PROTOCOLS

Fadi Mohsen

OUTLINE

- The `protocol' concept
  - Authentication
- Perfect Forward Secrecy
- Kerberos
- SSL

EXPECTATION

- At end of the class, students should be able to:
  - Understand the internal working of at least three security protocols.
  - Identify the security properties a given a security protocol provide.
  - Determine the security flaws and overheads in a given security protocol.

PROTOCOL

- What is a protocol?
  - Protocols are `scenarios' , 'sequence of operations"
  - Well known networking protocols: *http, https, ssh, telnet, ftp*
  - Protocols may be vulnerable to attacks. E.g.,
    - Replay
    - MiM

PROTOCOL

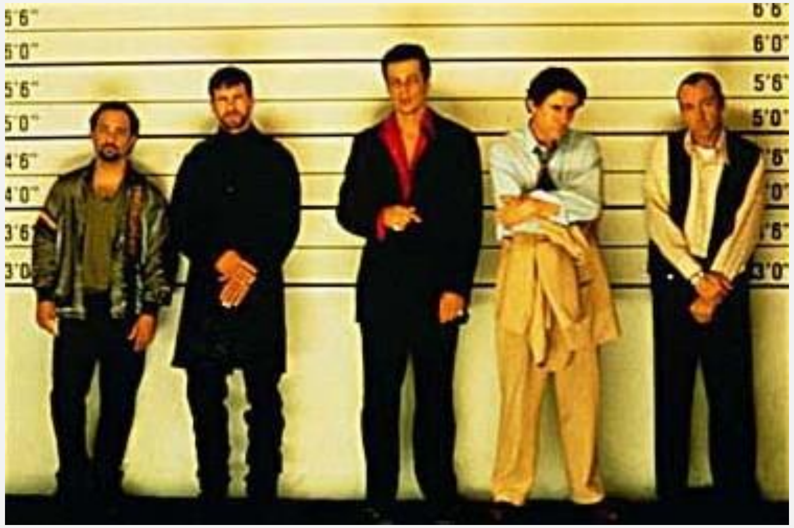
- Protocols may be vulnerable to attacks.
  - Attacks against the protocol:
    - What can the attacker do with the messages that are passed?
  - Attacks against the authentication method:
    - What do you *have, know,* or what is it you *are*.
  - Attacks against the implementation:
    - e.g., buffer overflow attacks

PROTOCOL

- A simple real-life protocol:
  - Slide your bank-pass through the card-reader
  - Enter your pin
  - The pass (or maybe even your account) is invalidated if you enter 3 times an incorrect pin-code (Possibility of denial of service attack)

AUTHENTICATION

- Authentication
  - answers the question:
    who am I communicating with?
- Related concepts:
  - *Identification*:
    who is X.
  - *Authorization*:
    which permissions were granted to whom?



AUTHENTICATION

- Authentication
  - Verification through:
    - What you *are*,
    - What you *have*,
    - What you *know*.
  - When 2 items are required: *two-factor* authentication
    - Bank-pass + pin-code

AUTHENTICATION

- Authentication
  - By passwords
    - *Should* not be guessable
    - There are (too) many of them
    - People tend to select weak passwords

    - A cryptographic key would be preferable

AUTHENTICATION

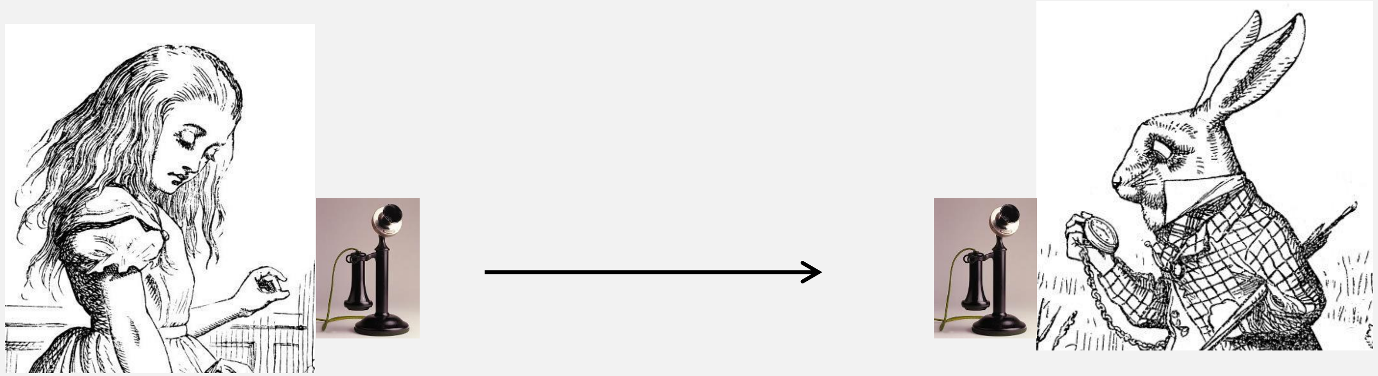
- Authentication
  - Passwords
    - A cryptographic key would be preferable
      - A password that is 8 characters long, with 256 possible choices for each character: $256^8 = 2^{64}$. *(possible passwords)*
      - In practice: a ***much*** smaller number of chars is **used**.
      - The number of tries needed to crack the password is 2^{64}
      - A 64 bits key (if the key was chosen at **random**), attacker would require on average $2^{64}/2 = \mathbf{2^{63}}$ tries. Big enough?

AUTHENTICATION

- Breaking authentication:
  - Require password changes?
  - Limit the number of attempts?
    - How long is an account blocked?
      - Too short: keep trying;
      - Too long: easy DoS attack vector
  - Attack vector: search the hash in Rainbow Tables.
    - Not feasible with salted password hashes

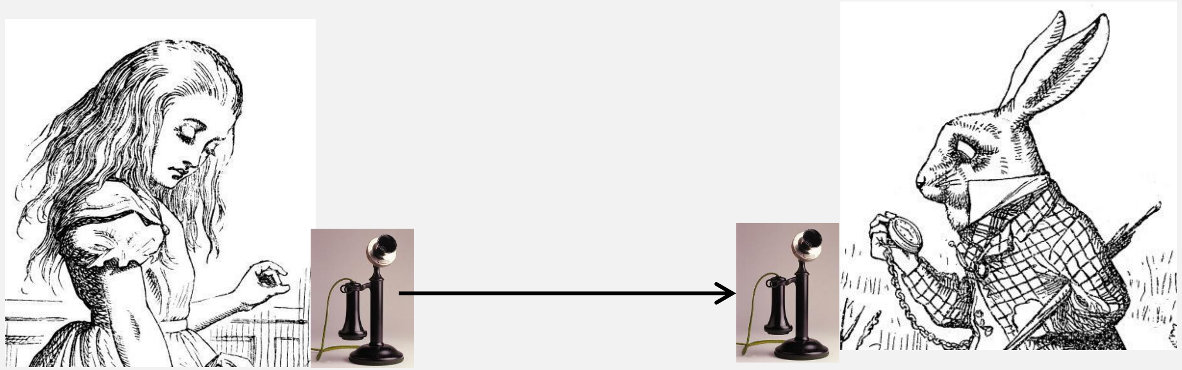
AUTHENTICATION PROTOCOLS

- Setting: *Alice* calls *Bob*. How does *Bob* know it's *Alice*?
  - *Bob* may use *voice recognition*.
  - What if *Bob* and *Alice* don't know each other?



