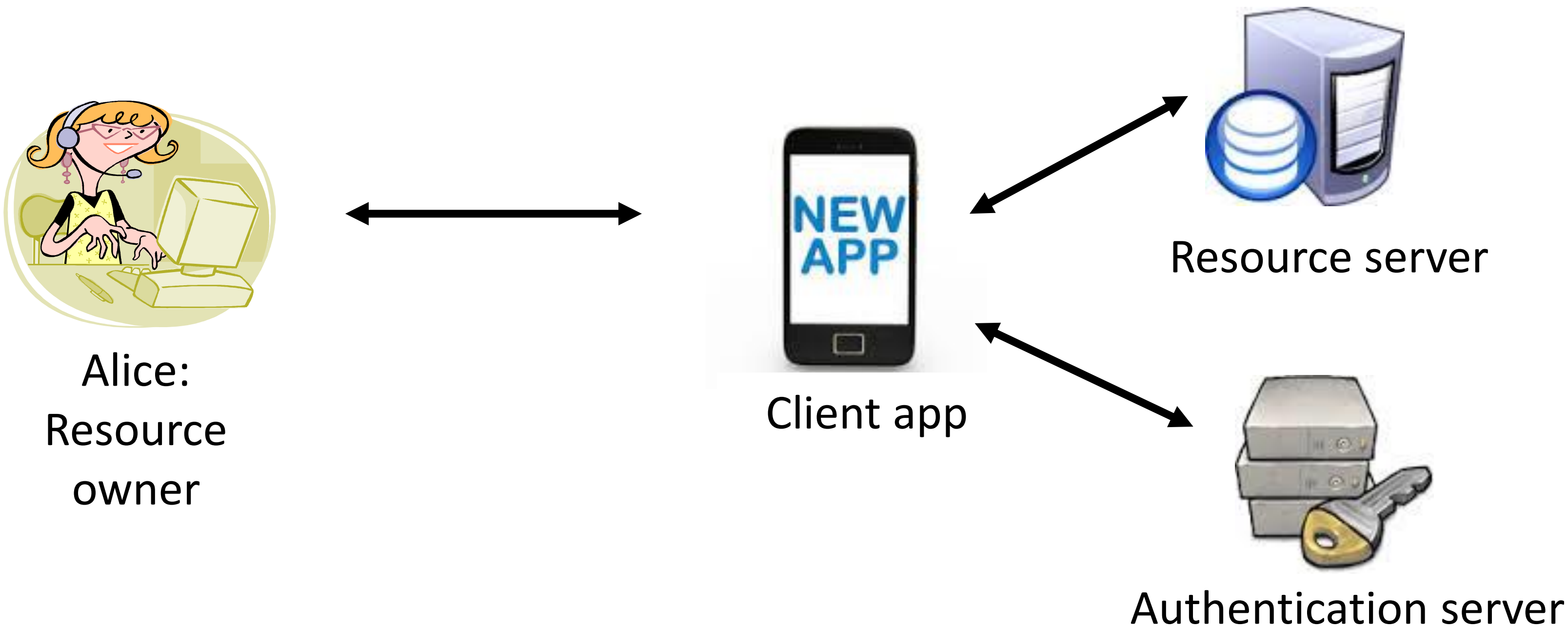
https://gist.github.com/mziwisky/10079157

OAuth 2.0



- The user accesses the app
- The app asks the user to login to the app via Facebook
- The user logs into Facebook, and is sent back to the app
- The app can now access the users data in Facebook
  - Call functions in Facebook on behalf of the user: post status updates)

The Roles



- Resource owner: person or app that owns the data
- Resource server: server hosting the data
- Client: app needs access to data stored on the resource server
- Authorization server: authorizes client to access the data
  - Can be same of different from resource server

Step 1: Client App Registration

- One time process

1. Register, R_URI

2. IdC, passwordC

Client app

Service

Store:
Oauth_clients: [
  Client_app: {
    client_id: IdC
    shared_secret: passwordC
    redirect_URI: R_URI
  }
  …]

Store:
  [service_name: Service
   client_id: IdC
   shared_secret: passwordC
  ]

