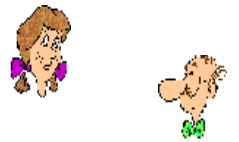


# Security protocols

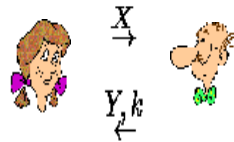
1. Authentication protocols (*this lecture*)
2. Electronic voting protocols
3. Fair exchange protocols
4. Digital cash protocols

# Alice and Bob



- **Alice and Bob** want to communicate with each other over an in secure medium, e.g., the internet.
- But first, they need to
  - Authenticate each other: they want to be sure they are talking to each other, not to an imposter
  - Agree a secret key: they want to encrypt their conversation, for privacy and integrity.
- Alice and Bob might be humans, but they might also be: a web browser and a web server; a mobile phone and the network operator; etc.

# Security protocols



- **Alice and Bob** will engage in an exchange of messages (= a protocol) which will result in each of them having authenticated each other, and established a session key.
  - **A protocol** is an agreed way, or convention, for two or more parties to achieve a goal. It fixes the number and format of the messages between the parties. These rules are known to all of the participants.
- The protocol will typically involve cryptographic primitives, such as encryption and signing, and may involve a trusted third party (TTP).

# Authentication

- **Authentication between humans**
  - How can your bank know that an instruction to pay money to X from your account is actually coming from you?
- **Authentication between devices**
  - How can the mobile phone network know that instructions apparently coming from your phone are actually from your phone?
- **Authentication between programs**
  - How can your browser establish that the server at the other end is really “hsbc.co.uk” and not a look-alike web site designed to defraud you?



# Assumptions and rules of protocol design

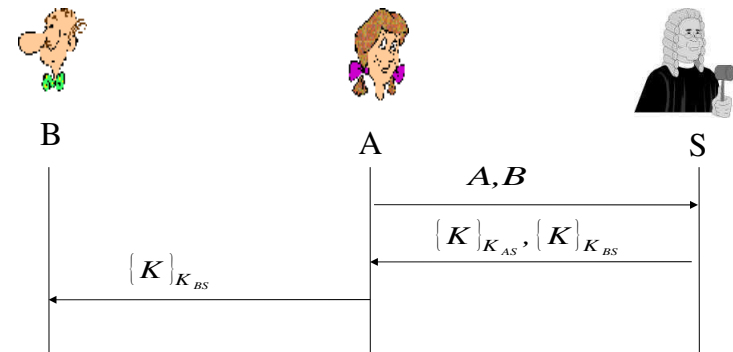
- The “Dolev-Yao model”: the attacker
  - Can read messages
  - Can block messages
  - Can generate fake messages
  - Can read/generate encrypted messages if s/he has the key
  - Knows the protocol being used. (This may not be obvious if A and B are humans, but it is obvious if A is a web server and B a browser.)
- The attacker cannot read/generate encrypted messages if s/he doesn't have the key. (Thus, we assume the crypto is unbreakable.)

# Assumptions and rules contd.

- When we use a *trusted third party* (TTP), we should avoid overloading it:
  - It should not have to *remember state* (e.g., remember nonces, keep lists of used keys, etc.)
    - Each time it is called, it performs a function independently of the other times it has been called.
  - Preferably, it should deal with only one of the participants
- If we don't adhere to these principles, we may make it vulnerable to denial of service attacks.