AUTHENTICATION PROTOCOLS

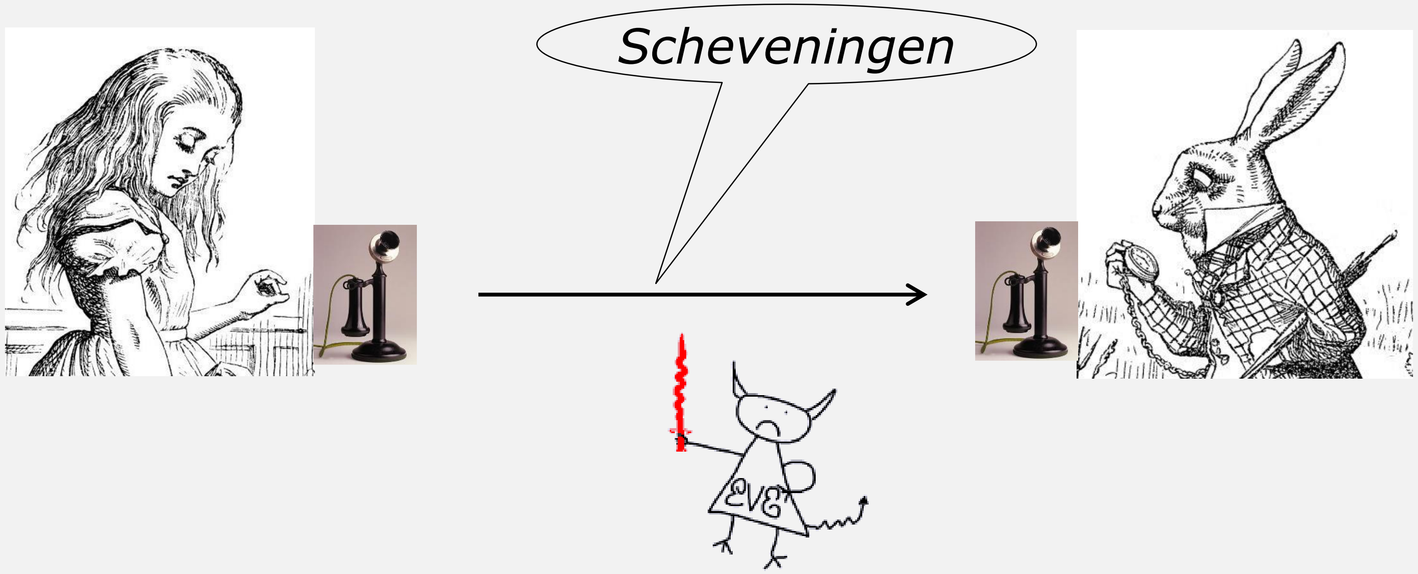
- Setting: *Alice* calls *Bob*. How does *Bob* know it's *Alice*?
  - Alice uses a password: "*the password is Scheveningen*"



When looking at a protocol, ask yourself: Given the protocol what would happen if Eve would take Alice's place. Would that be possible?

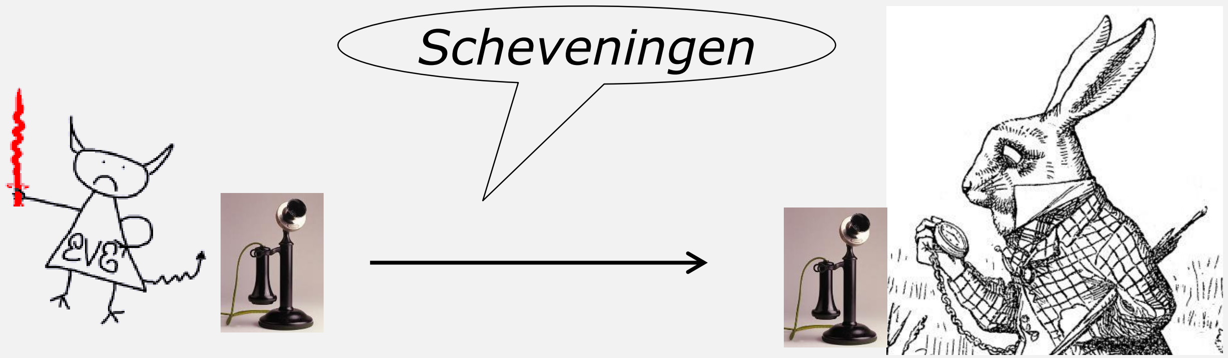
AUTHENTICATION PROTOCOLS

- Setting: *Alice* calls *Bob*. How does *Bob* know it's *Alice*?
  - **Problem 1**: *Eve* may tap the wire and obtain the password.
  - **Problem 2**: Bob must know Alice's password...



AUTHENTICATION PROTOCOLS

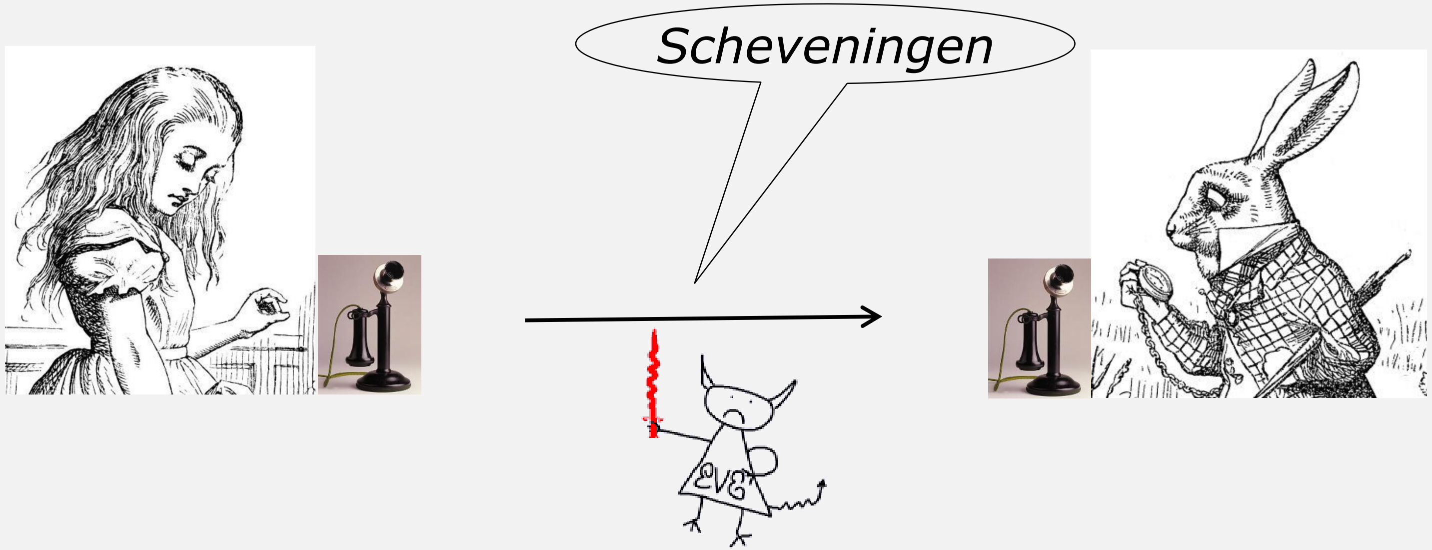
- **Issue**: Now *Eve* calls *Bob* (replay attack) who is tricked into believing it's *Alice*.



AUTHENTICATION PROTOCOLS

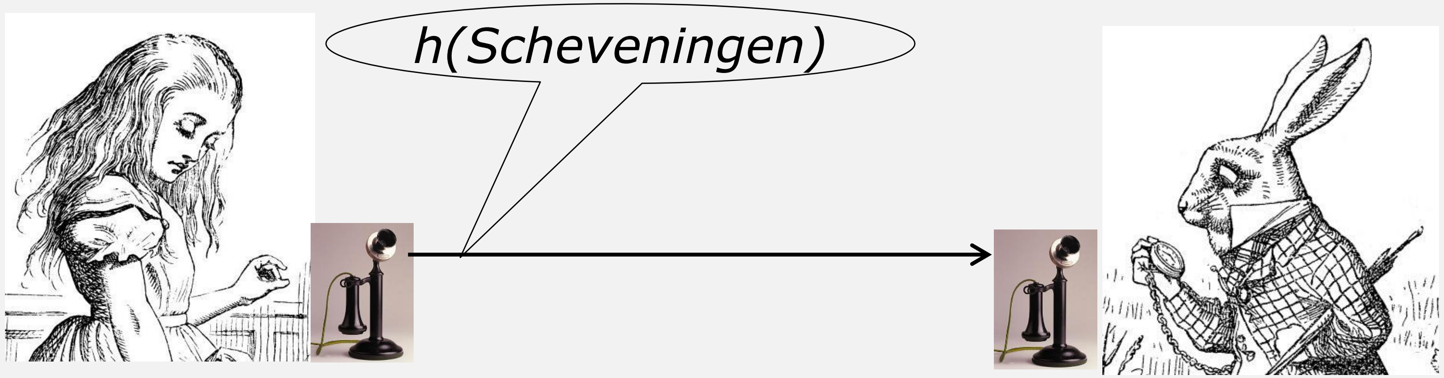
- *Alice* calls *Bob*. How does *Bob* know it's *Alice*?
  - The protocol ***exists***: many old-fashioned protocols use this authentication method: *ftp, telnet, rsh, smtp*...

