UNIVERSITÄT PASSAU

Fakultät für Informatik und Mathematik

6090: Security of Computer and Embedded Systems

Week 9: Security Protocols

Elif Bilge Kavun

elif.kavun@uni-passau.de

December 14, 2021

This Week’s Outline

- Building A Key Establishment Protocol
- Notation Used In Protocol Modeling
- Protocol Attacks

Motivation

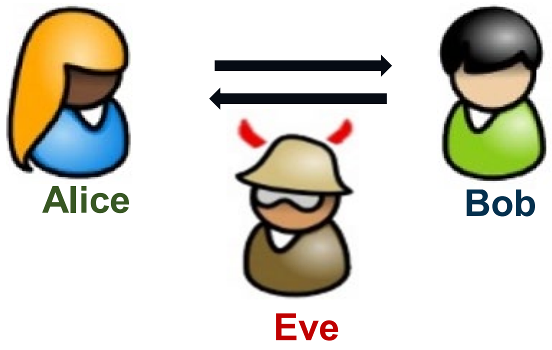
- RSA, AES, etc. provide (probably) very good cryptographic primitives
- How can we construct secure distributed applications with these primitives?
  - Securing Internet connections
  - E-commerce
  - E-banking
  - E-voting
  - Mobile communications
  - Digital contract signing

We will learn that:

Even if cryptography is hard to break, constructing these is not a trivial task!

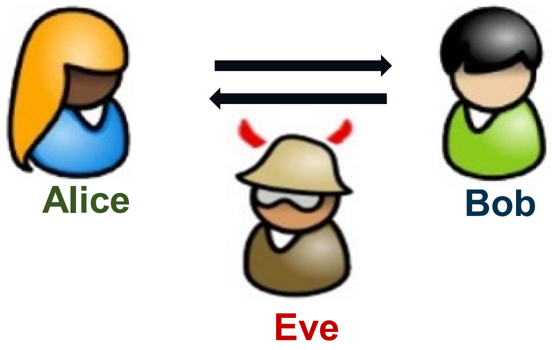
Establishing An Authentic Channel: NSPK

Alice wants to be sure that she talks to Bob
(authenticity)

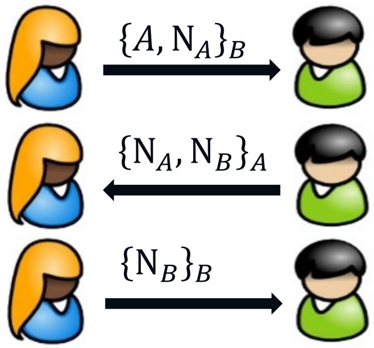


Establishing An Authentic Channel: NSPK

Alice wants to be sure that she talks to Bob
<u>(authenticity)</u>



Needham and Schroeder proposed in 1978 the following protocol (NSPK)



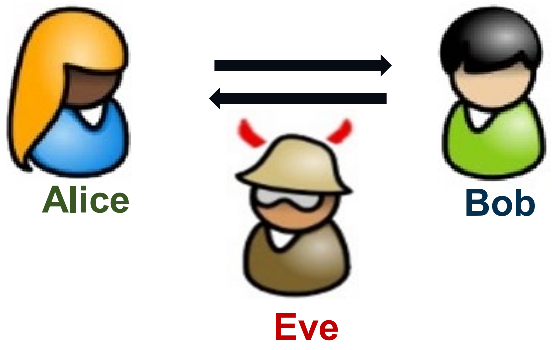
"I am Alice and here is my **nonce** N_A"

"Here is your **nonce** N_A and I have one for you"

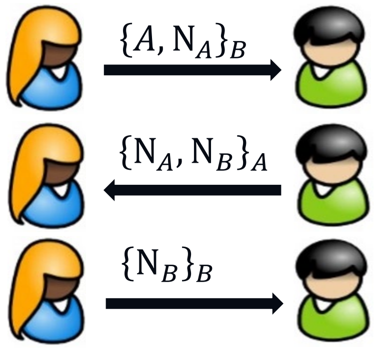
"I got it, it is N_B"

Establishing An Authentic Channel: NSPK

Alice wants to be sure that she talks to Bob (authenticity)



Needham and Schroeder proposed in 1978 the following protocol (NSPK)



"I am Alice and here is my **nonce** N_A"

"Here is your **nonce** N_A and I have one for you"

"I got it, it is N_B"

* A **nonce** (short for: "number once") is a fresh secret only known to the person generating it

NSPK: Correctness

Goal: *Mutual Authenticity (Two-way Authentication)*
after executing the protocol successfully

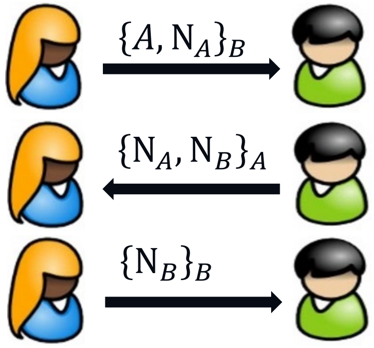
Alice and Bob can be sure to talk to each other (and not to somebody else)

NSPK: Correctness

Goal: *Mutual Authenticity (Two-way Authentication)*
after executing the protocol successfully

Alice and Bob can be sure to talk to each other (and not to somebody else)

Correctness argument (informal):



"This is Alice and I have chosen a nonce N_A"

