Cryptography (and Information Security)
6CCS3CIS / 7CCSMCIS

Prof. Luca Viganò

Department of Informatics
King's College London, UK

First term 2019/20

Lecture 4

Roadmap

The sections labeled with (*) contain additional material that will help understanding but will not be subject of exam questions.

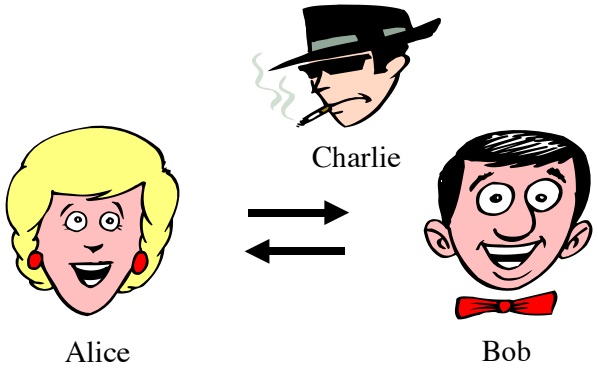
Outline

1. Security protocols
2. Building a key establishment protocol (*)
3. Messages, communication, protools
4. An example: Needham-Schroeder Public Key protocol
5. Examples of protocols and possible vulnerabilities (*)
   - Kinds of attack
   - Otway-Rees protocol
   - Andrew Secure RPC protocol
   - Key exchange with CA (Denning & Sacco)
   - Example of binding attack
   - Example of parallel sessions attack
   - Example of replay attack
6. Prudent engineering of security protocols (*)
7. Kerberos

Table of contents I

Security Protocols

- The world is distributed:
  - Our basic infrastructures are increasingly based on networked information systems.
  - Business, finance, communication, energy distribution, transportation, entertainment...



Alice $\rightarrow$ Bob@Bank: "Transfer $100 to account X"
Bob@Bank $\rightarrow$ Alice: "Transfer carried out"

  - How does Bob know that he is really speaking with Alice?
  - How does Bob know Alice just said it?
  - Confidentiality, integrity, accountability, non-repudiation, privacy... ?

Solutions involve protocols like IPSEC, KERBEROS, SSH, SSL/TLS, SET, PGP... We'll consider underlying ideas and some example protocols.

Definitions

Protocol

A protocol consists of a set of rules (conventions) that determine the exchange of messages between two or more principals.
In short, a distributed algorithm with emphasis on communication.

Security protocol

Security (or cryptographic) protocols use cryptographic mechanisms to achieve security objectives.
Examples: Entity or message authentication, key establishment, integrity, timeliness, fair exchange, non-repudiation, ...

- Small recipes, but nontrivial to design and understand.
  Analogous to **programming Satan's computer**
  as protocols have to run in an environment
  that is under the control of an attacker.

Table of contents I

Building a key establishment protocol (1)

- An attempt to design a good protocol (from first principles).
- First step: establish the **communications architecture**.

  We choose here **one common scenario** from several alternatives:
  - A **set of users**, any 2 of whom may wish to establish a new **session key** for use in subsequent communications.

N.B.: successful completion of key establishment (& entity authentication) is only the beginning of a secure communications session: once a new key has been established, its use comes in protecting the data to be communicated with whatever cryptographic mechanisms are chosen.

  - To this end, users interact with a trusted **server**.

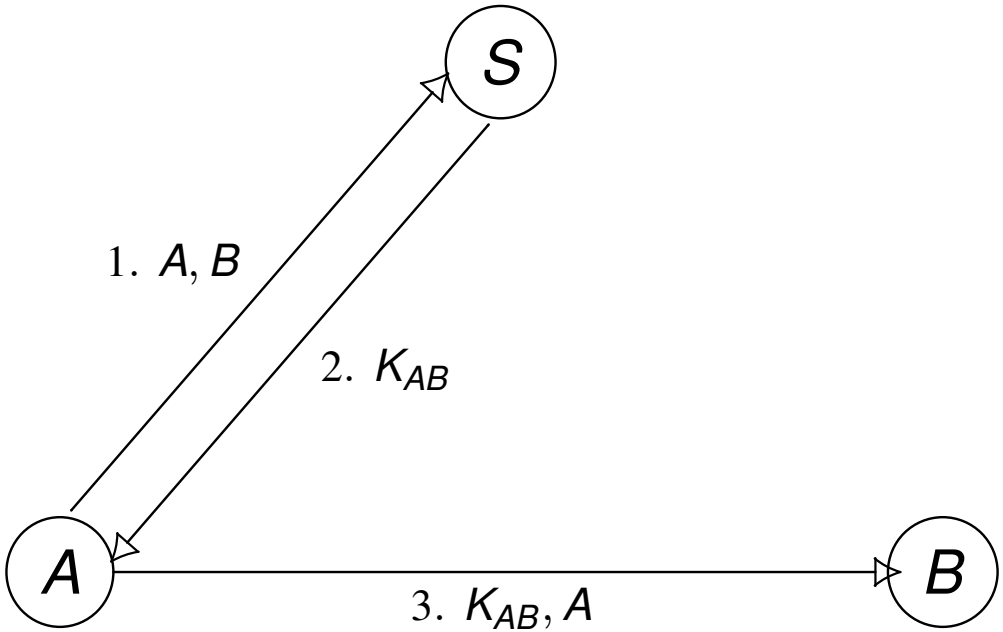
N.B.: The server is trusted not to engage in any other activity that will deliberately compromise the users' security, and to generate the new key and to do so in such a way that it is sufficiently random to prevent an attacker gaining useful information about it.

Building a key establishment protocol (2)

- Hence, protocol involves 3 **principals** (a.k.a. **agents**, **parties**...):
  - *Alice* (*A*), *Bob* (*B*), and the trusted *Server* (*S*).
  - *A* and *B* are often also called protocol **initiator** and **responder**.
  - Role of *S*: generate the new session key K_{AB} (= K_{BA}) and transport it to *A* and *B*.
- **Aims of the protocol:**
  - At the end of the protocol, K_{AB} should be known to both *A* and *B*, but to no other parties with the possible exception of *S*.
  - *A* and *B* should know that K_{AB} is newly generated.
- Formalization questions (that we will consider later):
  - How do we formalize the protocol steps and goals?
  - How do we formalize “knowledge”, “secrecy”, “newly”, ...?

Building a key establishment protocol (3)

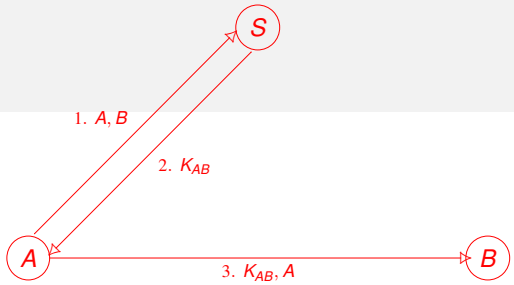
- A first attempt: a protocol that consists of 3 messages.



1. A contacts S by sending the identities of the 2 parties who are going to share the session key.
2. S sends the key K_{AB} to A.
3. A passes K_{AB} on to B.

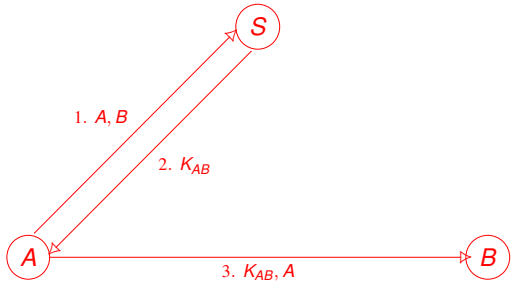
- **N.B.:** K_{AB} does contain any "information" about A and B, but is simply a name for the bit-string representing the session key.
- Before we examine the (lack of) security of this protocol, note that this is a significantly incomplete protocol specification.

Building a key establishment protocol (4)



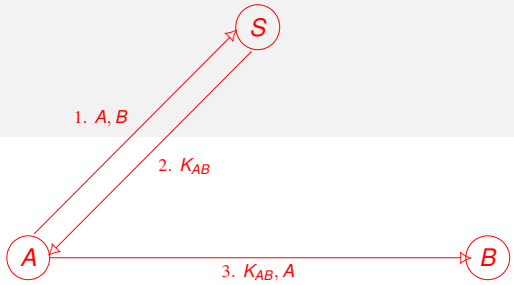
- Only messages passed in a successful protocol run are specified.
  - No description of what happens in the case that a message of the wrong format is received or that no message is received at all.
  - This is standard for security protocols, while in specifying ordinary communication protocols it is common to include this information (as it is essential to prove basic functional properties).
- No specification of internal actions of principals.
  - Internal actions are fairly obvious for many protocols (e.g. to calculate what is required to output the next message), but for other protocols there are numerous alternatives and the choice can have security-critical relevance.