    https://www.shodan.io/explore/search?query=tags%3Aftp&page=4



AUTHENTICATION PROTOCOLS

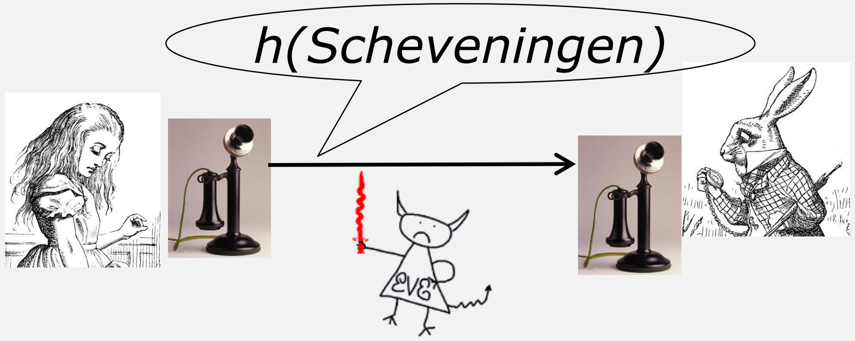
- Authentication protocols:
  - 1st Improvement of the protocol:
    - Don't send the password anymore.
    - Alice sends *h(password)*; Bob only needs *h(password)*



AUTHENTICATION PROTOCOLS

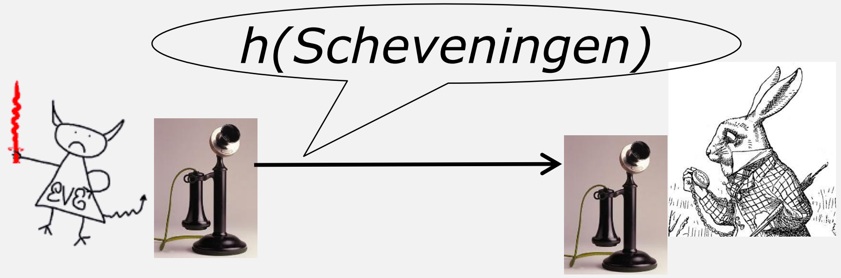
- Authentication protocols:
  - 1st Improvement of the protocol: Alice sends *h(password)*; Bob only needs *h(password)*

✔ Problem 2



  - Eve still can do evil things…:

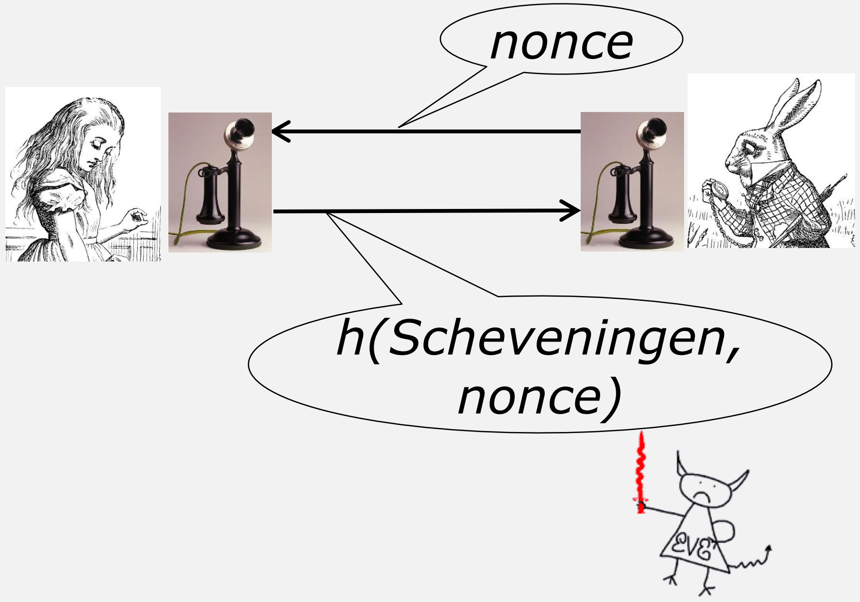
? Problem 1



AUTHENTICATION PROTOCOLS

- Authentication protocols:
  - 2nd improvement: Bob sends a *challenge*, e.g., a *nonce*, which is hashed by Alice together with her password.

Problem 2 is back

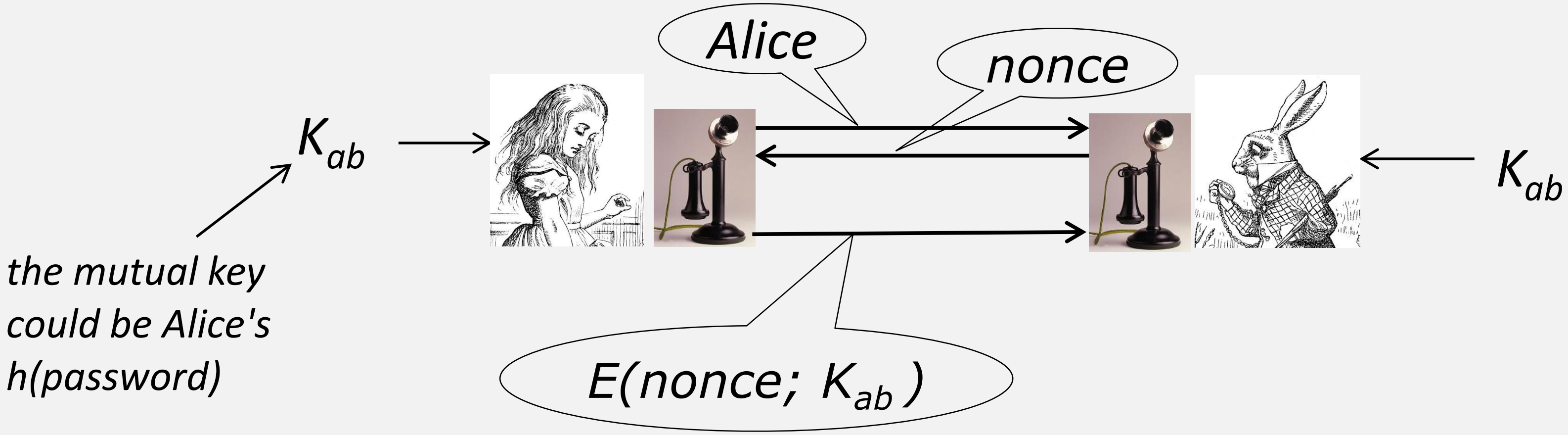


  - Now Eve has a problem... Good!

... but what is Alice's problem...?

AUTHENTICATION WITH SYMMETRIC KEYS

- Authentication protocols:
  - Alice and Bob use a symmetric key K_{ab}:

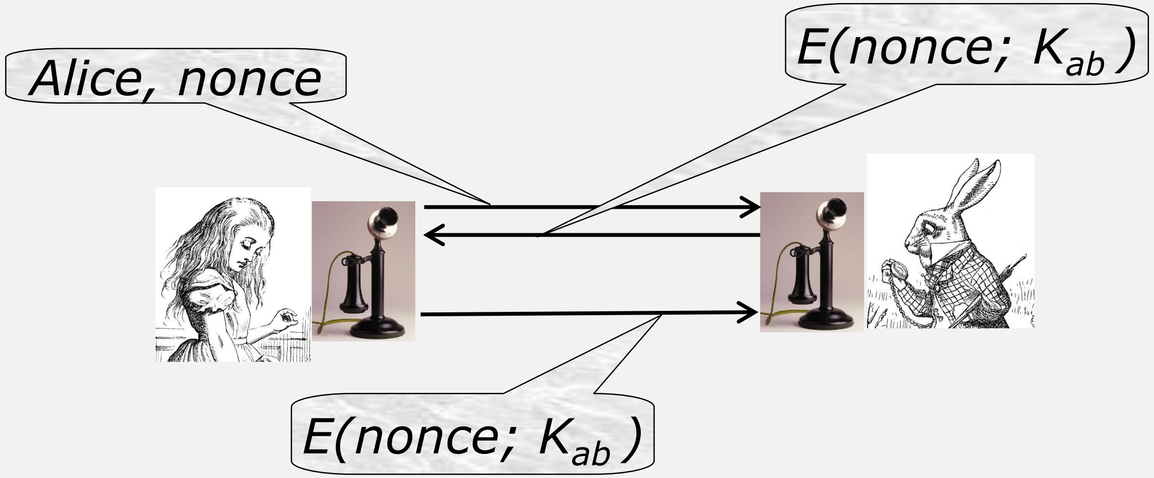


  - Alice authenticates to Bob, this is solved!