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

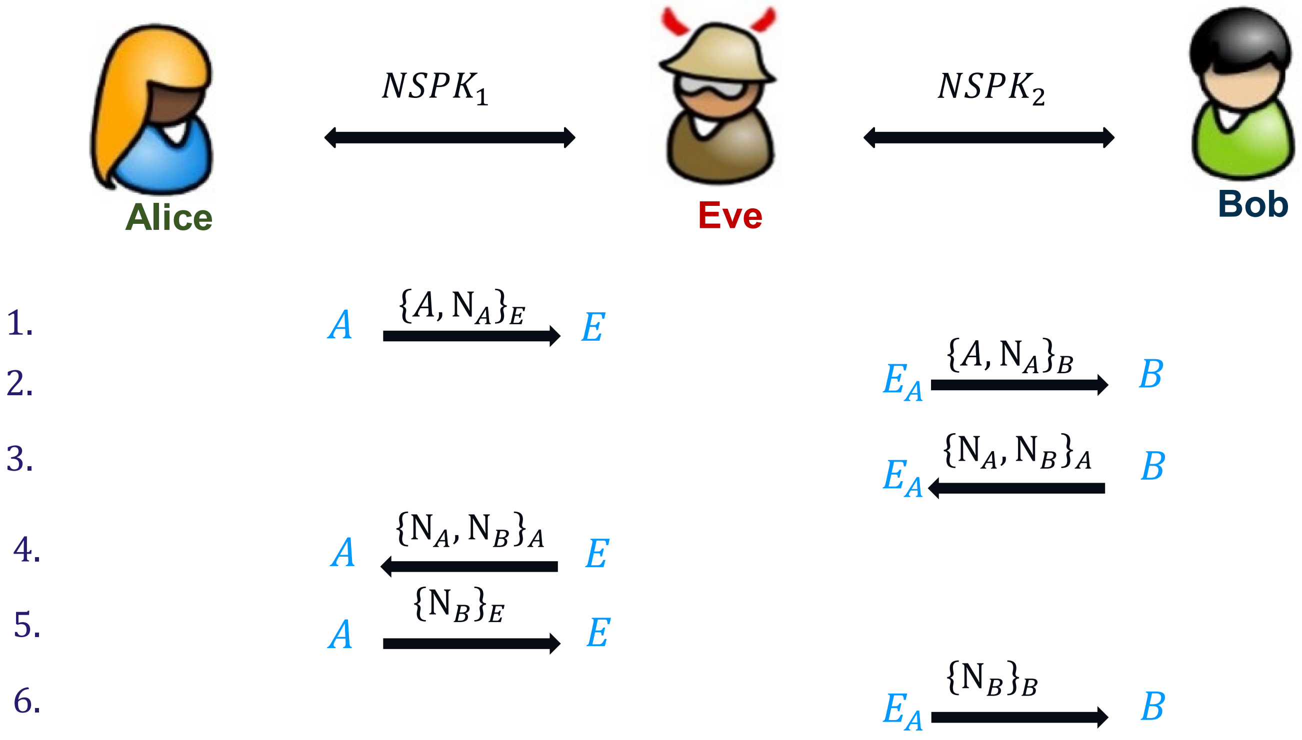
" You sent me N_B.
Since only Alice can read this and she sent it back, you must be Alice"

NSPK: Lowe’s Attack (1996)

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

NSPK: Lowe's Attack (1996)

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



NSPK: Lowe’s Attack (1996)

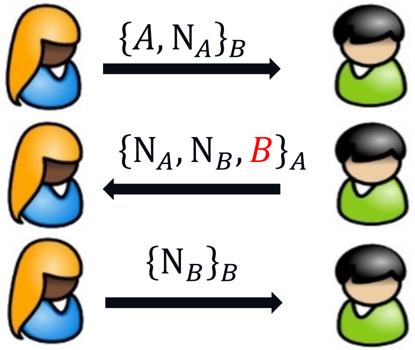
Protocols are typically *small* and *convincing* and often wrong!

1. $A \xrightarrow{\{A, N_A\}_E} E$
2. $E_A \xrightarrow{\{A, N_A\}_B} B$
3. $E_A \xleftarrow{\{N_A, N_B\}_A} B$
4. $A \xleftarrow{\{N_A, N_B\}_A} E$
5. $A \xrightarrow{\{N_B\}_E} E$
6. $E_A \xrightarrow{\{N_B\}_B} B$

Bob believes he is speaking with Alice (but talks to Eve)!

NSPK: Lowe's Fix

Needham-Schroeder with Lowe's fix:



"I am Alice and here is my nonce N_A"

"Here is your nonce N_A and I have one for you and I am Bob."

"I got it! It is N_B"

Definitions

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

Definitions

- A **protocol** consists of a set of rules (conventions) that determine the exchange of messages between two or more entities
  - In short, a distributed algorithm with emphasis on communication
- **Security (or cryptographic) protocols** use cryptographic mechanisms to achieve security objectives

Definitions

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

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

Questions

1. Is the protocol secure now?

2. How can we detect (and fix) such flaws?

Questions

1. Is the protocol secure now?

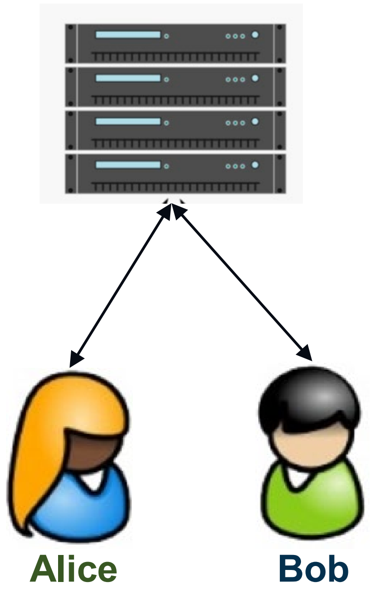
2. How can we detect (and fix) such flaws?