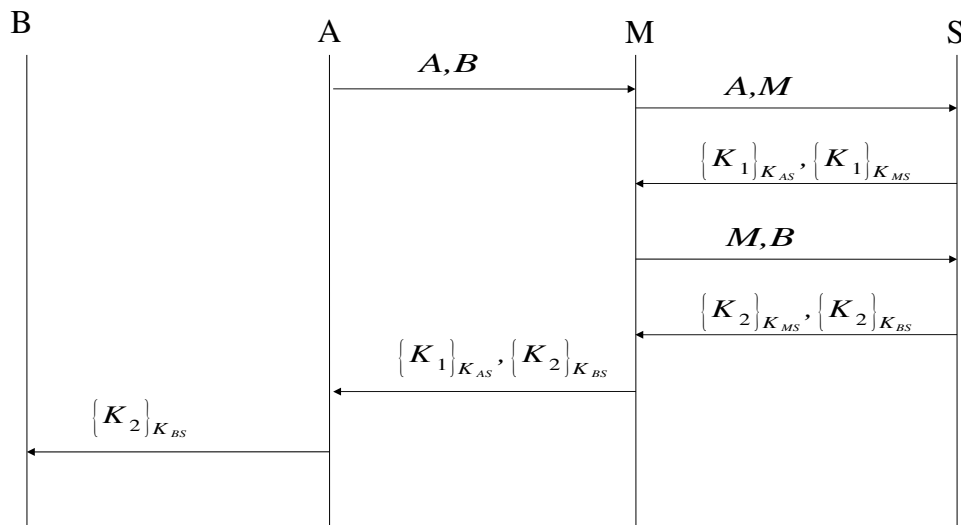
## Symmetric key protocols

## A simple protocol

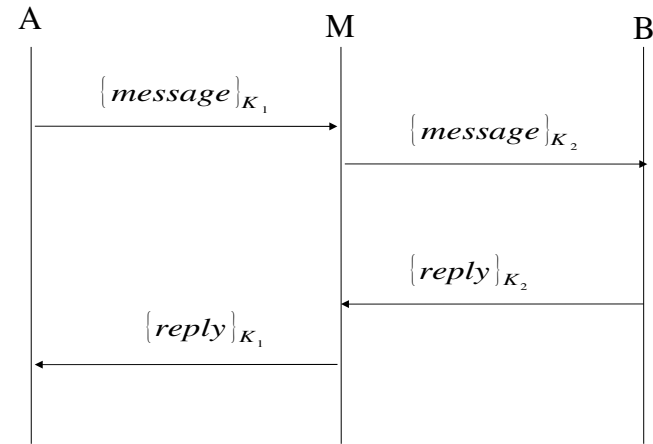


- We assume that  $K_{AS}$  is a long-term key shared only between A and S. Similarly for  $K_{BS}$ .
- Does this protocol work?

# Attack

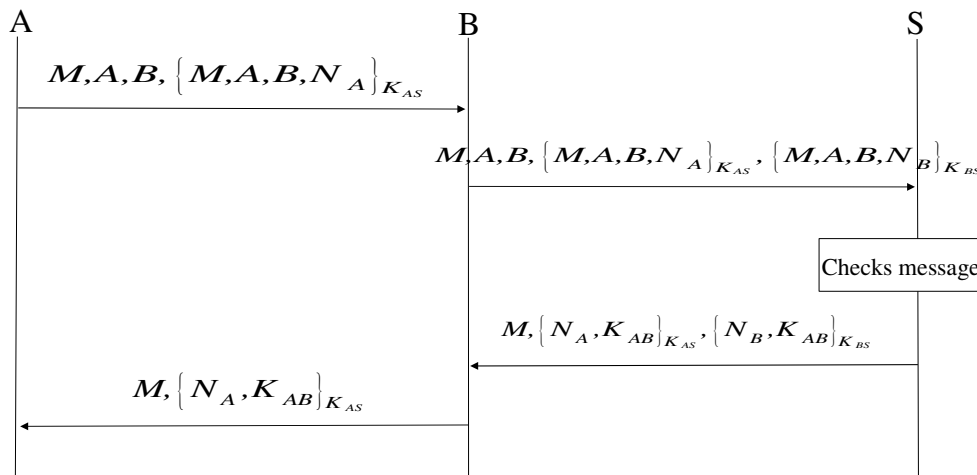


# Using the attack



- Is this a valid attack?
  - Can M intercept the first message of the previous slide (intended for S)
  - Won't S be suspicious since it gets two calls in the attack sequence, instead of just one?