Now, can they do *mutual authentication* with this?

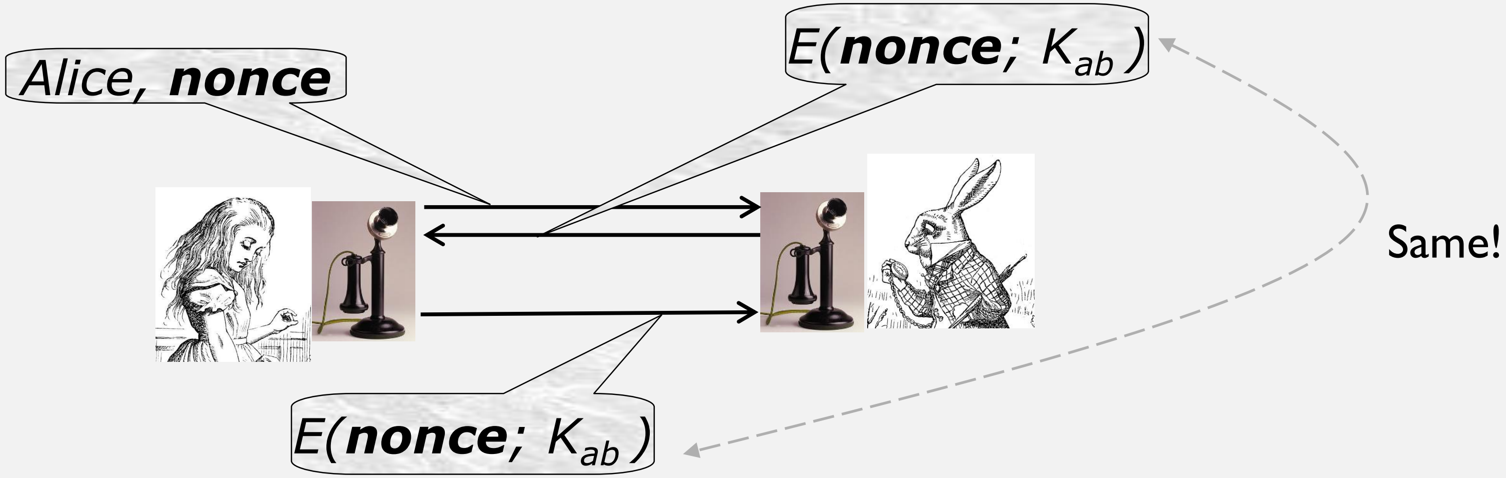
AUTHENTICATION WITH SYMMETRIC KEYS

- Authentication protocols:
  - Alice and Bob both have K_{ab}: could they do *mutual authentication*?



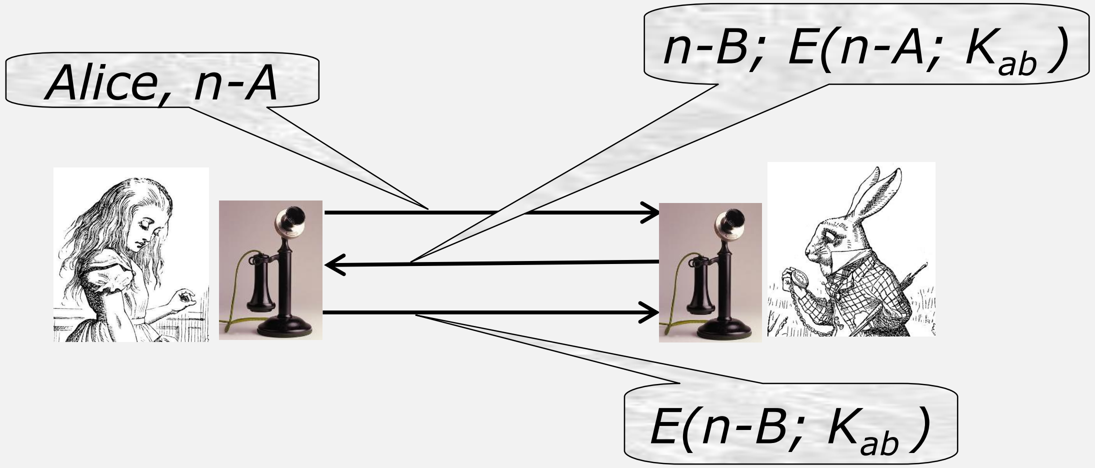
AUTHENTICATION WITH SYMMETRIC KEYS

- Authentication protocols:
  - Alice and Bob both have K_{ab}: could they do *mutual authentication*?
  - But...



AUTHENTICATION WITH SYMMETRIC KEYS

- Authentication protocols:
  - As Alice and Bob both have K_{ab} they could maybe use their *own nonce (nonce for each)*?



  - This protocol has a (subtle) flaw...

AUTHENTICATION WITH SYMMETRIC KEYS

- Authentication protocols:
  - Using own nonces opens possibilities for Trudy: *MiM attack*:

Alice,n-A ⟶

⟵ *n-B, E(n-A; K_{ab})*

Alice,n-B ⟶

⟵ *n-C, E(n-B; K_{ab})*

a side-channel (new connection)

E(n-B; K_{ab}) ⟶ Alice authenticated

AUTHENTICATION WITH SYMMETRIC KEYS

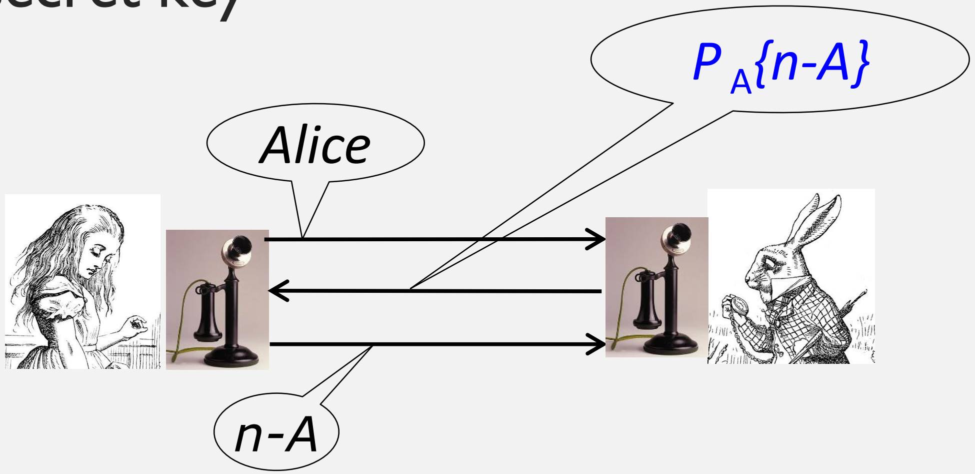
- Conclusions:
  - Should allow identification (mutual authentication);
  - Each side must do something ***different***;
  - *Solution* to the previous MiM attack: include identification for Bob and Alice in the **encryption**
    - Alice sends $E(\text{"Alice"}, n\text{-}A; K_{ab})$
    - Bob sends $E(\text{"Bob"}, n\text{-}B; K_{ab})$

Identity information is part of the ciphertext!