   - We will work on these questions in “formal analysis” as well

Building A Key Establishment Protocol

Scenario: Overview

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

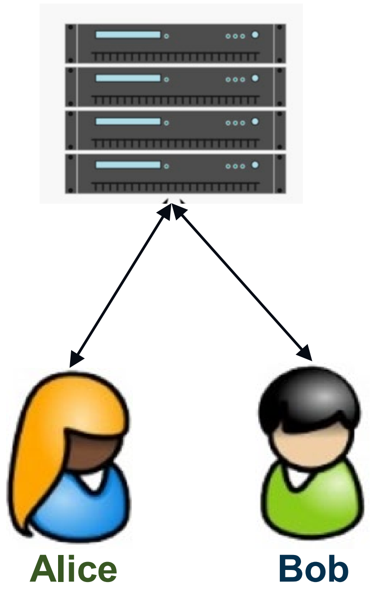


Building A Key Establishment Protocol

Scenario: Overview

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

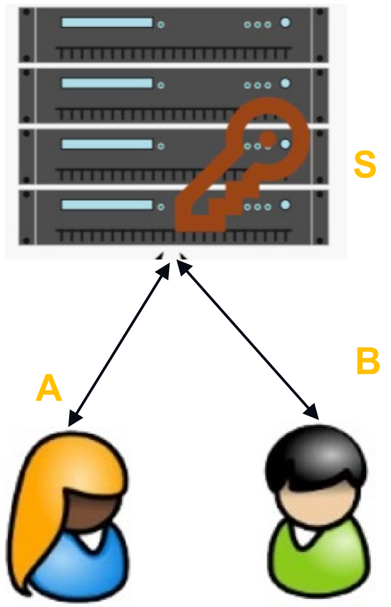
- We choose <u>a common scenario</u> (among many)
    - A set of users, any two of whom may wish to establish a new session key for subsequent secure communications
        - Successful completion of key establishment (& entity authentication) is only the beginning of a secure communications session. Further communication (often also through protocols) may be based on this key.
        - Users are not necessarily honest! (– more on this later)
    - There is <u>an honest server</u>
        - Often called "trusted server", but trust ≠ honesty! We assume that an honest server never cheats and never gives out user secrets



Building A Key Establishment Protocol

Scenario: The Details

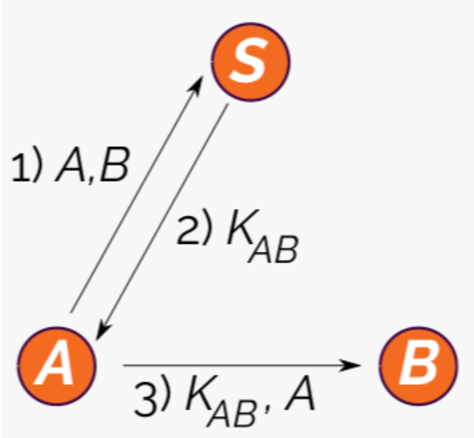
- We thus consider in this scenario protocol with three roles
  1. Initiator role *A* (Alice)
  2. Responder role *B* (Bob)
  3. Server role *S*
- In a concrete execution of a protocol, the roles are played by agents a.k.a. entities a, b, c (Charlie), s, i (Intruder), …
- We use i as the name of the intruder
  Important: No agent in our model knows that i is not honest
- Aims of the protocol
  - At the end of the protocol, K_{AB} should be known to A and B, and possibly to S (who forgets it immediately), but to no other parties
  - A and B can assume that K_{AB} is newly generated
- Formalization of the 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

First Attempt: Specification

- Our first attempt: A protocol that consists of three messages
  1. *A* contacts *S* by sending the identities of the two parties who are going to share the session key: *A*, *B*
  2. *S* sends the key K_{AB} to *A*: K_{AB}
  3. *A* passes K_{AB} on to *B*



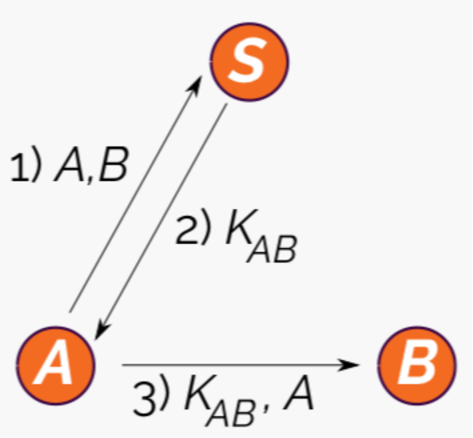
Note:

- K_{AB} does not contain any "information" about *A* or *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

First Attempt: Discussion (1/2)

- Only messages passed in a successful 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 often done for security protocols, since error messages are often not relevant for the security
    - But, what about exceptions?