Building a key establishment protocol (5)



- Implicit assumption: *A* and *B* "know" that the received messages are part of the protocol.
  - It is common to omit such details which would be required for a networked computer to be able to track the progress of a particular protocol run.
  - This may include details of which key should be used to decrypt a received message which has been encrypted.

Building a key establishment protocol (6)

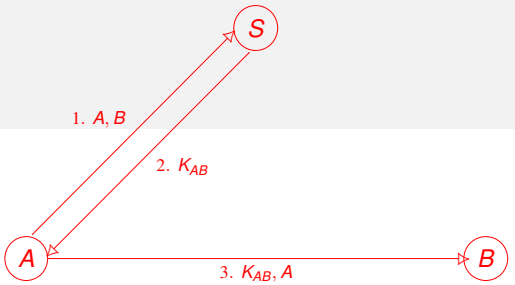


- Despite the obvious limitations associated with specifying protocols by showing only the messages of a successful run, it remains the most popular method of describing security protocols. But there are many works that have addressed these problems to "disambiguate" the notation. More in the following weeks.
- An equivalent representation: **Alice&Bob–notation**:

$$\begin{array}{lll} 1. & A \rightarrow S: & A,\ B \\ 2. & S \rightarrow A: & K_{AB} \\ 3. & A \rightarrow B: & K_{AB},\ A \end{array}$$

but note that sender/receiver names "$A \rightarrow B$" are not part of the message and it is not the case that messages automatically reach their destination securely.

Building a key establishment protocol (7)



- Problem with this protocol?
  The session key K_{AB} must be transported to A and B but to *no other parties*.
- A realistic assumption in typical communication systems such as the Internet and corporate networks:

Security Assumption 1

The adversary is able to eavesdrop on all messages sent in a security protocol.

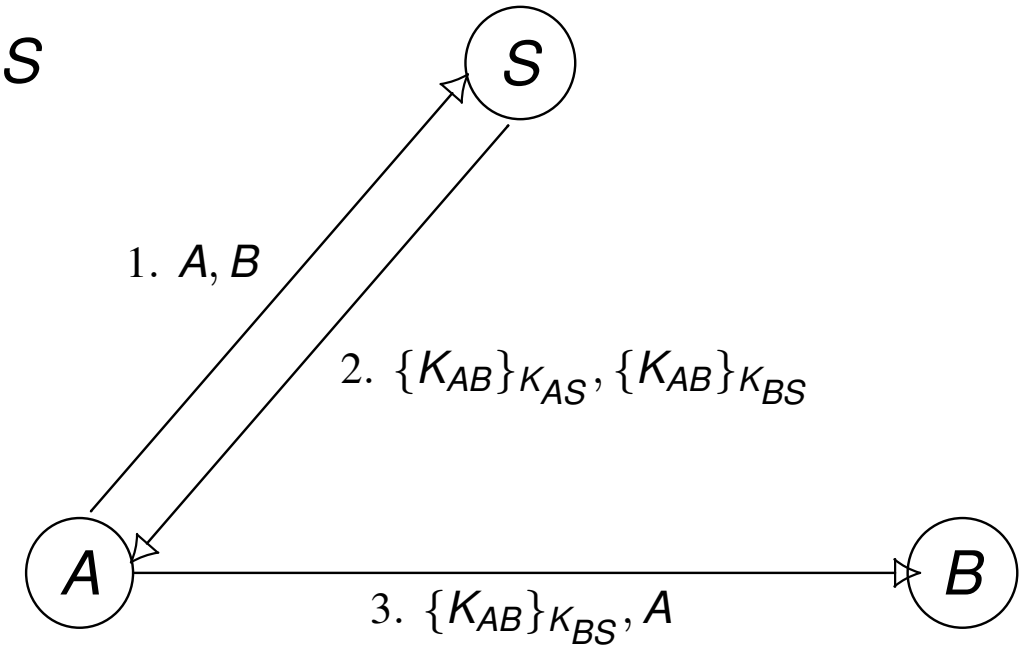
If this possibility can be discounted, then there is probably no need to apply security at all.

$\Rightarrow$ Use a cryptographic algorithm and associated keys.

Building a key establishment protocol (8)

- Second attempt: assume that S initially shares a secret key with each user of the system:
  - K_{AS} with A,
  - K_{BS} with B,

  and encrypts message 2.
- Problems with protocol?
  - Eavesdropping? No.



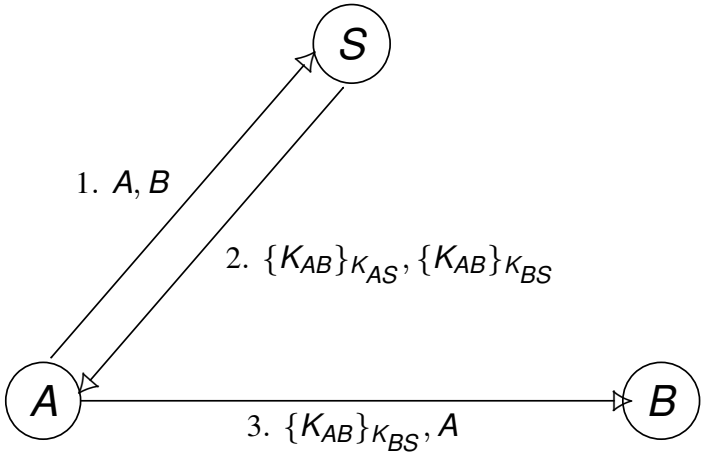
Perfect cryptography assumption

Encrypted messages may only be read by the legitimate recipients who have the keys required to decrypt...

While an understanding of cryptographic algorithms is essential for the correct design of protocols, the details of the algorithms are often irrelevant and are frequently avoided during both protocol design and analysis.

Building a key establishment protocol (9)

- Problem is not that the protocol gives away the secret key K_{AB}, but that the information about who else has K_{AB} is not protected.



- Adversary could not only be able to eavesdrop on messages sent, but also to capture messages and alter them.

Security Assumption 2

The adversary is able to alter all messages sent in a security protocol using any information available. He can also re-route any message to any other principal (or just intercept them). This includes the ability to generate and insert completely new messages.

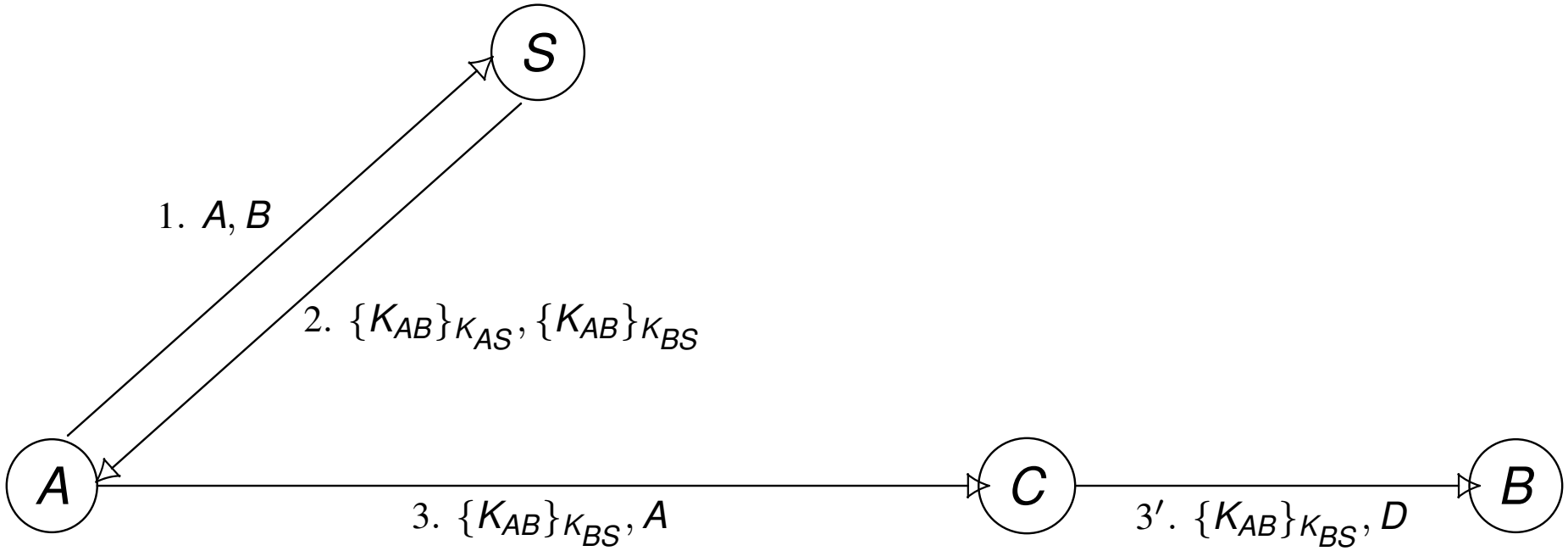
Building a key establishment protocol (10)

- In other words: the adversary has complete control of the channel(s) over which protocol messages flow.