    - Trudy can't do this as she has to reply with a *new* encrypted message: $E(\text{"Alice"}, n\text{-}B; K_{ab})$

AUTHENTICATION WITH ASYMMETRIC KEYS

- **Using a public key infrastructure (PKI) for *encryption* and *authentication*.**
  - P: public key, S: secret key



  - Only Alice can obtain and send *n-A*:
    $S[P\{n\text{-}A\}] = n\text{-}A$

AUTHENTICATION WITH ASYMMETRIC KEYS

- **Using a PKI for encryption and authentication**.
  - What if *encryption* and *authentication* keys are the same?
  - In that case: $P\{X\}$ to encrypt, $S[X]$ to decrypt; but also:

    $S[X]$ to sign, $P\{X\}$ to verify the signature.
  - $S[P\{X\}] = X$, and $P\{S[X]\} = X$.
  - Assume Trudy intercepts $P_A\{M\}$...

    (an encrypted message for Alice)

What happens?

AUTHENTICATION WITH ASYMMETRIC KEYS

- Using plain PKI with Session Keys *K*
  - Drawback: no *mutual* authentication (only Alice authenticates)

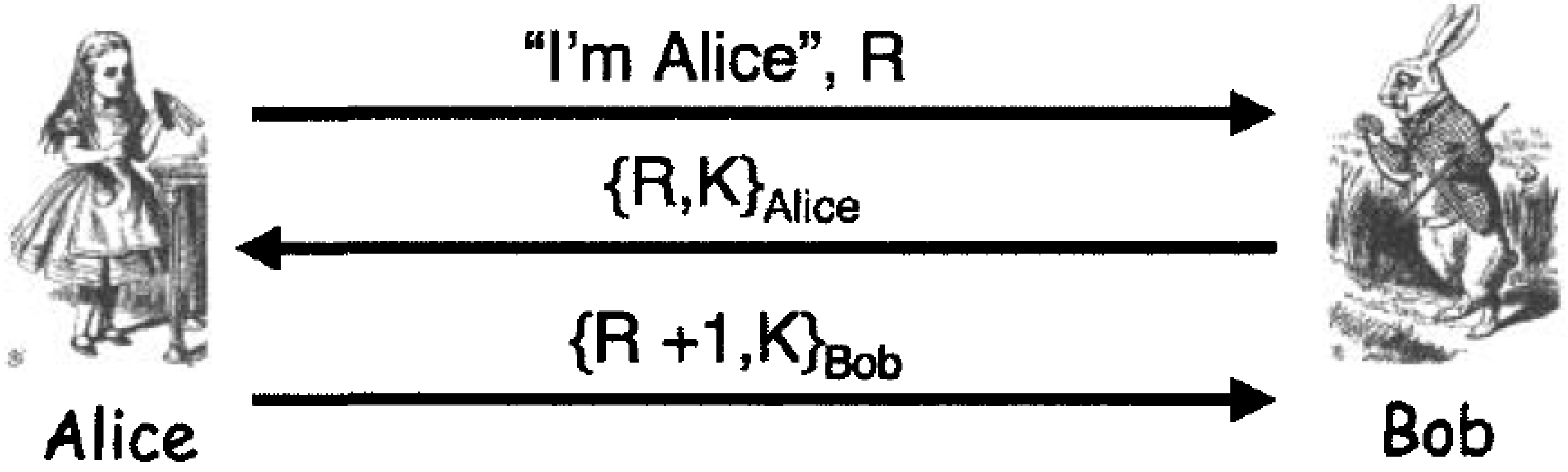


Figure 9.15: Authentication and a Session Key

AUTHENTICATION WITH ASYMMETRIC KEYS

- Using plain PKI with Session Keys *K*
- Drawback: anybody can use Bob's (or Alice's) public key and find the session key *K*.

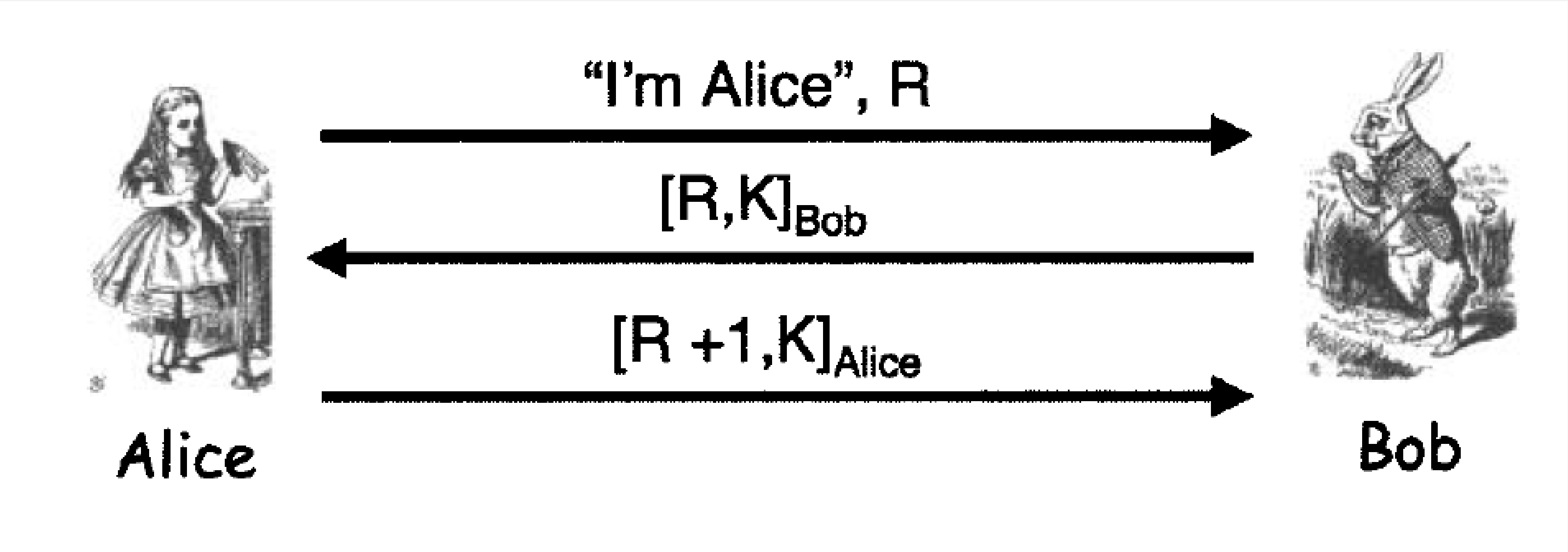


Figure 9.16: Signature-Based Authentication and Session Key

mutual authentication, But???

AUTHENTICATION WITH ASYMMETRIC KEYS

- Using plain PKI with Session Keys *K*
- Combine the previous two approaches (**Sign and encrypt**).