- No specification of internal actions of entities

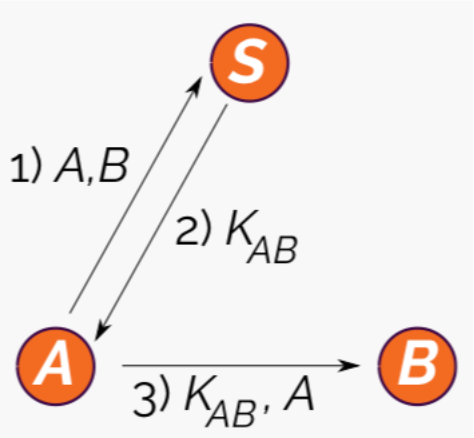
  e.g., "create fresh K_{AB}" and store that as a key for *A* and *B*

- 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

First Attempt: Discussion (2/2)

- 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 notations



- An equivalent representation: Alice & Bob notation

$$
\begin{aligned}
&1.\quad A \longrightarrow S : A, B\\
&2.\quad S \longrightarrow A : K_{AB}\\
&3.\quad A \longrightarrow B : K_{AB}, A
\end{aligned}
$$

Note that sender/receiver names (e.g., "$A \longrightarrow B$") are not part of the message and it is not the case that messages automatically reach their destination (securely)

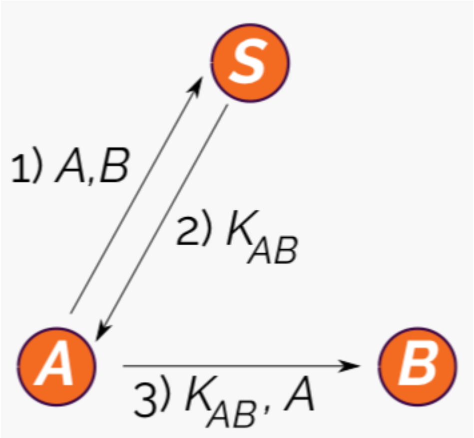
Building A Key Establishment Protocol – Block Examples

First Attempt: A First Problem

- 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 intruder is able to eavesdrop on all the messages sent in a security protocol***



Building A Key Establishment Protocol – Block Examples

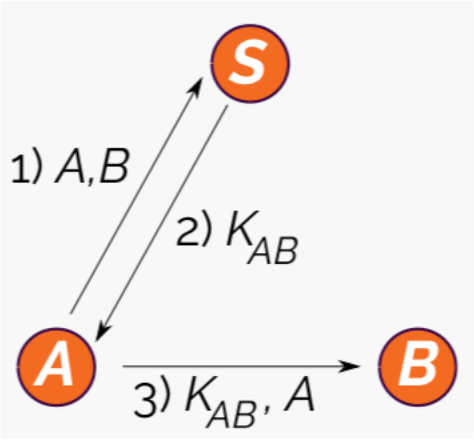
First Attempt: A First Problem

- 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 intruder is able to eavesdrop on all the messages sent in a security protocol***

→ Use a cryptographic algorithm and associated keys

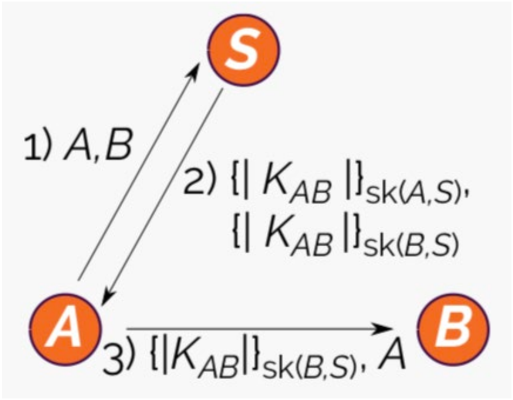


Building A Key Establishment Protocol

Second Attempt: Specification

- Second attempt
  - Assume that S initially shares a secret key $\mathrm{sk}(U, S)$ with each user U of the system
    - $\mathrm{sk}(A, S)$ with *A*
    - $\mathrm{sk}(B, S)$ with *B*

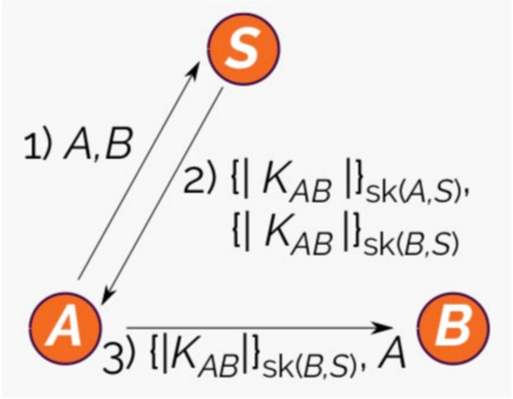
  and *S* encrypts message 2)



Building A Key Establishment Protocol
Second Attempt: Specification

- Second attempt
  - Assume that S initially shares a secret key $\mathrm{sk}(U, S)$ with each user U of the system
    - $\mathrm{sk}(A, S)$ with *A*
    - $\mathrm{sk}(B, S)$ with *B*

    and *S* encrypts message 2)



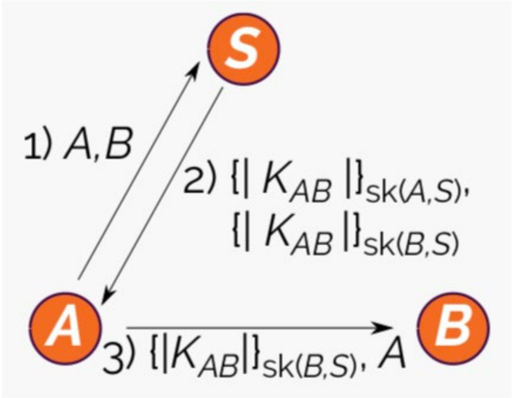
- Problems
  - Eavesdropping? No.

Building A Key Establishment Protocol

Second Attempt: Specification

- Second attempt
  - Assume that S initially shares a secret key $\mathrm{sk}(U, S)$ with each user U of the system
    - $\mathrm{sk}(A, S)$ with *A*
    - $\mathrm{sk}(B, S)$ with *B*

  and *S* encrypts message 2)



- Problems
  - Eavesdropping? No.