The adversary has complete control over the network.

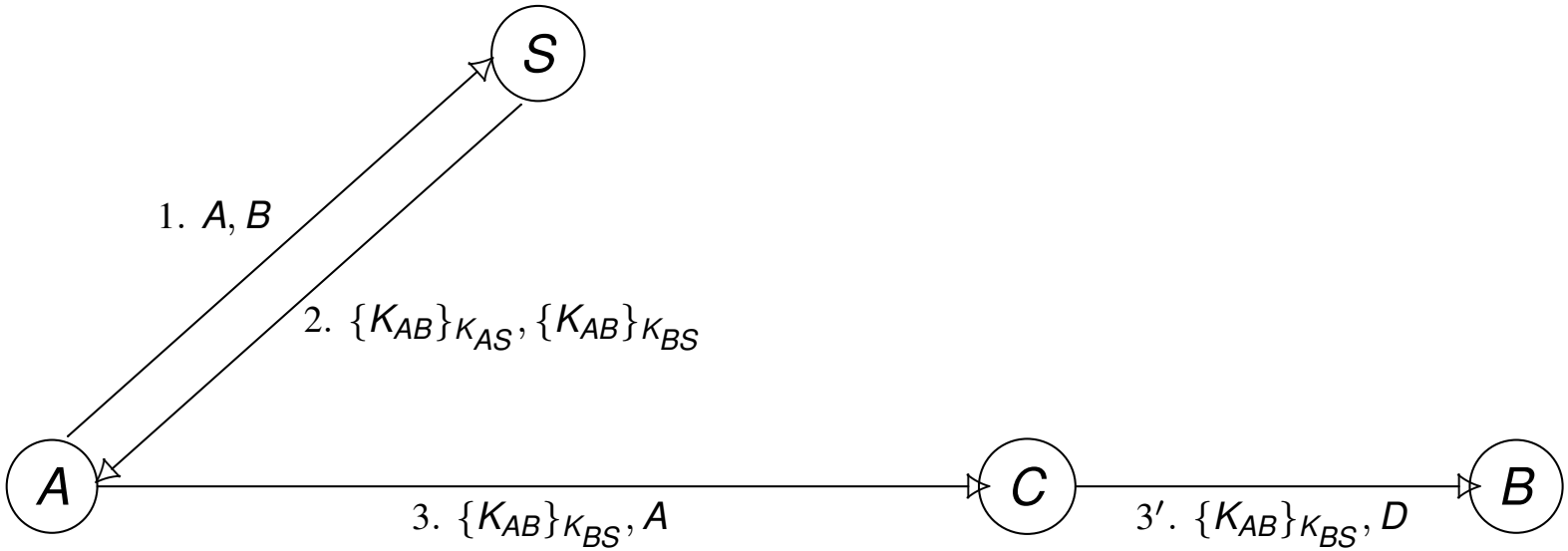
- In contrast to ordinary communication protocols, in security protocols there is an unknown and unpredictable principal involved.
  - Although there may be no more than 4 or 5 messages involved in a legitimate run of the protocol, there are an infinite number of variations in which the adversary participates.
  - These variations have an unbounded number of messages and each must satisfy the protocol's security requirements.
- The protocol can then be attacked as follows:

Building a key establishment protocol (11)



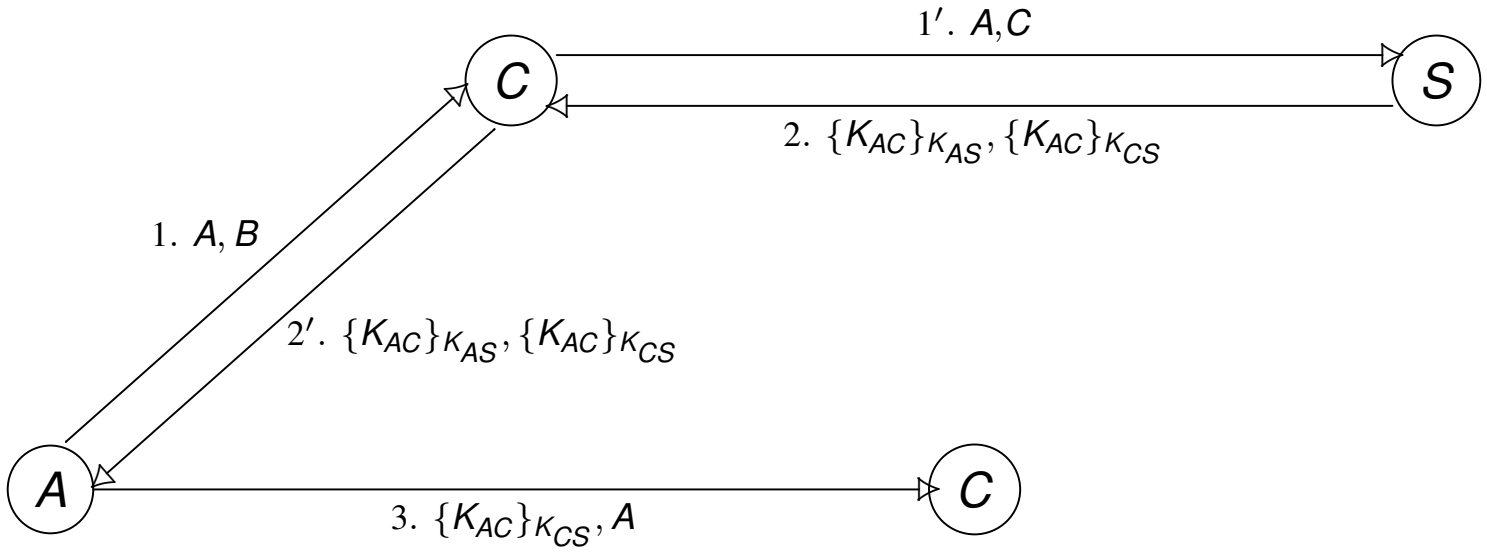
- The adversary C simply intercepts the message from A to B and substitutes D's identity for A (where D could be any identity including C's own).
  $\Longrightarrow$ B believes that he is sharing the key with D, whereas in fact he is sharing it with A.

Building a key establishment protocol (12)



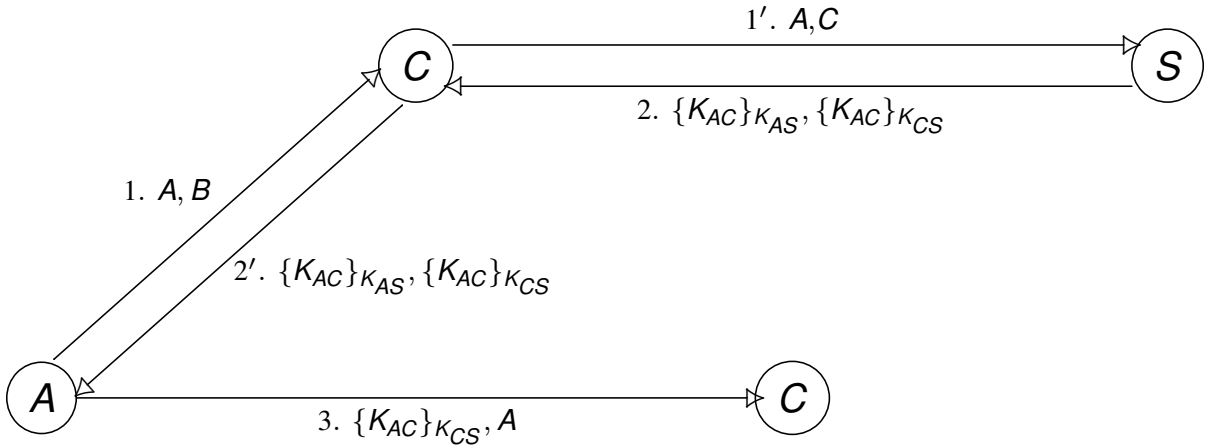
- The result of this attack will depend on the scenario in which the protocol is used, but may include such actions as B giving away information to A which should have been shared only with D.
- Although C does not obtain K_{AB}, we can still regard the protocol as broken since it does not satisfy the requirement that the users should know who else knows the session key.
- But there is also another (more serious) attack:

Building a key establishment protocol (13)



- C alters the message from A to S so that S encrypts the key K_{AC} with C's key, K_{CS}, instead of with B's key.
- Since A cannot distinguish between encrypted messages meant for other principals she will not detect the alteration. Note also that
  - K_{AC} is simply a formal name for the bit-string representing the session key, so it will be accepted by A.
  - C intercepts the message from A intended for B so that B will not detect any anomaly.