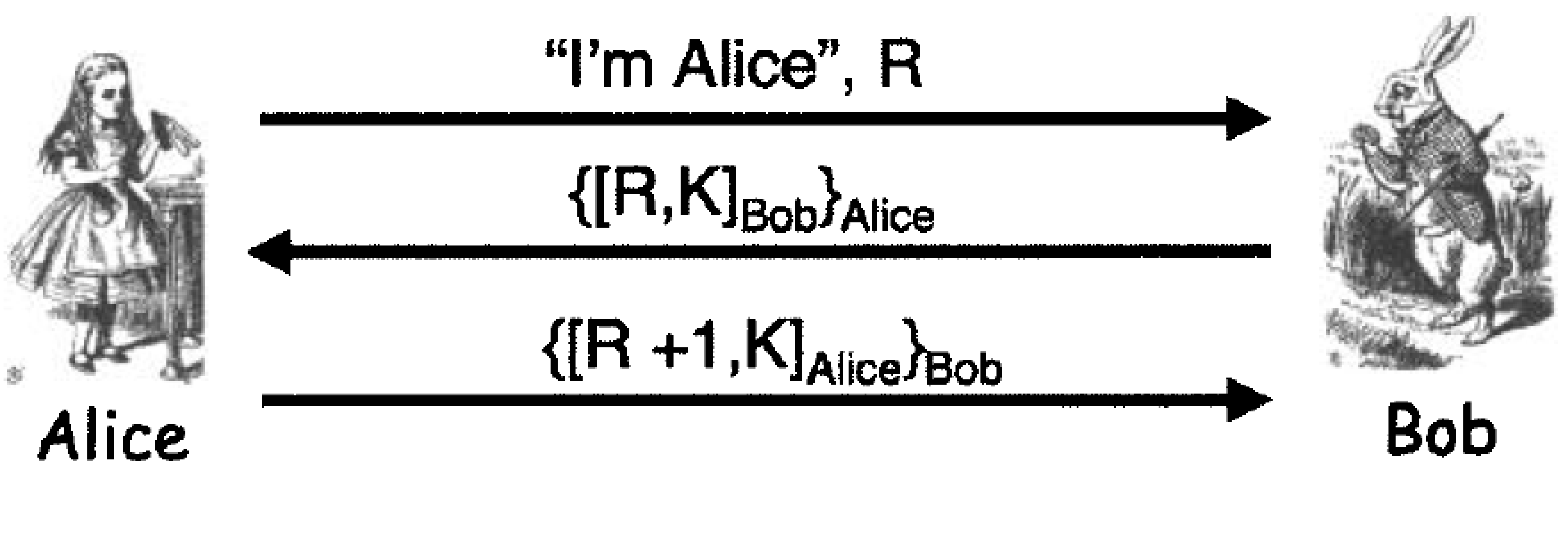


Figure 9.17: Mutual Authentication and Session Key

AUTHENTICATION WITH ASYMMETRIC KEYS

- Using plain PKI with Session Keys *K*
- Combine the previous two approaches (**Encrypt and sign**).

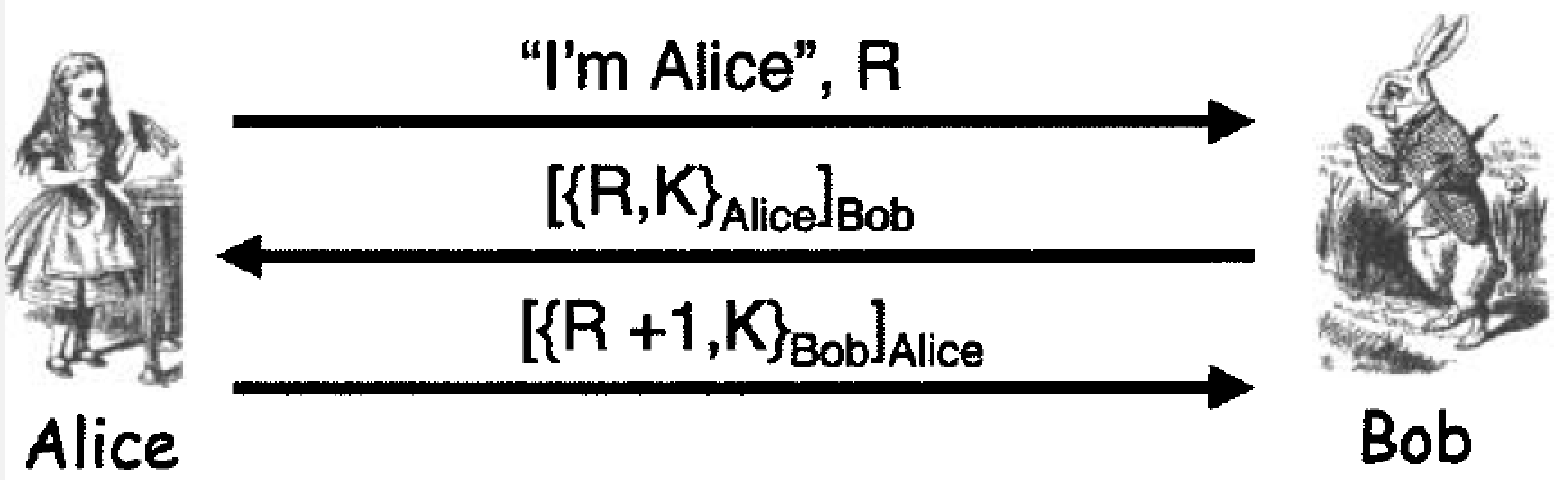


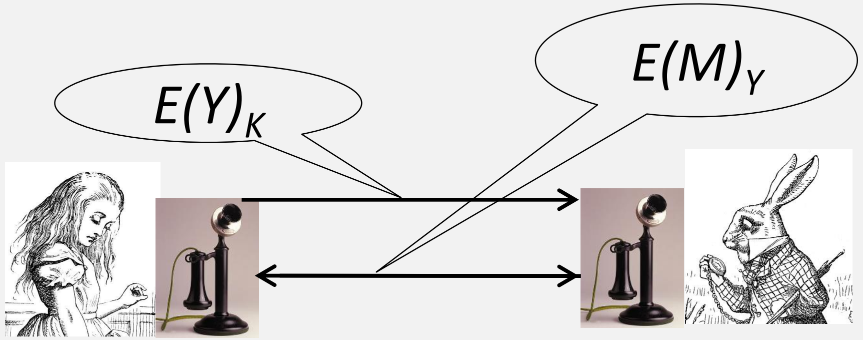
Figure 9.18: Encrypt and Sign Mutual Authentication

PERFECT FORWARD SECRECY (PFS)

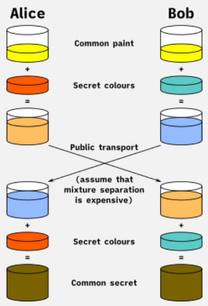
- *K*: a shared (between Alice and Bob) symmetric key
- A & B use *K* to exchange encrypted messages
- Trudy intercepts and records encrypted messages/ciphers
- At some point in the future, Trudy accesses Alice's computer and finds *K*
- *Trudy is able decipher and read all the captured messages.*
- Unlikely attack, but potentially significant.

PFS provides a solution to this unlikely scenario!

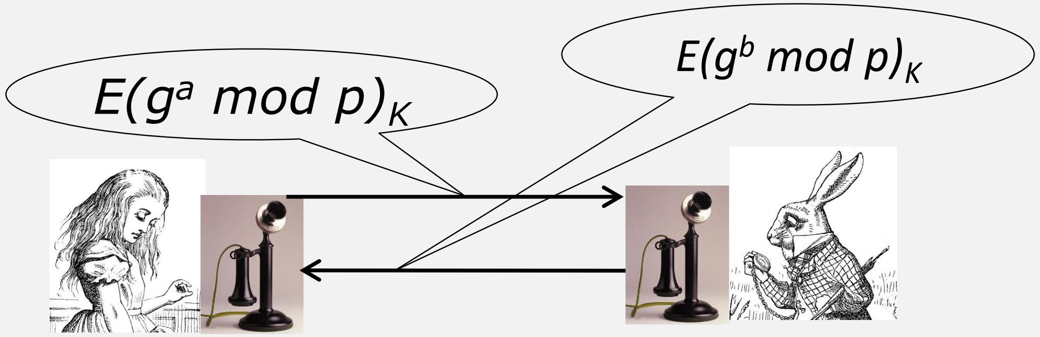
They need to destroy the key once they finished!
What if it is a long term key?
If one is careful with the keys, how can you be sure of the other party?



PERFECT FORWARD SECRECY (PFS)



- One elegant way: PFS using ***ephemeral*** *Diffie-Hellman (symmetric key encrypts the comm.):*



To prevent the MiM attack, the initial communication must be encrypted using the long term key

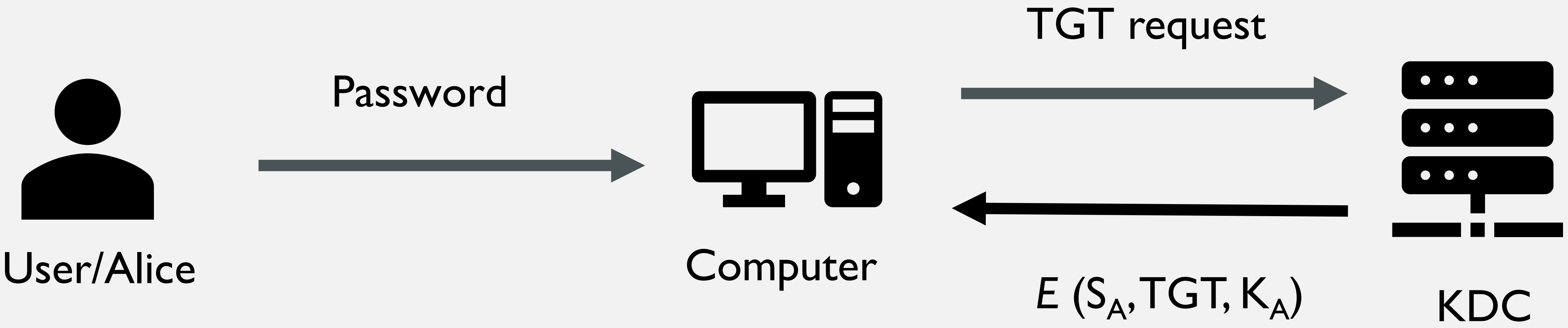
- In this protocol, g and p are public, Alice chooses her secret exponent a and Bob chooses his secrete exponent b.
- Then, Alice sends $g^a \bmod p$ to Bob and Bob sends $g^b \bmod p$ to Alice
- Alice & Bob compute $g^{ab} \bmod p$ (shared session secret) then erase a and b.