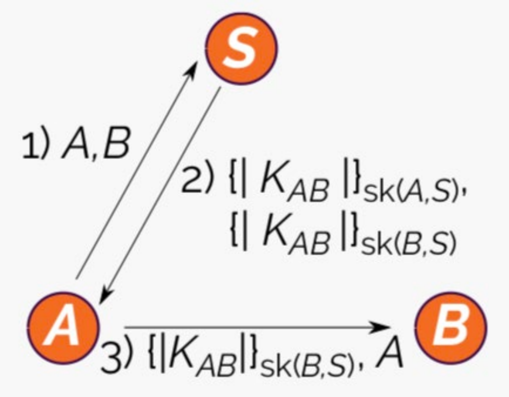
Perfect Cryptography Assumption

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

Building A Key Establishment Protocol

Second Attempt: Problems (1/2)

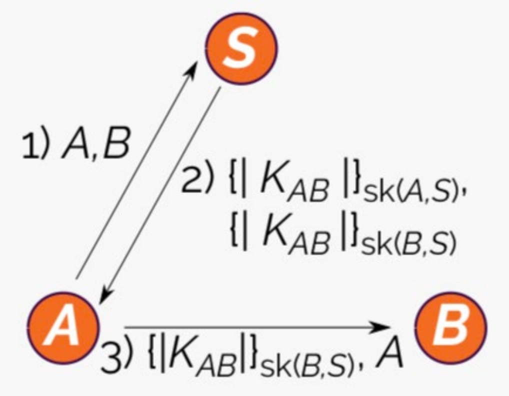
- Problem is not that the protocol gives away the secret key K_{AB}, but that the information about who else has K_{AB} is not protected
- Intruder could not only be able to eavesdrop on messages sent, but also to capture messages and alter them



Building A Key Establishment Protocol

Second Attempt: Problems (1/2)

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



Security Assumption 2

The intruder is able to intercept messages on the network and send messages to anybody (under any sender name)

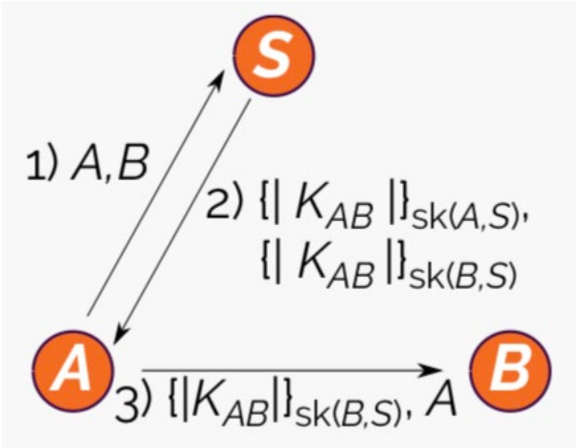
Building A Key Establishment Protocol

Second Attempt: Problems (2/2)

- The intruder has complete control of the channel(s) over which protocol messages flow

The intruder has complete control over the network

- In contrast to ordinary communication protocols, we assume the worst-case of a malicious agent
  - 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 intruder can participate
  - These variations involve an unbounded number of messages and each must satisfy the protocol's security requirements



Building A Key Establishment Protocol

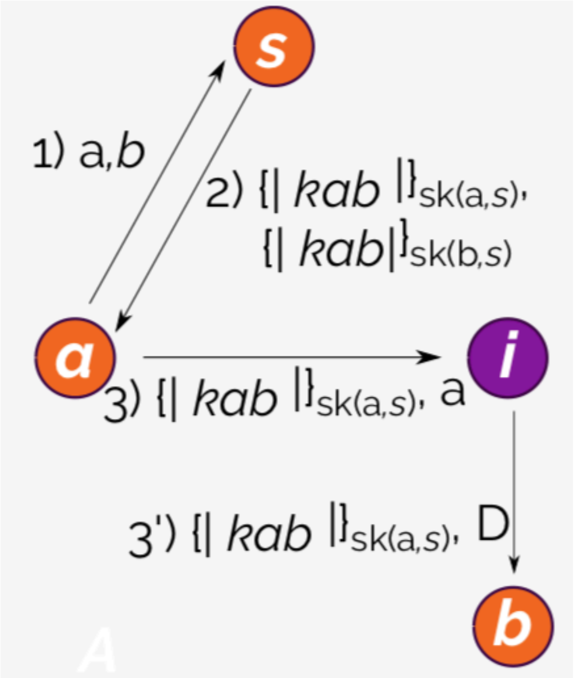
Second Attempt: A Simple Attack

- The intruder *i* simply intercepts the message from *a* to *b* and substitutes *D* for *a*'s identity (*D* is any agent name).

  => *b* believes that he is sharing the key with *a*, whereas in fact he is sharing it with *D*

- 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 *D* which should have been shared only with *a*

- Although *i* does not obtain kab, 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



Building A Key Establishment Protocol

Second Attempt: A Simple Attack

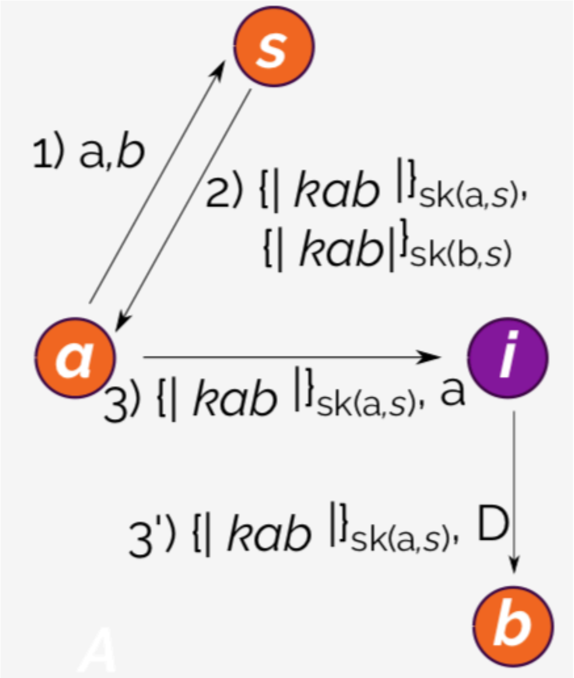
- The intruder *i* simply intercepts the message from *a* to *b* and substitutes *D* for *a*'s identity (*D* is any agent name).