Building a key establishment protocol (14)



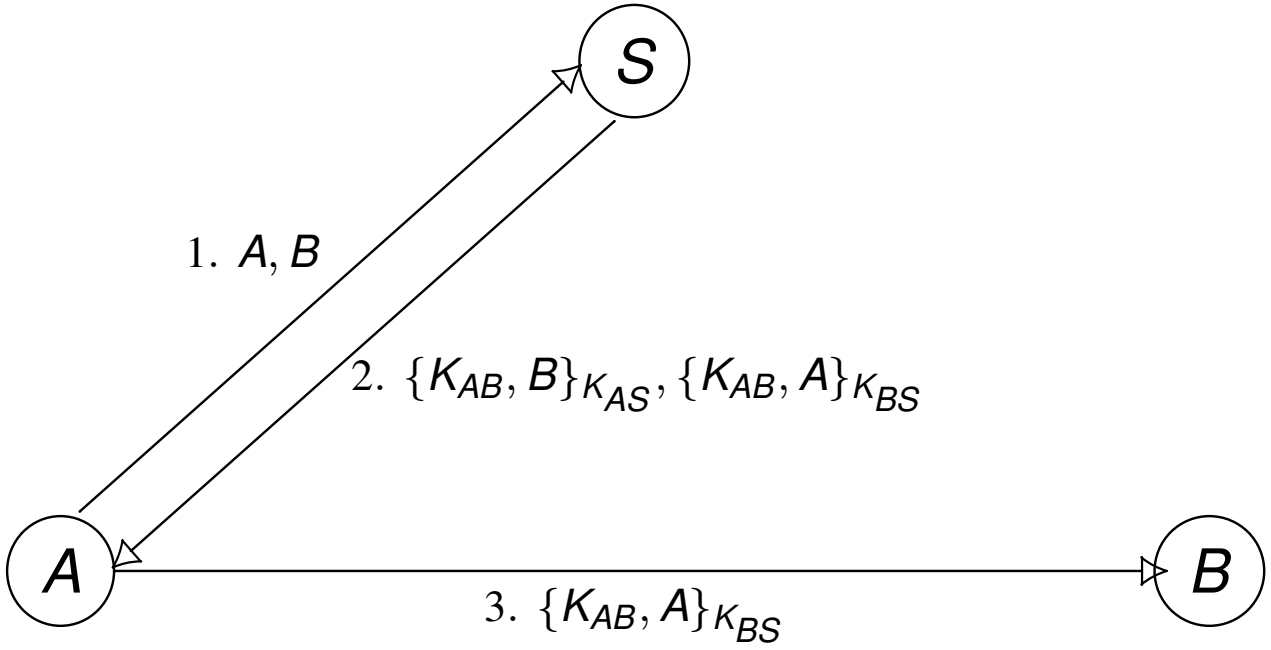
- Hence, A will believe that the protocol has been successfully completed with B whereas C knows K_{AC} and so can masquerade as B as well as learn all information that A sends intended for B.
- In contrast to the previous attack, this one will succeed if C is a legitimate system user known to S, which is a realistic assumption:

Security Assumption 3

The adversary may be a legitimate protocol participant (an insider), or an external party (an outsider), or a combination of both.

Building a key establishment protocol (15)

- To overcome this attack, the names of the principals who are to share K_{AB} need to be bound cryptographically to the key. Neither of 2 previous attacks succeeds on the following protocol:



- The protocol has improved to the point where an adversary is unable to attack it by eavesdropping or altering the messages sent between honest parties. However, even now the protocol is not good enough to provide security on normal operating conditions.

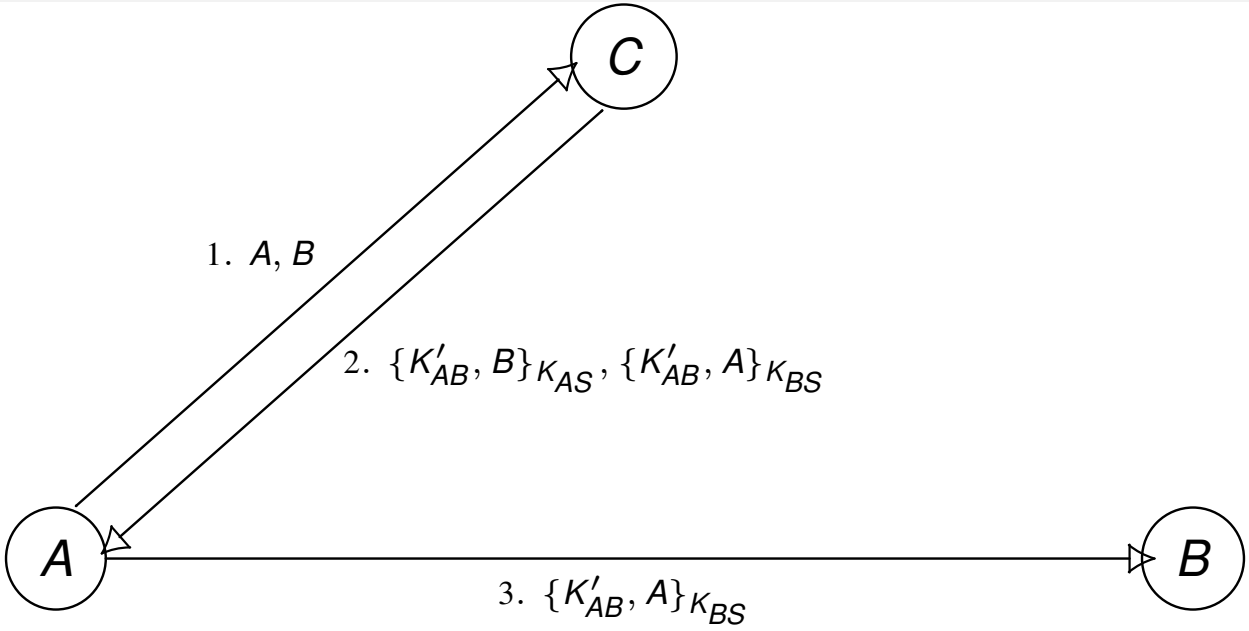
Building a key establishment protocol (16)

- The problem stems from the difference in quality between the long-term key-encrypting keys shared initially with S, and the session keys K_{AB} generated for each protocol run.
- Reasons for using session keys:
  - They are expected to be vulnerable to attack (by cryptanalysis).
  - Communications in different sessions should be separated. In particular, it should not be possible to replay messages from previous sessions.
- A whole class of attacks becomes possible when old keys (or other security-relevant data) may be **replayed** in a subsequent session.

Security Assumption 4

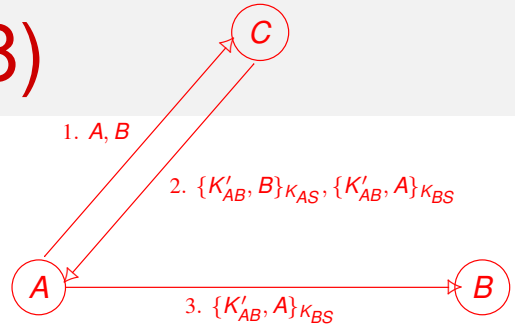
The adversary is able to obtain the value of the session key K_{AB} used in any "sufficiently old" previous run of the protocol.

Building a key establishment protocol (17)



- C intercepts message from A to S (who plays no part in this run).
- Replay: K'_{AB} is an old key used by A and B in a previous session.
  - By Security Assumption 1, C can be expected to know the encrypted messages via which K'_{AB} was transported to A and B.
  - By Security Assumption 4, C can be expected to know the value of K'_{AB}.
  - Thus, when A completes the protocol with B, C is able to decrypt subsequent information encrypted with K'_{AB} or insert or alter messages whose integrity is protected by K'_{AB}.