Example R_URI: app.com/oauth_response

All OAuth communications are encrypted SSL/TLS

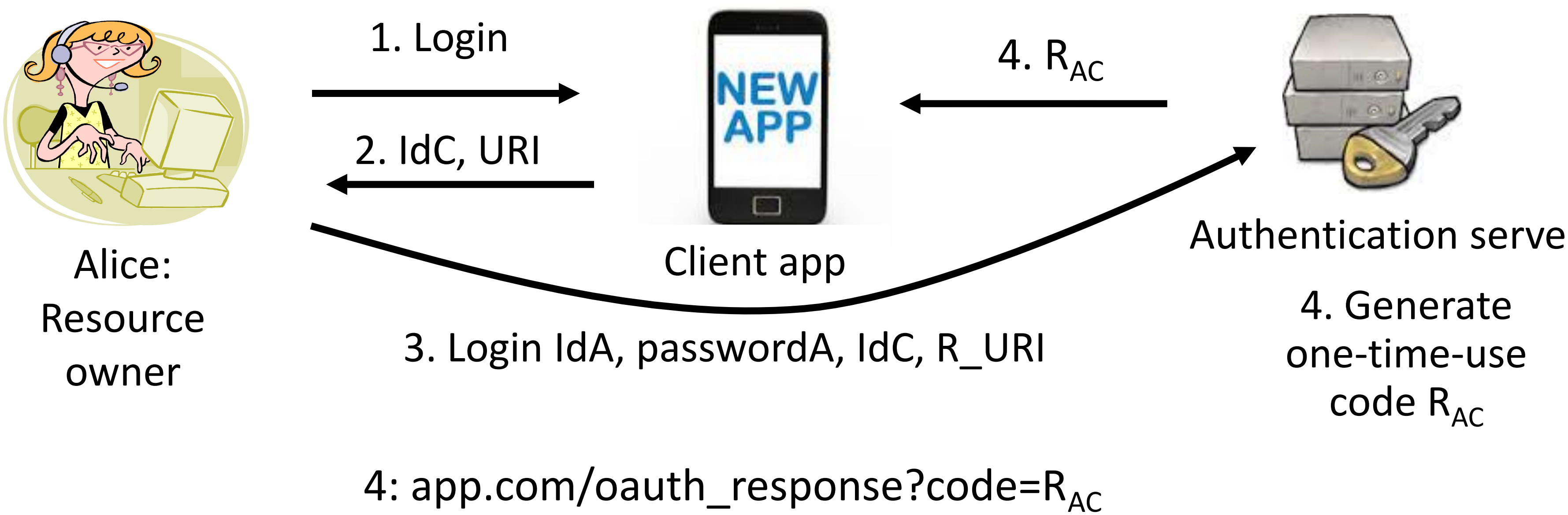
Step 2: User Login

- User starts the app
- Click “Login thru Facebook/Gmail/ …”
- Redirect user to the authentication server
- Authentication server: display page saying “App wants to access your data. Do you authorize?”

2: URI = facebook.com/oauth2/auth?client_id=IdC&redirect_uri=R_URI

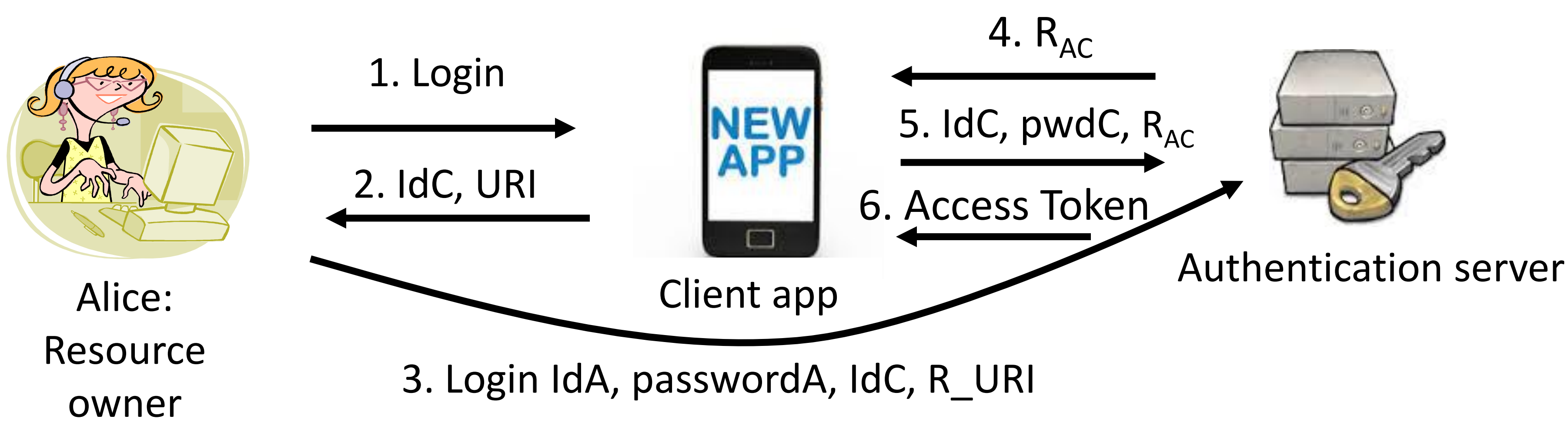
Step 2: User Login (cont'd)

- Authentication server:
  - Associate one-time-use code R_{AC} with app.com
  - Redirects user to the "redirect URI" passing R_{AC} to it



Step 2: User Login (cont'd)

- App takes the code and directly (i.e., not via a REDIRECT) queries authentication server
- Server verifies and then invalidates the R_{AC}
  - Responds with an AccessToken
- App can use Access Token to access the user's data



5: GET facebook.com/oauth2/token?client_id=IdC&client_secret=passwordC&code=R_{AC}

Step 3: User Accesses App