  => *b* believes that he is sharing the key with *a*, whereas in fact he is sharing it with *D*

- 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 *D* which should have been shared only with *a*

- Although *i* does not obtain kab, 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

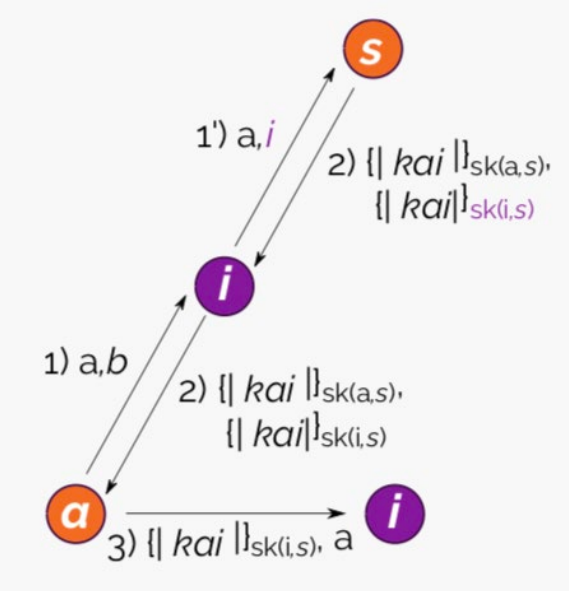
- But there is also another (more serious) attack…



Building A Key Establishment Protocol

Second Attempt: Another Attack

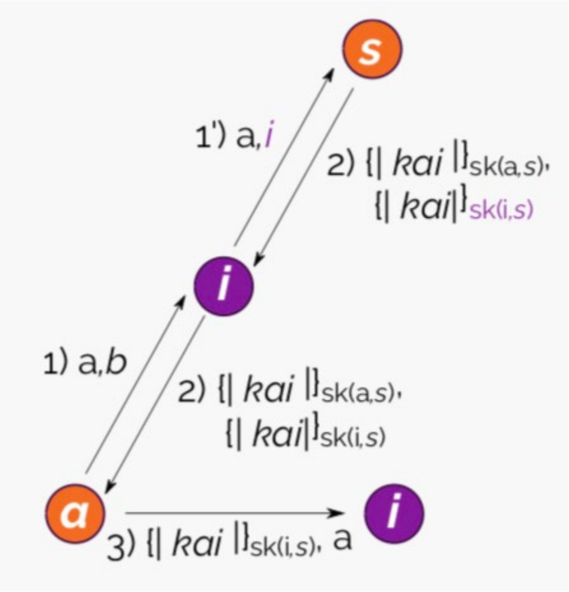
- *i* alters the message from *a* to *s* so that *s* generates *a* key kai for *a* and *i* and encrypts it with the key $\mathrm{sk}(i, s)$of the intruder
- Since *a* cannot distinguish between encrypted messages meant for other principals she will not detect the alteration
    - *kai* is simply a formal name for the bit-string representing the session key, so it will be accepted by *a*
    - *i* intercepts the message from *a* intended for *b* so that *a* will not detect any anomaly
- Hence *a* will believe that the protocol has been successfully completed with *b* whereas *i* knows *kai* 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 *i* is a *[legitimate system user]* known to *s*



Building A Key Establishment Protocol

Second Attempt: Another Attack

- *i* alters the message from *a* to *s* so that *s* generates *a* key kai for *a* and *i* and encrypts it with the key $\mathrm{sk}(i,s)$of the intruder
- Since *a* cannot distinguish between encrypted messages meant for other principals she will not detect the alteration
    - *kai* is simply a formal name for the bit-string representing the session key, so it will be accepted by *a*
    - *i* intercepts the message from *a* intended for *b* so that *a* will not detect any anomaly
- Hence *a* will believe that the protocol has been successfully completed with *b* whereas *i* knows *kai* 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 *i* is a *legitimate system user* known to *s*



Security Assumption 3

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

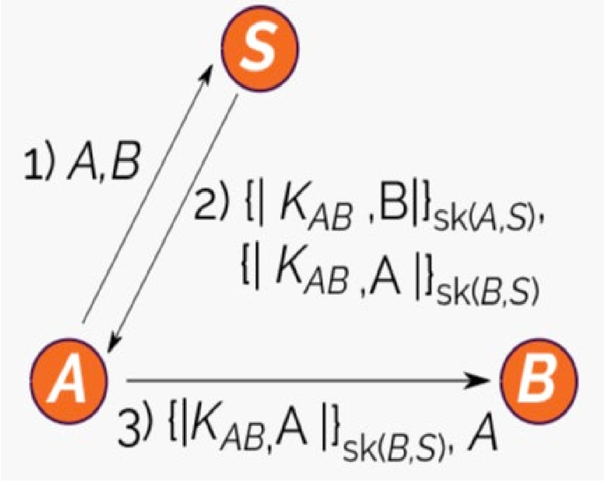
Building A Key Establishment Protocol

Third Attempt: Specification

- To overcome these attacks, the names of the principals who are to share K_{AB} need to be bound cryptographically to the key