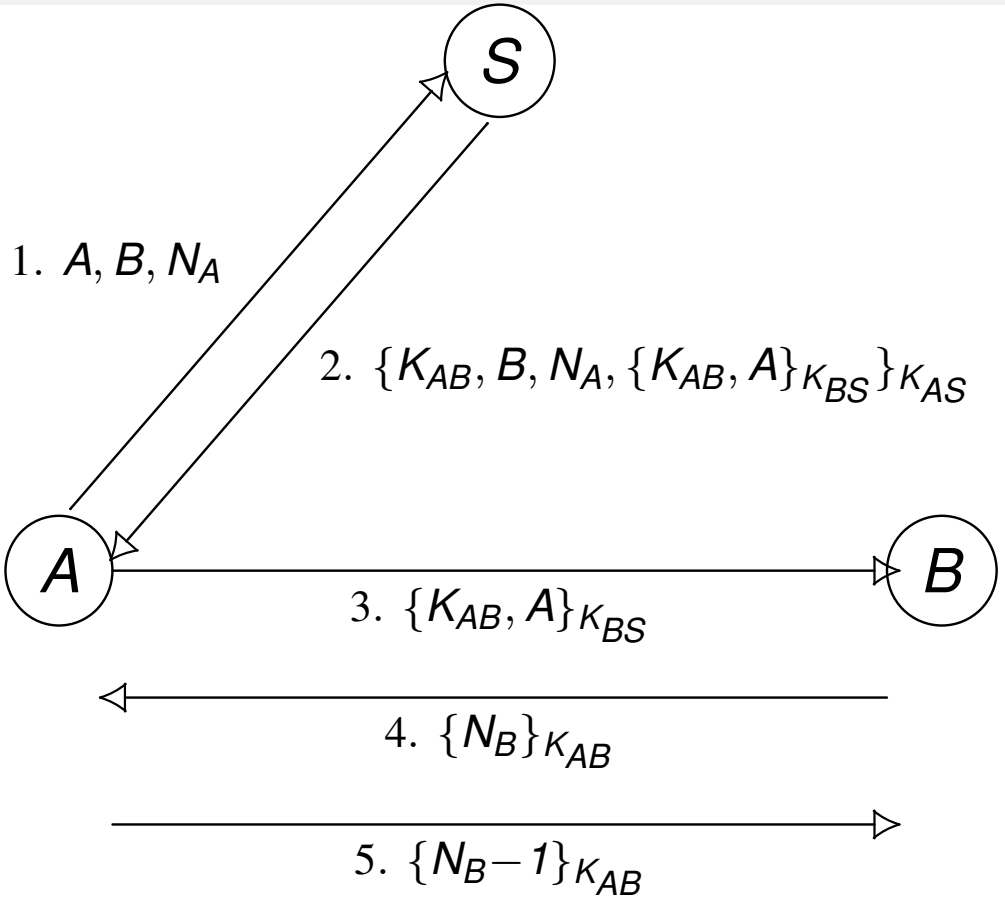
Building a key establishment protocol (18)



- The replay attack can still be regarded as successful even if C has not obtained the value of K'_{AB}.
  - C has succeeded in making A and B accept an old session key!
  - The attack allows C to replay messages protected by K'_{AB} which were sent in the previous session.
- Of course: *provided that A and B don't check the key!*
  - "Principals don't think" but just follow the protocol.
  - Various techniques may be used to allow principals to check that session keys have not been replayed, e.g. the **challenge–response** method:

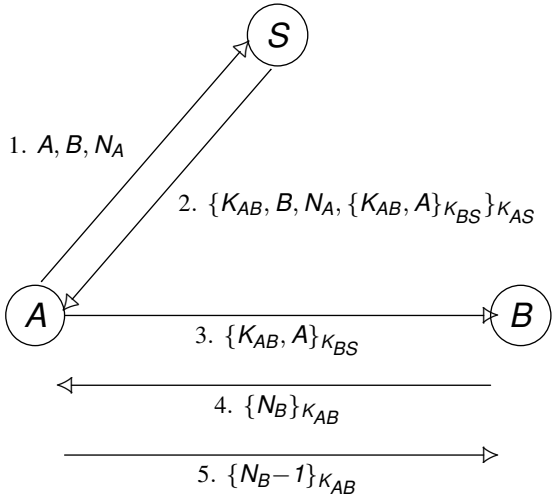
A **nonce** ("a number used only once") is a random value generated by one principal and returned to that principal to show that a message is newly generated.

Building a key establishment protocol (19)



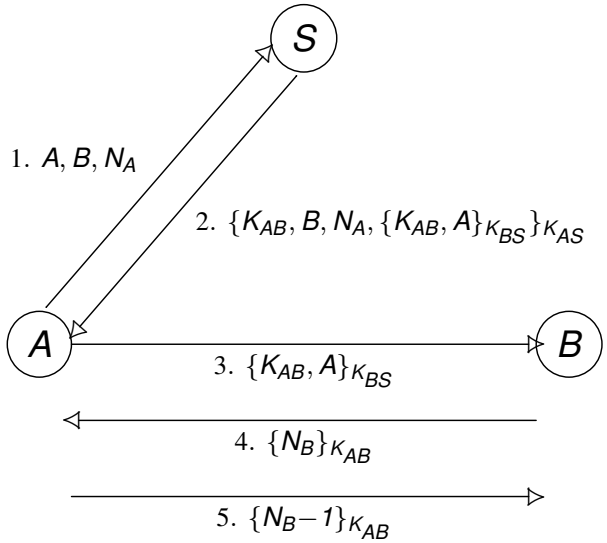
- A sends her nonce N_A to S with the request for a new key.
- If this same value is received with the session key, then A can deduce that the key has not been replayed.
  This deduction will be valid as long as session key and nonce are bound together cryptographically in such a way that only S could have formed such a message.

Building a key establishment protocol (20)



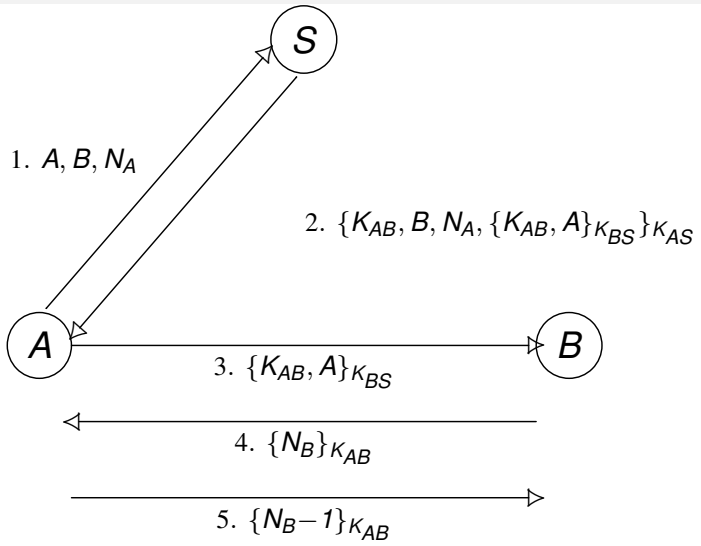
- **N.B.:** N_A is just a number!
  - There is nothing in N_A that identifies who has created it.
  - Hence, we will also write *NA* or even better N_1 or *N1*.
- If the encrypted key for *B* is included in the encrypted part of *A*'s message, then *A* can gain assurance that it is fresh.
- It is tempting to believe that *A* may pass this assurance on to *B* in an extra handshake:
  - *B* generates a nonce N_B and sends it to *A* protected by K_{AB} itself.
  - *A* uses K_{AB} to send a reply to *B* ("-1" to avoid replay of message 4).

Building a key establishment protocol (21)

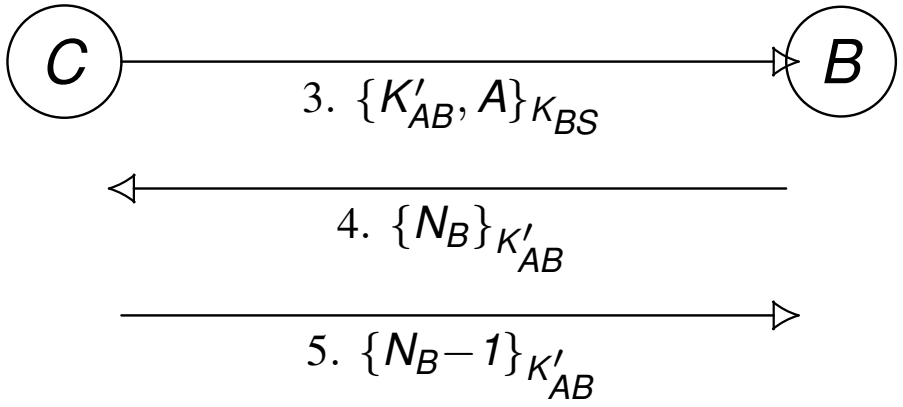


- This is actually one of the most celebrated security protocols.
  - Published by Needham and Schroeder in 1978, it has been the basis for a whole class of related protocols.
- Unfortunately, this protocol by Needham & Schroeder is vulnerable to an almost equally celebrated attack due to Denning and Sacco.
  - The argument we used to justify the protocol design is flawed.
  - Problem is assumption that only *A* will be able to form a correct reply to message 4 from *B*.