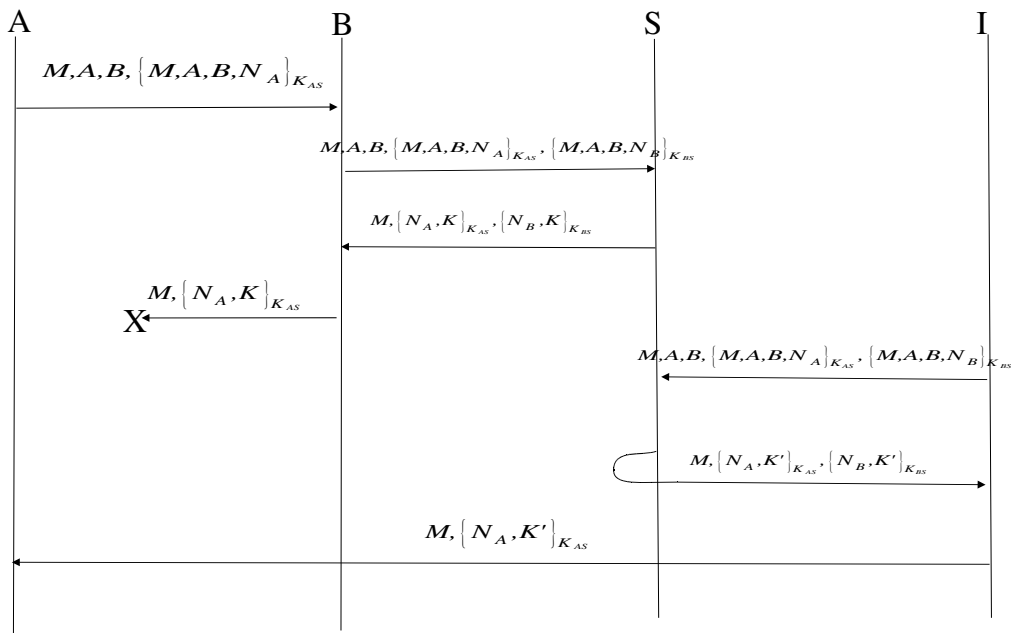
# Otway-Rees



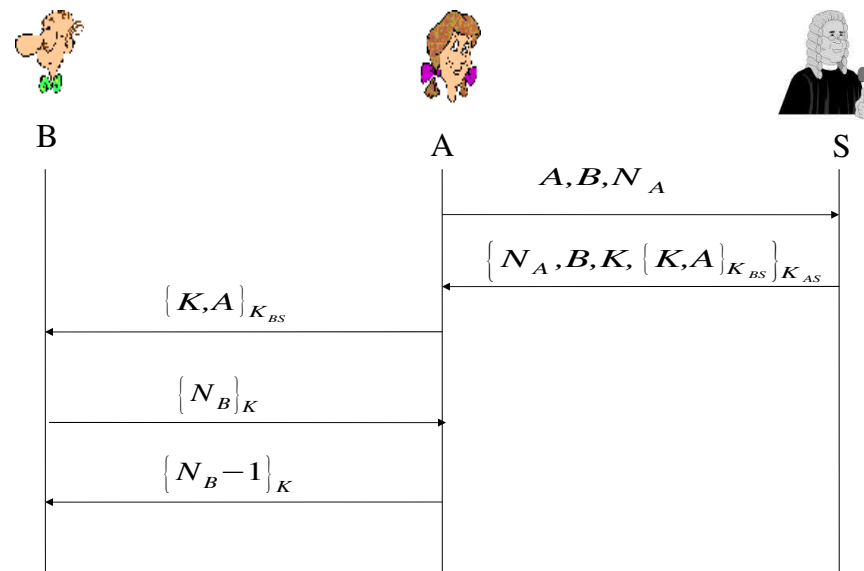
# Otway-Rees -- remarks

- Good things
  - M is a session identifier.
  - Messages are tagged with M, and the identities A,B.
  - S still has to deal only with one of the participants
- Bad things
  - S has quite a lot of work to do: decrypting twice, checking the message format, inventing key, encrypting twice. This makes it vulnerable to Denial of Service attacks.
  - An intruder could interfere with the protocol so that A and B get different keys!