  Neither of two previous attacks succeed on the modified protocol (see figure)

- The protocol has improved to the point where an intruder is unable to attack it by eavesdropping or altering the messages sent between honest parties



Building A Key Establishment Protocol

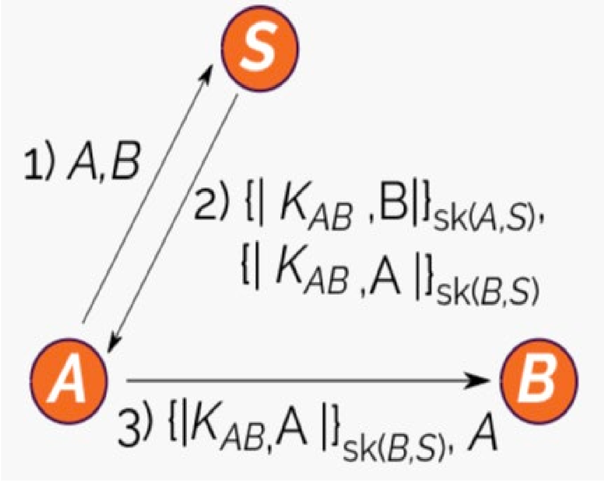
Third Attempt: Specification

- To overcome these attacks, the names of the principals who are to share K_{AB} need to be bound cryptographically to the key

  Neither of two previous attacks succeed on the modified protocol (see figure)

- The protocol has improved to the point where an intruder 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



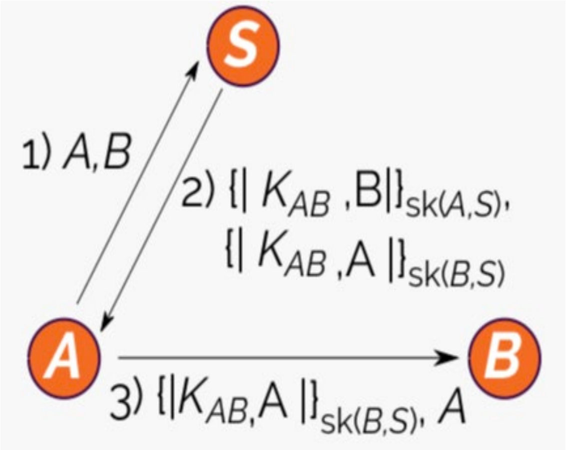
Building A Key Establishment Protocol

Third Attempt: Problem

- 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 intruder 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

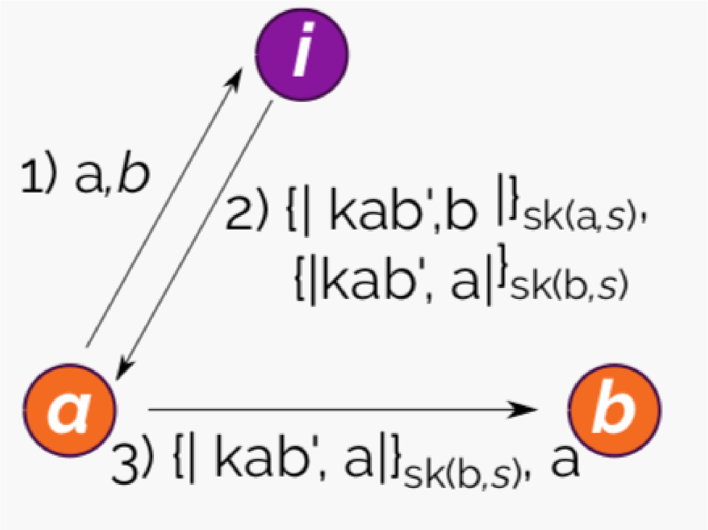
Third Attempt: Attack (1/2)

- *i* masquerades as *s*

- Replay

kab' is an old key used by *a* and *b* in a previous session

  - By Security Assumption 1, *i* can be expected to know the encrypted messages in which kab' was transported to *a* and *b*
  - By Security Assumption 4, *i* can be expected to know the value of kab'
  - Thus, when *a* completes the protocol with *b*, *i* is able to decrypt subsequent information encrypted with kab' or insert or alter messages whose integrity is protected by kab'



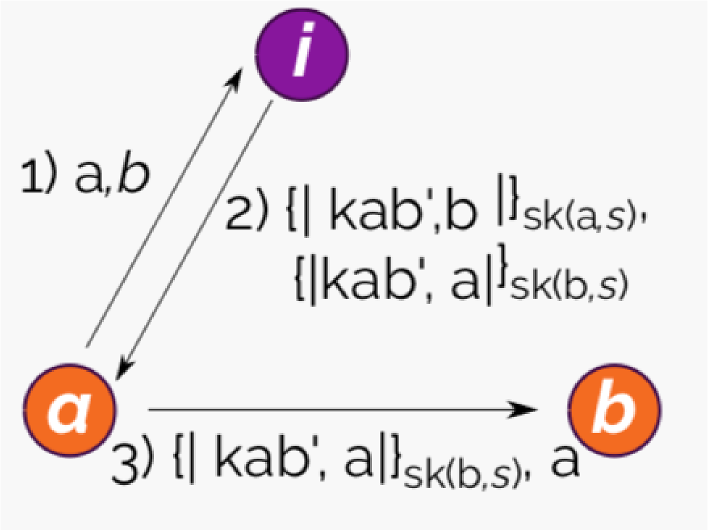
Building A Key Establishment Protocol

Third Attempt: Attack (2/2)