KERBEROS

- Is a popular authentication protocol that uses symmetric key cryptography and timestamps.
- The Kerberos Trusted Third Party (TTP) is known as the key distribution center, or KDC.
- The KDC shares a symmetric key with every client/user, e.g. K_A
- The KDC also has a master key K_{KDC}, which is known only to the KDC
- The KDC acts as a go-between that enables any pair of users to communicate securely with each other.
- The KDC issues various types of tickets to access network resources.
  - all-important ticket-granting ticket, or TGT (one special ticket)

KERBEROS

A TGT, which is issued when a user initially logs into the system, acts as the user's credentials. The TGT is then used to obtain (ordinary) tickets that enable access to network resources.



Since the TGT is encrypted with the key K_{KDC}, why is the TGT encrypted again with the key K_A?

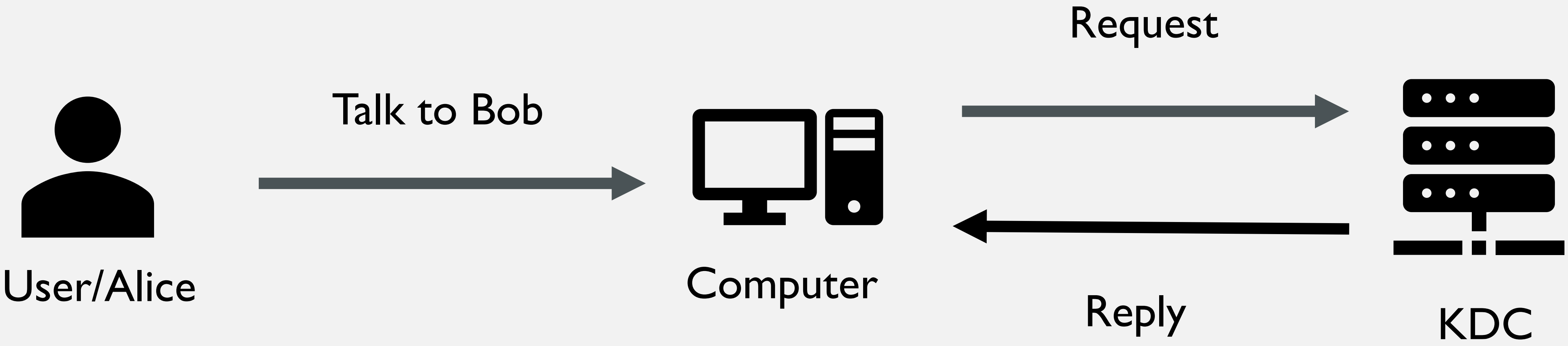
The key K_A is derived as $K_A = h$ (Alice's password), is the key that Alice and the KDC share.

The KDC creates a session key S_A. TGT = E ("Alice", S_A, TIMESTAMP, K_{KDC})

The TGT (response) helps the KDC to stay stateless. Otherwise, it will have to maintain a database of which users are logged in, their session keys, etc.

KERBEROS

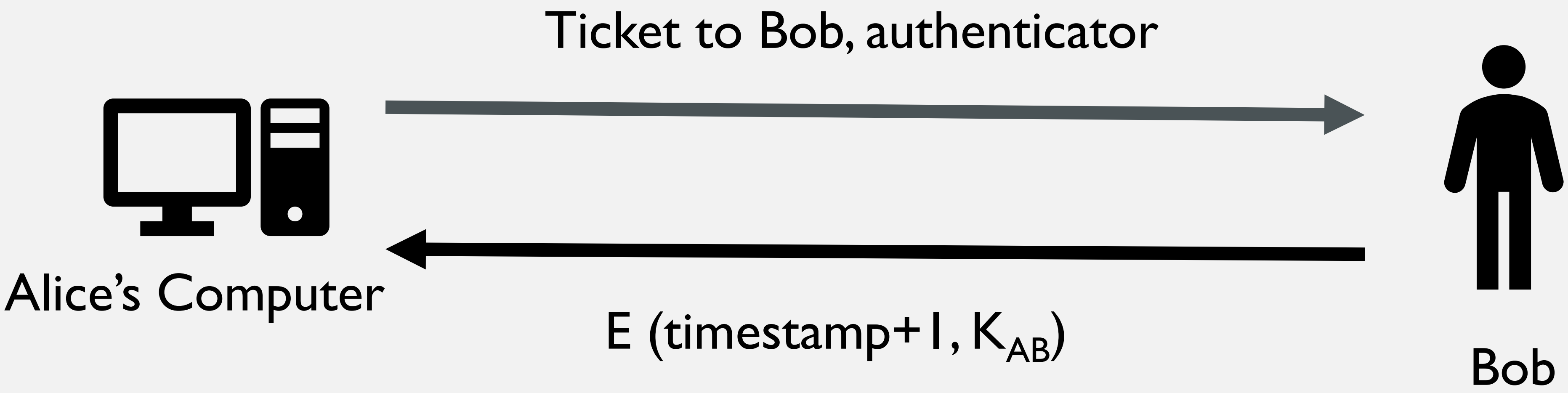
Once Alice's computer receives its TGT and S_A, it can then use the TGT to request access to network resources. For example, suppose that Alice wants to talk to Bob.



Request = (TGT, authenticator). Authenticator = *E* (timestamp, S_A)

Replay = *E* (“Bob”, K_{AB}, ticket to Bob, S_A). Ticket to Bob = *E* (“Alice”, K_{AB}, K_B)

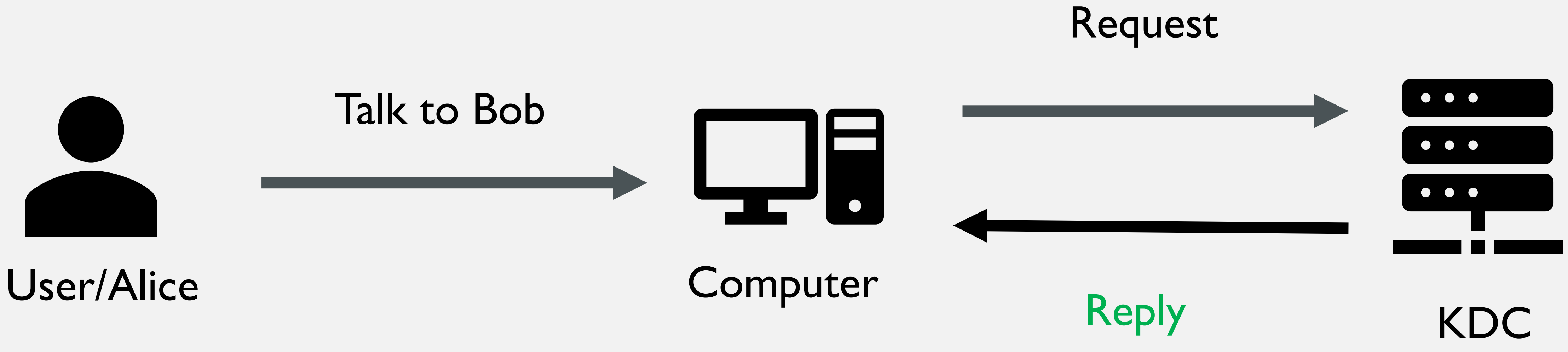
KERBEROS



Ticket to Bob = E ("Alice", K_{AB}, K_B). Authenticator = E (timestamp, K_{AB})

Timestamps are used for replay prevention.