Building a key establishment protocol (22)

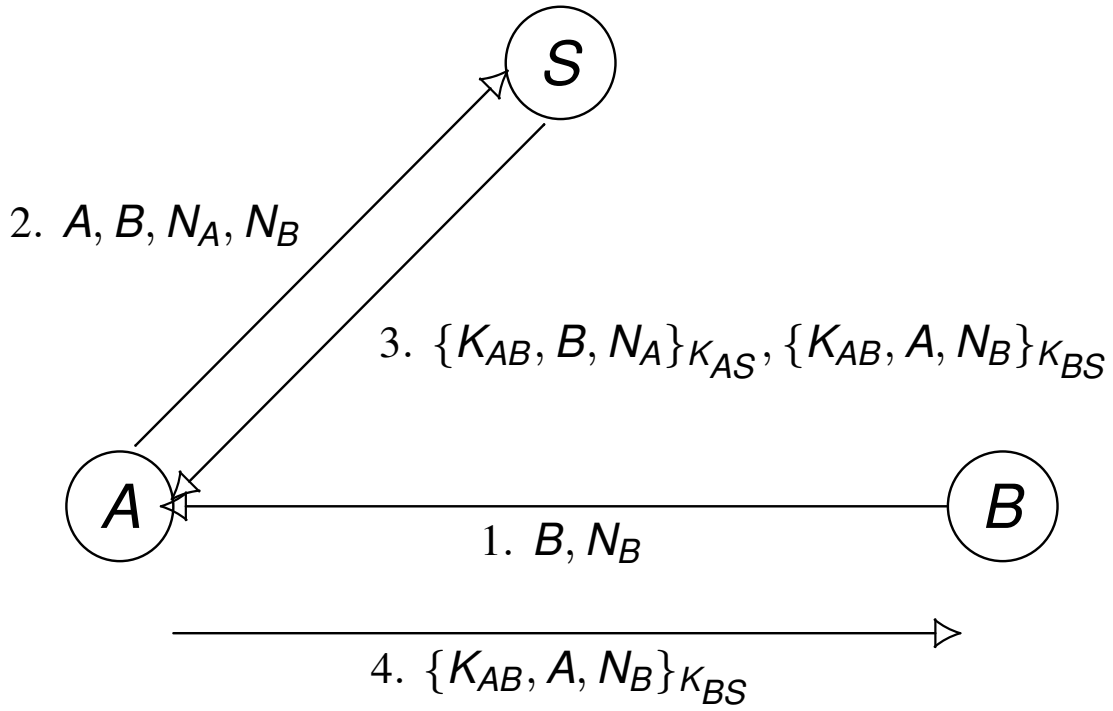


- Since the adversary C can be expected to know the value of an old session key, this assumption is unrealistic.
- C masquerades as A and thus persuades B to use old key K'_{AB}:



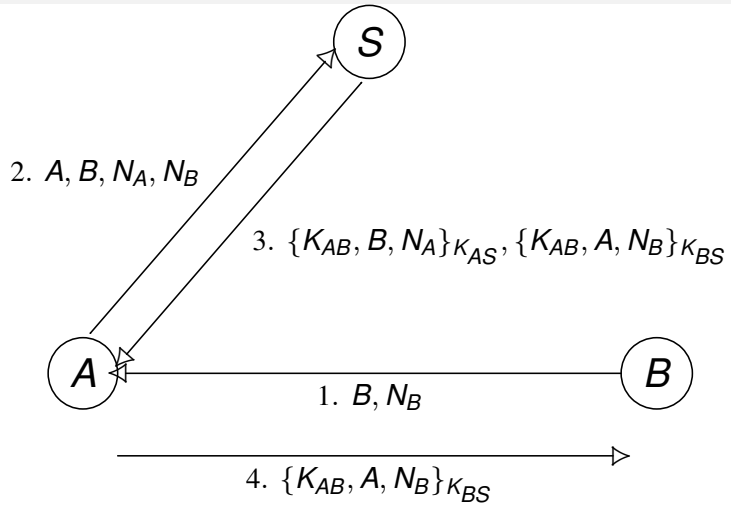
Building a key establishment protocol (23)

A different approach: throw away the assumption that it is inconvenient for both B and A to send their challenges to S.



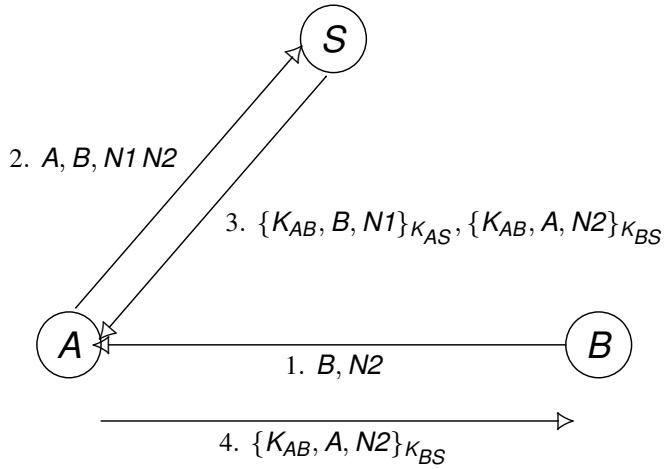
- Protocol is now initiated by B who sends his nonce N_B first to A.
- A adds her nonce N_A and sends both to S, who is now able to return K_{AB} in separate messages for A and B, which can each be verified as fresh by their respective recipients.

Building a key establishment protocol (24)



- It may seem that we have achieved more than the previous protocol using fewer messages, but in fact...
  - Before (slide “building... (18)”), A could verify not only that the key is new and known only by A, B, and S, but also that B has in fact received the key.
  - This property of **key confirmation** is achieved due to B’s use of the key in message 4, assuming that $\{N_B\}_{K_{AB}}$ cannot be formed without knowledge of K_{AB}.
  - In our final protocol, neither A nor B can deduce at the end of a successful protocol run that the other has actually received K_{AB}.

Building a key establishment protocol (25)



- This protocol avoids all the attacks that we have seen so far, as long as the cryptographic algorithm used provides the properties of both confidentiality and integrity, and the server S acts correctly.
- It would be rash to claim that this protocol is secure before giving a precise meaning to that term!
- The security of a protocol must always be considered relative to its goals.
  Hence, we need means to formalize protocols and goals (and attacks).

Table of contents I

Messages (message constructors)

- **Names**: *A*, *B* or *Alice*, *Bob*, ...
- **Keys**: K and **inverse keys** K^{-1}.
  - **Encryption**: $\{M\}_K$. Encryption with *A*'s public key: $\{M\}_{K_A}$.
  - **Signing**: $\{M\}_{K^{-1}}$. Signing with *A*'s private key: $\{M\}_{K_A^{-1}}$.
  - **Symmetric keys**: $\{M\}_{K_{AB}}$.
- **Nonces**: *NA*, *N*1, ... fresh data items used for challenge/response. N.B.: sometimes subscripts are used, e.g. N_A, but it does not mean that principals can find out that N_A is related to (e.g., was generated by) *A*.
- **Timestamps**: T. Denote time, e.g., used for key expiration.
- **Message concatenation**: M_1, M_2.

Example of a message: $\{A, T_1, K_{AB}\}_{K_B}$.

Communication

- Fundamental event is communication between principals.

$$A \rightarrow B : \{A, T_1, K_{AB}\}_{K_B}$$

- A and B name **roles**.
  Can be instantiated by any principal playing in the role.
- Communication is asynchronous (depending on semantic model).
- Sender/receiver names "$A \rightarrow B$" are not part of the message.
- Protocol specifies actions of principals.
  Equivalently, protocol defines a set of event sequences (traces).

Protocols

A typical protocol description combines prose, data type specifications, various kinds of diagrams, ad hoc notations, and message sequences like

$$
\begin{array}{lll}
1. & A \rightarrow B: & \{N1, A\}_{K_B} \\
2. & B \rightarrow A: & \{N1, N2\}_{K_A} \\
3. & A \rightarrow B: & \{N2\}_{K_B}
\end{array}
$$

They often include informal statements concerning the properties of the protocol and why they should hold.

Protocols (cont.)

What does a message $A \rightarrow B : M$ actually mean?

> *We assume that an intruder can interpose a computer in all communication paths, and thus can alter or copy parts of messages, replay messages, or emit false material.*
> *We also assume that each principal has a secure environment in which to compute such as is provided by a personal computer...*
>
> *Needham and Schroeder*

In what follows, we shall be more formal about our assumptions.

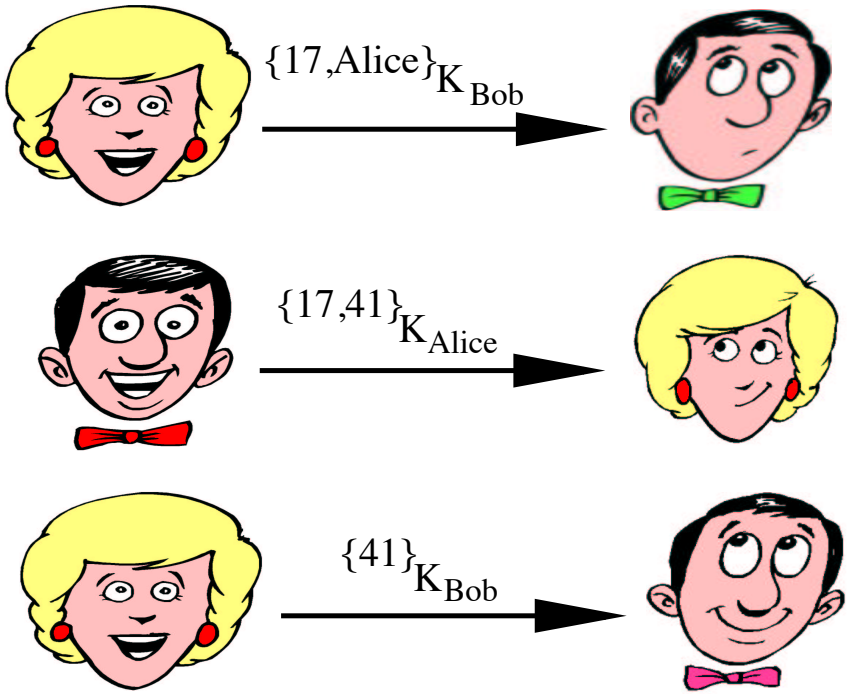
Table of contents I

An authentication protocol

The Needham-Schroeder Public Key protocol (NSPK):

$$
\begin{array}{lll}
1. & A \rightarrow B: & \{NA, A\}_{K_B} \\
2. & B \rightarrow A: & \{NA, NB\}_{K_A} \\
3. & A \rightarrow B: & \{NB\}_{K_B}
\end{array}
$$

Here is an instance (a protocol run):



How the protocol is executed

1. $A \to B : \quad \{NA, A\}_{K_B}$
2. $B \to A : \quad \{NA, NB\}_{K_A}$
3. $A \to B : \quad \{NB\}_{K_B}$

Each principal executes a "protocol automaton", e.g., Alice in role A.