# Attack resulting in different keys

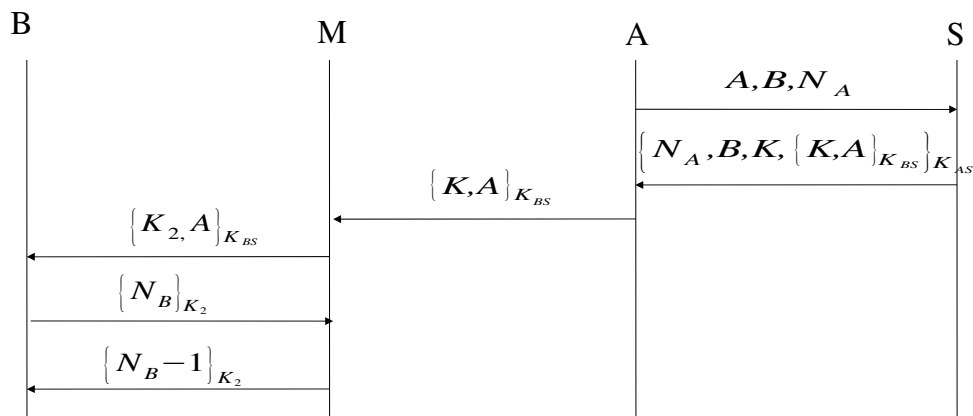


# Needham-Shroeder symmetric key protocol (1978)



## NSSKP vulnerability

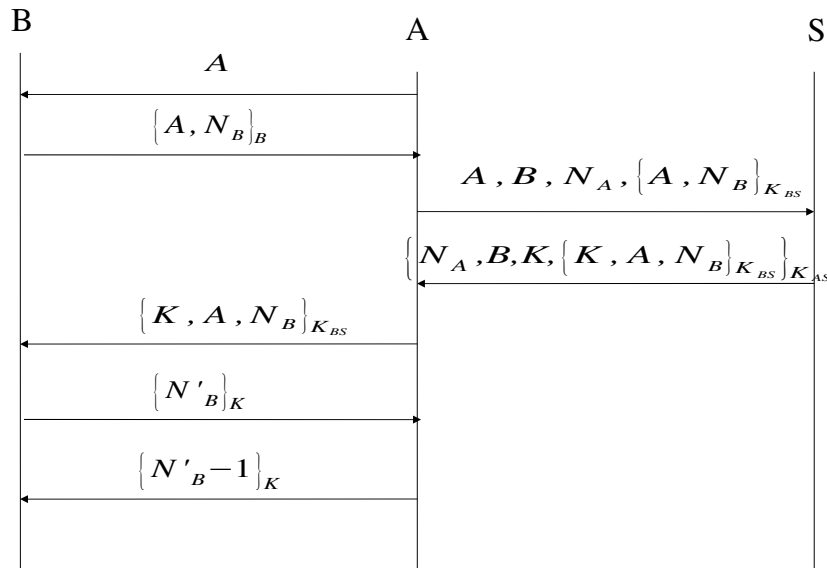
- Mallory might get hold of a previously used session key  $K_2$ , through A's and B's carelessness much later or through off-line cryptanalysis. He has also previously stored the corresponding ticket  $\{K_2, A\}_B$ . Mallory can now trick Bob, by impersonating Alice.



## Solution 1: use timestamps

- Timestamps can be used to solve this problem
  - The Kerberos authentication protocol used (for example) in Windows is based on the Needham Schroeder symmetric key protocol, with the addition of timestamps
- However, timestamps bring with them their own problems
  - Agents need to have synchronised clocks
  - Some tolerance needs to be made for network delays – but how much?