KERBEROS



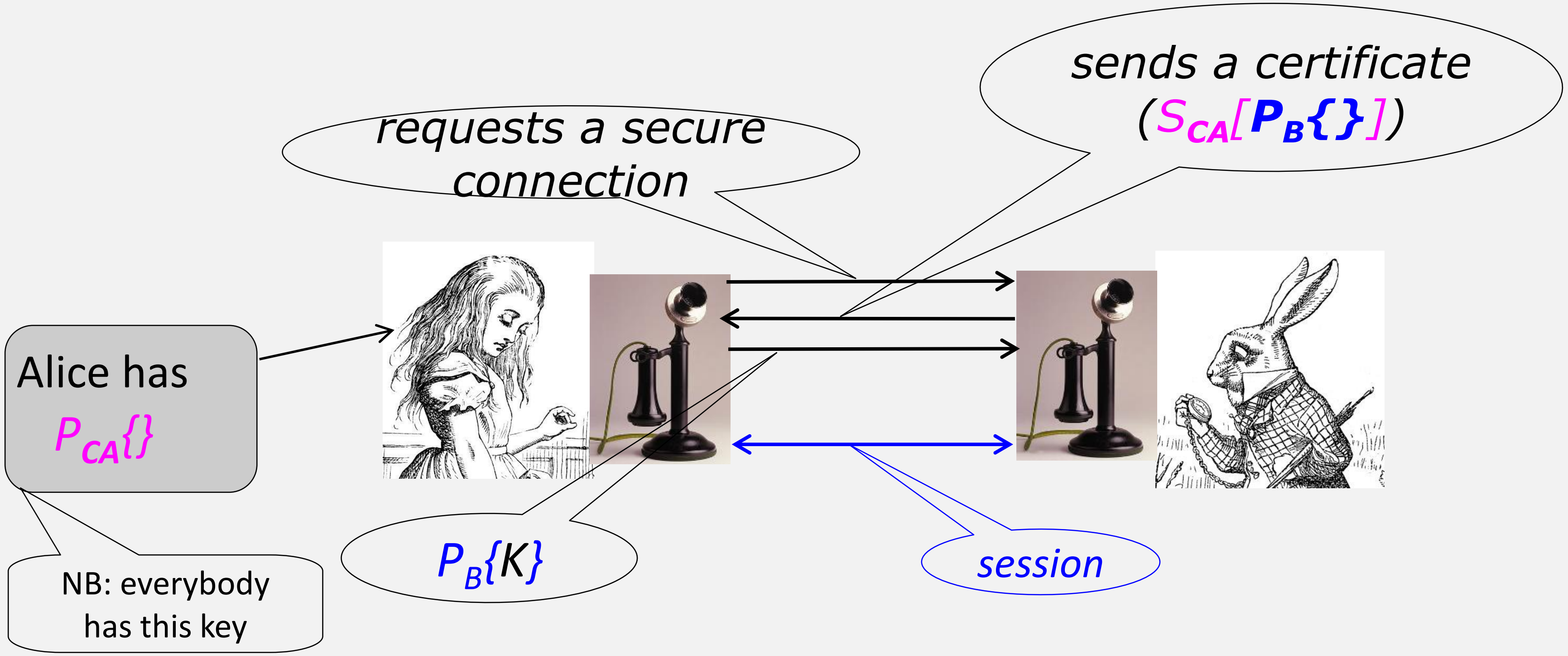
Replay = E (“Bob”, K_{AB}, ticket to Bob, S_A). Ticket to Bob = E (“Alice”, K_{AB}, K_B)

Why not send “ticket to Bob” to Bob instead of Alice?!!

Bob would then need to remember K_{AB} until needed (maintain state). Kerberos is stateless

SECURE SOCKET LAYER (SSL)

- Protocol to obtain confidentiality and integrity for inter-computer communications.
- Essential steps:



SECURE SOCKET LAYER (SSL)

- An SSL-certificate:

SECURE SOCKET LAYER (SSL)

- Details of an SSL-certificate:

This certificate has been verified for the following uses:

SSL Server Certificate

Email Signer Certificate

Email Recipient Certificate

Issued To

Common Name (CN)	security.rc.rug.nl
Organization (O)	University of Groningen
Organizational Unit (OU)	Computing Center (Security Section)
Serial Number	01:00:00:00:00:01:11:9C:AE:1D:0B

Issued By

Common Name (CN)	Cybertrust Educational CA
Organization (O)	Cybertrust
Organizational Unit (OU)	Educational CA

Validity

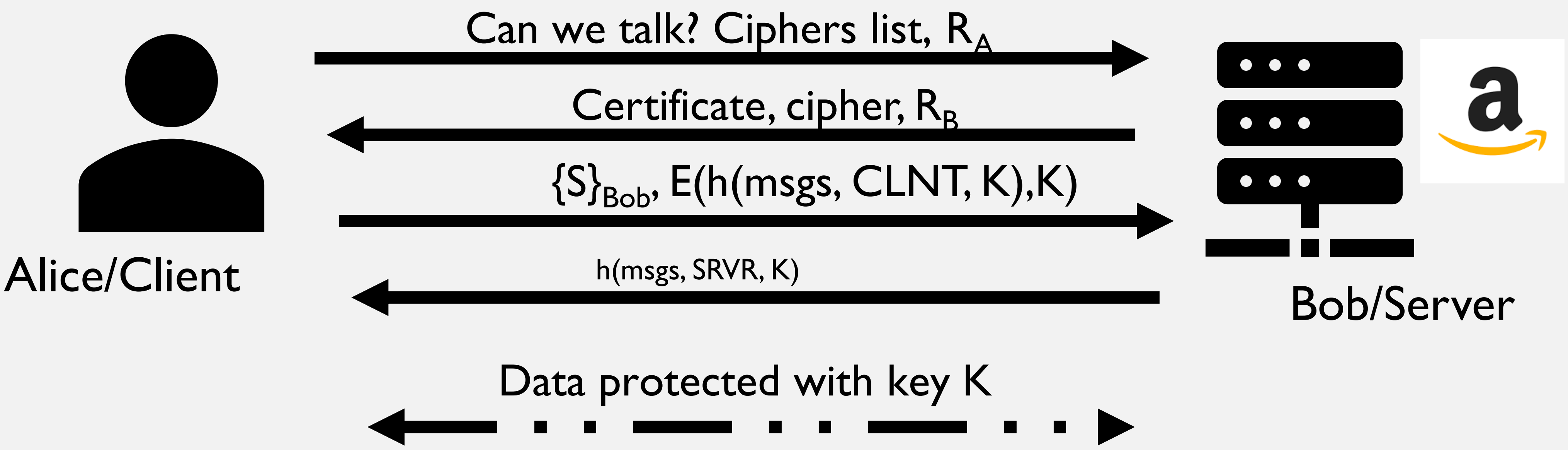
Issued On	03/29/07
Expires On	03/29/10

Fingerprints

SHA1 Fingerprint	27:01:AA:E2:3F:7A:D9:3E:C8:00:6D:7A:4A:AF:AF:8A:F6:DC:80:D5
MD5 Fingerprint	13:70:F8:DF:39:B1:9D:56:DF:42:D2:49:39:51:B7:6A

SECURE SOCKET LAYER (SSL)

Do you notice any flaw/unnecessary step in SSL?



Bob responds with a similar hash. Alice
messages correctly, and she can authent
decrypted S, which is required to gener

$$S = \text{the pre-master secret}$$
$$K = h(S, R_A, R_B)$$
$$\text{msgs} = \text{shorthand for “all previous messages”}$$
$$\text{CLNT} = \text{literal string}$$
$$\text{SRVR} = \text{literal string}$$

SECURE SOCKET LAYER (SSL)

- SSL: MiM protection
  - Bob's certificate must be signed by a CA.
  - Eve can't obtain *K*.
  - Eve *could* send her own certificate, but that certificate either
    - doesn't show Bob (but may be signed by a CA)
    - shows Bob, but isn't signed by a CA.
  - And what happens in that case…?

Nothing happens!

This attack is a very real threat, it's not due to a flaw in the SSL protocol. Instead, it's caused by a flaw in human nature, making a patch much more problematic.

That's All, Folks,

for today.