- **State** s_1:
  - Generate nonce *NAlice*, pair it with name, and encrypt with K_{Bob}.
  - Send $\{NAlice, Alice\}_{K_{Bob}}$ to *Bob*.
  - Goto state s_2.
- **State** s_2:
  - Receive message C and decrypt it: $M = D_{K^{-1}_{Alice}}(C)$.
  - If M is not of the form $\{NAlice, X\}$ for some nonce X, then goto reject state else goto state s_3.
- **State** s_3: ...
- **State reject**: terminate with failure.

N.B.: principals can be engaged in multiple runs (= multiple automata).

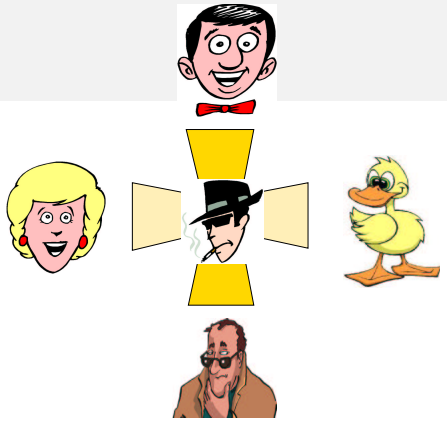
Assumptions and goals

- **Assumptions (for principals)**: Implicit (or explicit) prerequisites.
  - Principals know their private keys and public keys of others.
  - Principals can generate nonces.
- **Goals**: What the protocol should achieve. E.g.
  - **Authenticate** messages, binding them to their originator.
  - Ensure **timeliness** of messages (recent, fresh, ...).
  - Guarantee **secrecy** of certain items (e.g. generated keys).

Theses:

- A protocol without clear goals (and assumptions) is useless.
- A protocol without a proof of correctness is probably wrong.

Assumptions: attacker



How do we model the attacker? Possibilities:

- He knows the protocol but cannot break cryptography. (Standard: **perfect encryption**.)
- He is **passive** but overhears all communications.
- He is **active** and can intercept and generate messages.

  "Transfer $20 to Diane" $\leadsto$ "Transfer $10,000 to Charlie"
- He might even be one of the principals running the protocol!

A friend's just an enemy in disguise. You can't trust nobody.
(Charles Dickens, *Oliver Twist*)

Standard attacker model

Dolev-Yao attacker (or *intruder*) model

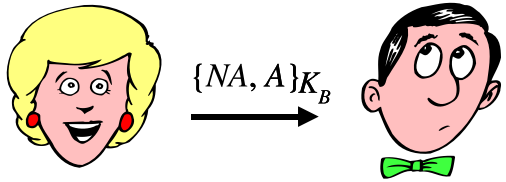
- He can intercept and read all messages.
- He can decompose messages into their parts.
  But cryptography is secure: decryption requires inverse keys.
- He can build new messages with the different constructors.
- He can send messages at any time.

These are a very strong assumptions about the attacker... so that protocols that are shown secure with respect to it will actually work in a very large range of environments.

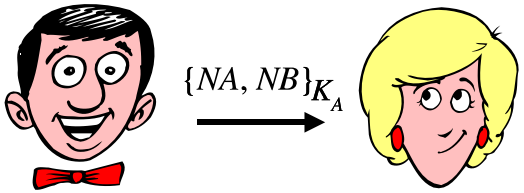
The Needham-Schroeder Public Key Protocol (NSPK)

1. $A \rightarrow B: \quad \{NA, A\}_{K_B}$
2. $B \rightarrow A: \quad \{NA, NB\}_{K_A}$
3. $A \rightarrow B: \quad \{NB\}_{K_B}$

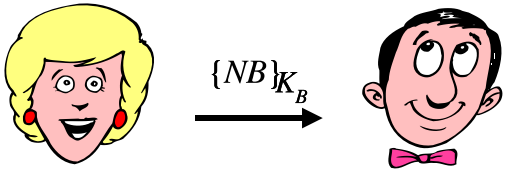
Goal: mutual authentication. Translation:



"This is Alice and I have chosen a nonce *NA*."



"Here is your nonce *NA*. Since I could read it, I must be Bob. I also have a challenge *NB* for you."

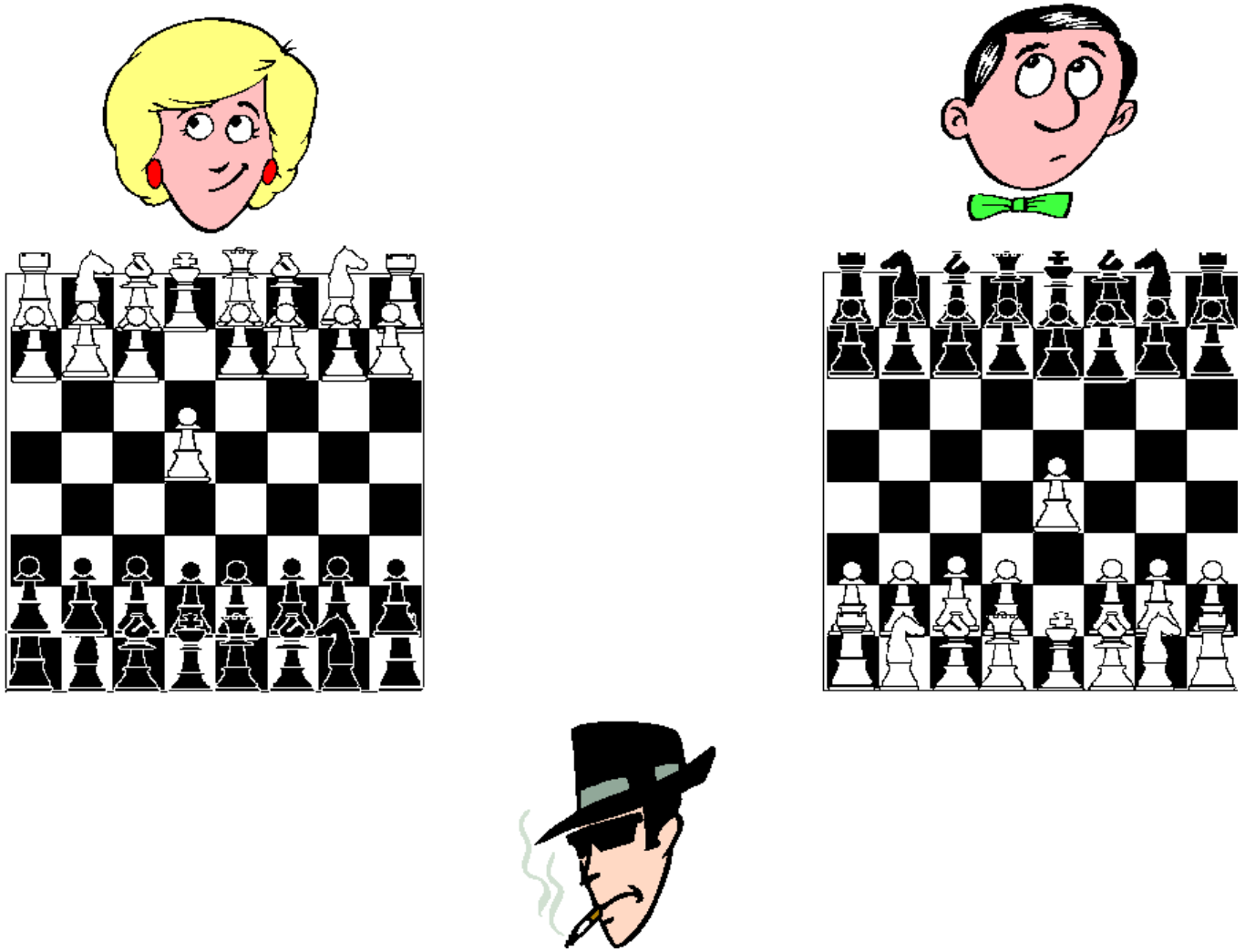


"You sent me *NB*. Since only Alice can read this and I sent it back, you must be Alice."