## Solution 2: Needham-Shroeder-Mullender symmetric key protocol (1987)

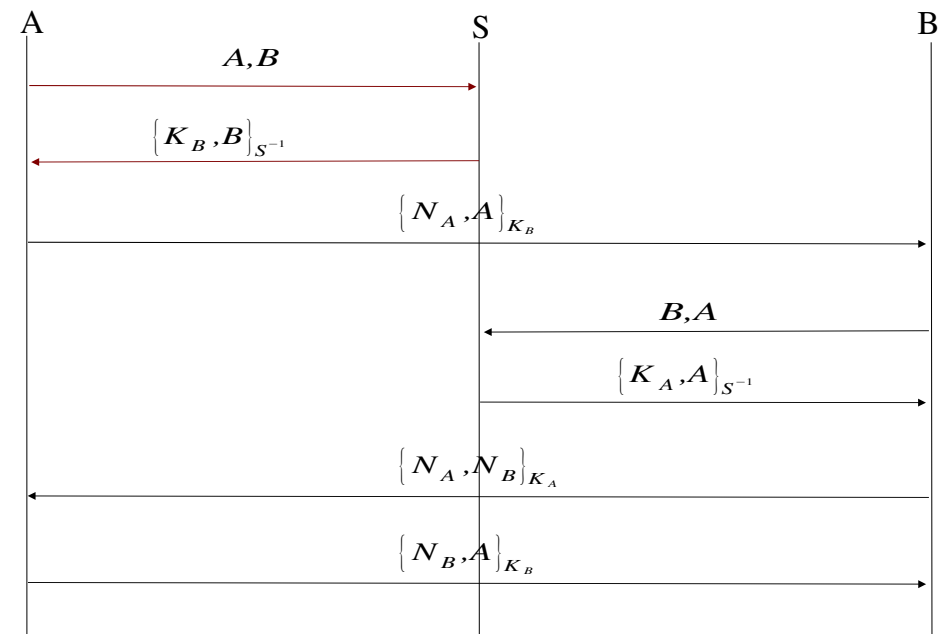


## Public-key protocols

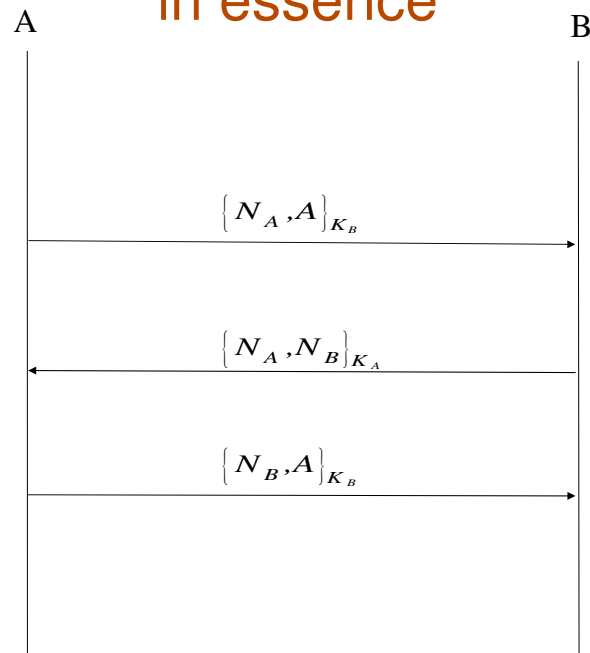
### Celebrated case

- The Needham-Schroeder *public key* authentication protocol was introduced in 1978.
  - We assume that A and B have S's public key.
  - The protocol consists of three parts.
    - In two of the parts, A and B are each obtaining the other one's public key from a server S.
    - In the other part, they exchange nonces which are intended to be used to set up a secret session key.
- A serious flaw in the protocol's third part was found by Gavin Lowe in 1995.

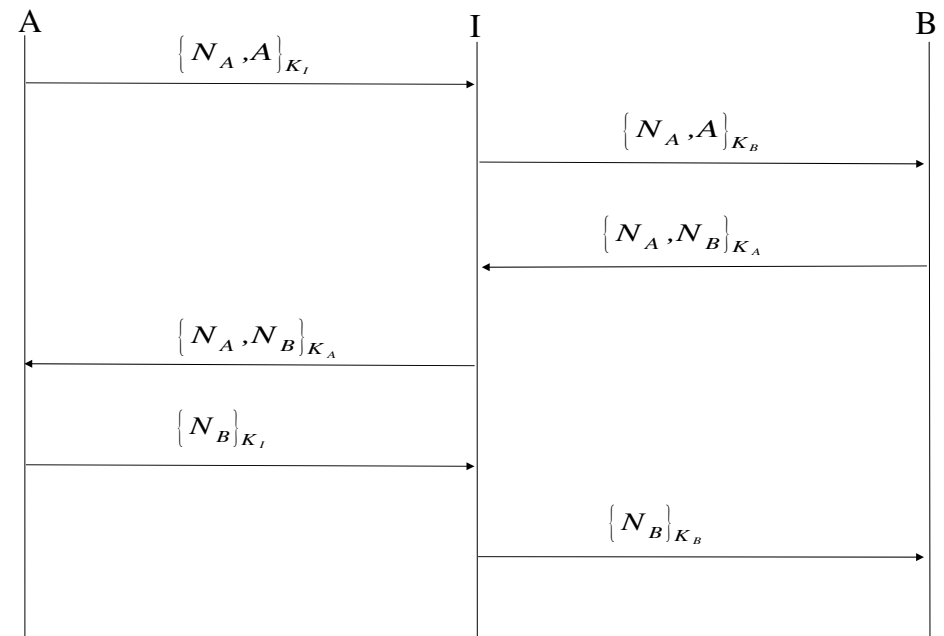
### Needham-Schroeder PK



# Needham-Schroeder PK, in essence

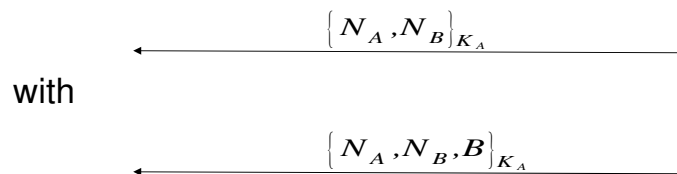


# Needham-Schroeder PK attack



## NS PK attack

- The attack:
  - A has engaged in a session with I, & is aware of that.
  - I has engaged in a session with B, but B thinks he is talking to A. B is duped.
  - Example: A=you, I=dodgy retailer, B=your bank
- The solution is very simple:
  - Replace the message



## Conclusions

- Security protocols are short, but very hard to get right.
  - This makes them ideal for formal analysis
  - One has to be clear about what the goals of a security protocol are.
  - And about the “attacker model”
- Next lectures:
  - Fair exchange protocols
  - Electronic voting protocols
  - Digital cash protocols