NSPK proposed in 1970s and used for decades, until...

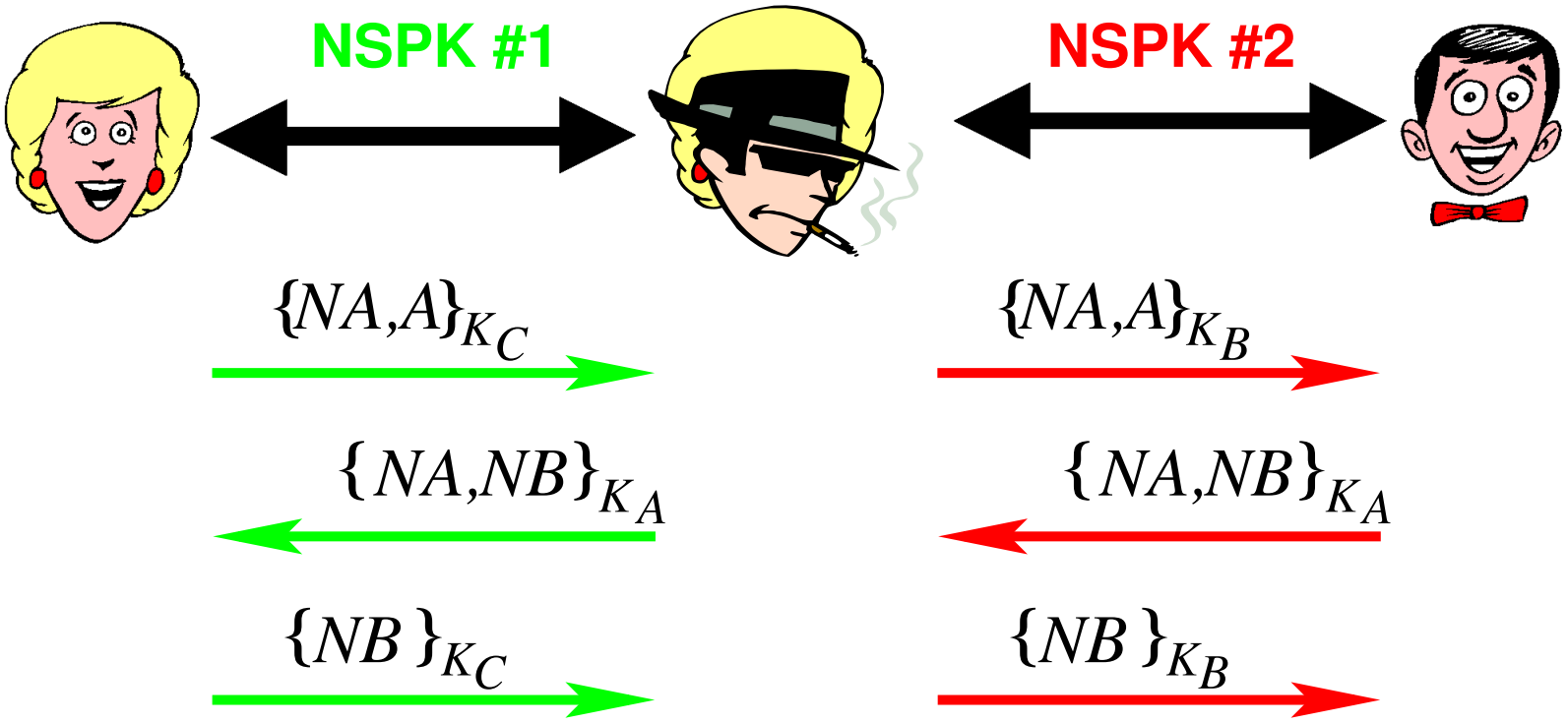
Protocols are typically small and convincing... and often wrong!

How to at least tie against a Chess Grandmaster



Man-in-the-middle attack on the NSPK

$$X \to Y : \{N1, X\}_{K_Y}$$
$$Y \to X : \{N1, N2\}_{K_X}$$
$$X \to Y : \{N2\}_{K_Y}$$



B believes he is speaking with *A*!

What went wrong?

- Problem in step 2:

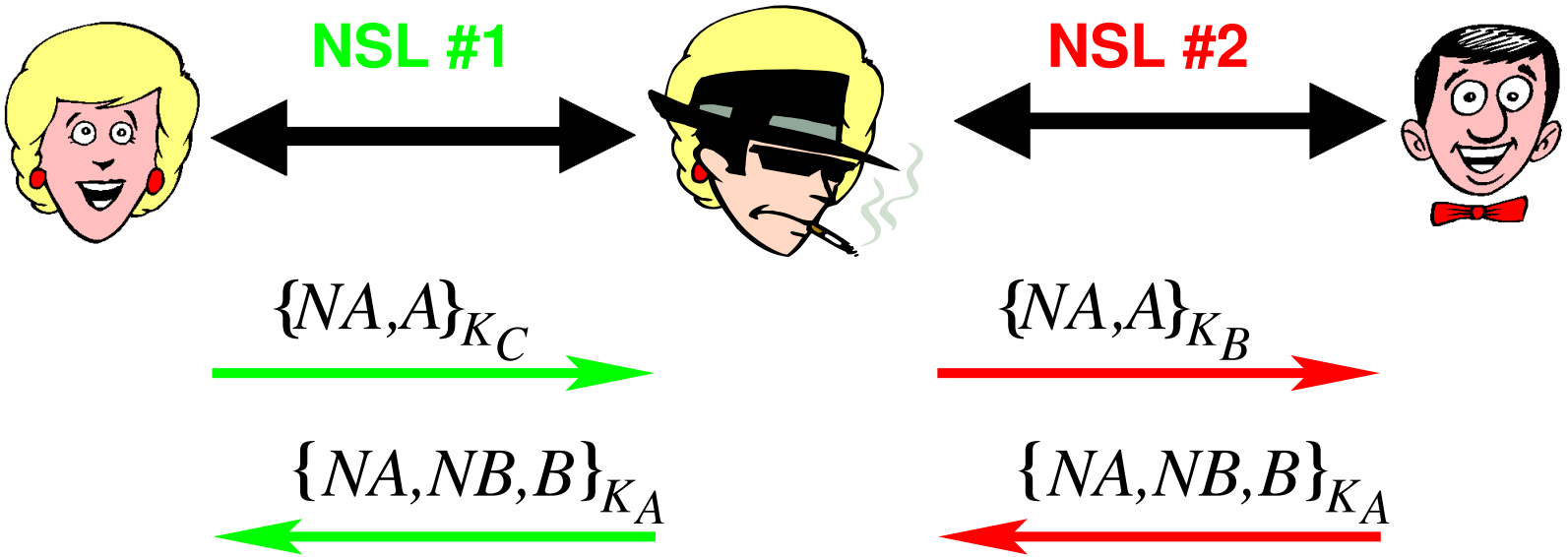
$$B \rightarrow A : \{NA, NB\}_{K_A}$$

  Agent B should also give his name: $\{NA, NB, B\}_{K_A}$.

- The improved version is called NSL protocol.
  Is the protocol now correct?

The NSL Protocol

$$X \rightarrow Y : \{N1, X\}_{K_Y}$$
$$Y \rightarrow X : \{N1, N2, Y\}_{K_X}$$
$$X \rightarrow Y : \{N2\}_{K_Y}$$



A aborts the protocol execution!
(or ignores the message)

What went wrong?

- Problem in step 2:

$$B \rightarrow A : \{NA, NB\}_{K_A}$$

  Agent B should also give his name: $\{NA, NB, B\}_{K_A}$.

- The improved version is called NSL protocol.
  Is the protocol now correct?

  Yes, it is secure against this attack but what about other kinds of attacks?
- Side-question: *Is Lowe's attack really an attack?*
  - If the goal of the protocol is *secrecy* or *authentication* then yes.
  - If the goal is only *aliveness* of the agents then no.

Summary

- Security protocols can achieve properties that cryptographic primitives alone cannot offer, e.g., authentication, secrecy, ...
- Even three liners show how difficult the art of correct design is.

  Let every eye negotiate for itself
  And trust no agent; for beauty is a witch
  Against whose charms faith melteth into blood.
  (William Shakespeare, *Much ado about nothing*)
- Informal analysis can easily miss the attacks.
- Can we raise this to a science?
- **Formal analysis of protocols is required**.
  However, formal analysis of protocols is nontrivial (even assuming perfect cryptography).
- Even more problematic: there are provably secure protocols that become insecure when combined with other protocols.