- The replay attack can still be regarded as successful even if *i* has not obtained the value of kab'
  - *i* has succeeded in making *a* and *b* accept an old session key!
  - The attack allows *i* to replay messages protected by kab' which were sent in the previous session

- Happens if *a* and *b* do not check the key!
  - Principals "do not 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



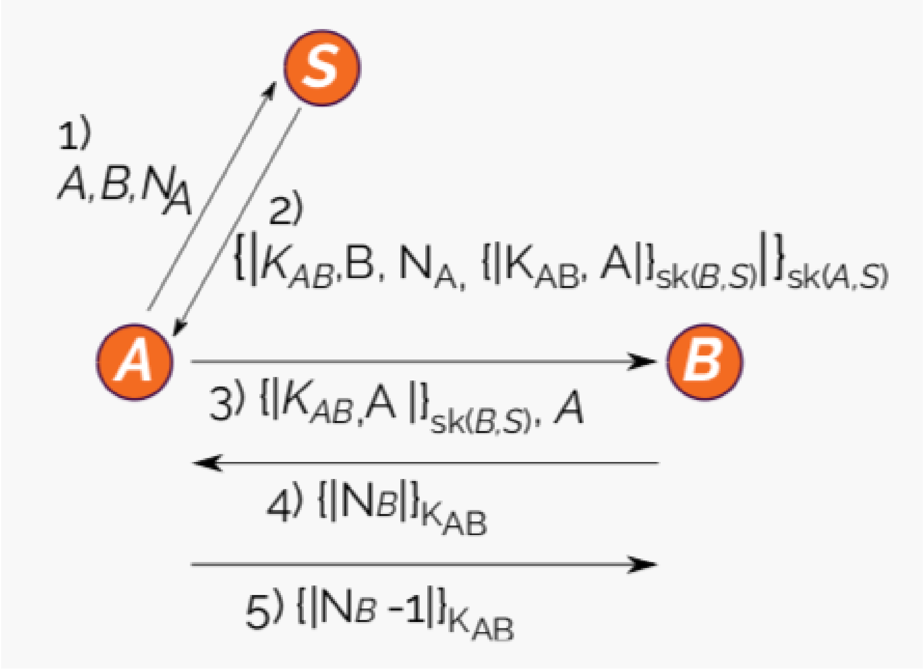
Definition:* *A* *nonce* *("a number used only once") is a random value generated by one principal and returned to that principal to show that a message is newly generated

Building A Key Establishment Protocol
Fourth Attempt (1/2)

- *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)

- Note, 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 $N1$



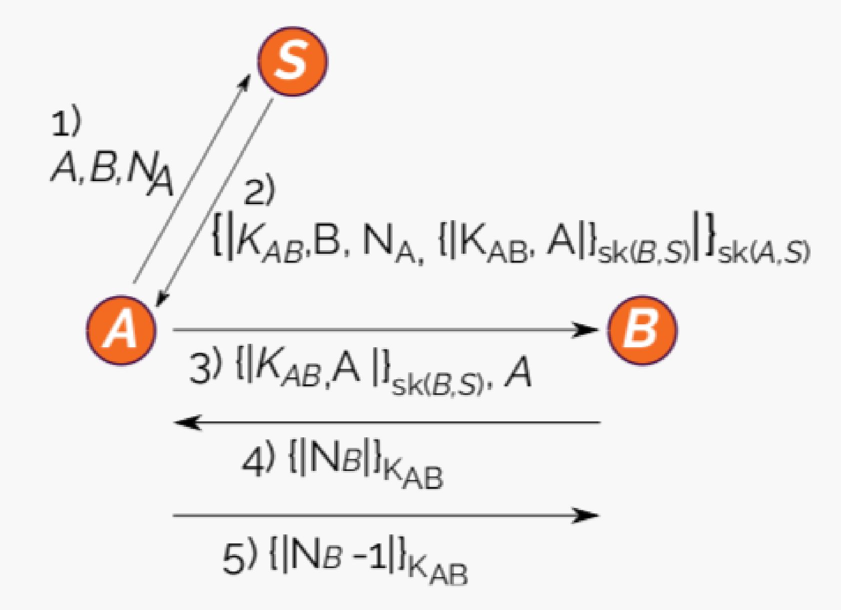
Building A Key Establishment Protocol

Fourth Attempt (2/2)

- 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 reply to *B* ("–1" to avoid replay of message 4)

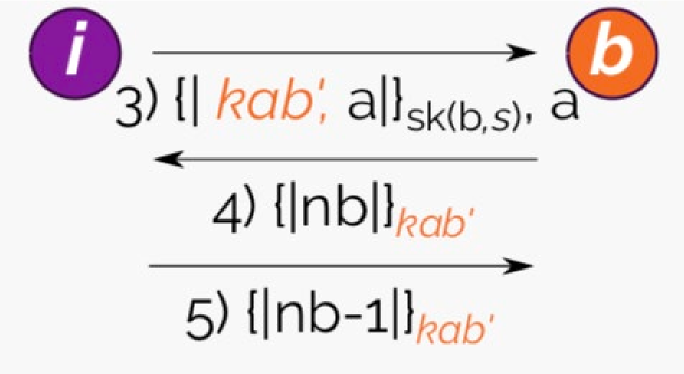
- This is actually a famous security protocol
  - Needham Schroeder with Conventional Keys (NSCK)
  - Published by Needham and Schroeder in 1978, it has been the basis for a whole class of related protocols
  - Unfortunately, this protocol is vulnerable to a famous attack as shown by Denning and Sacco

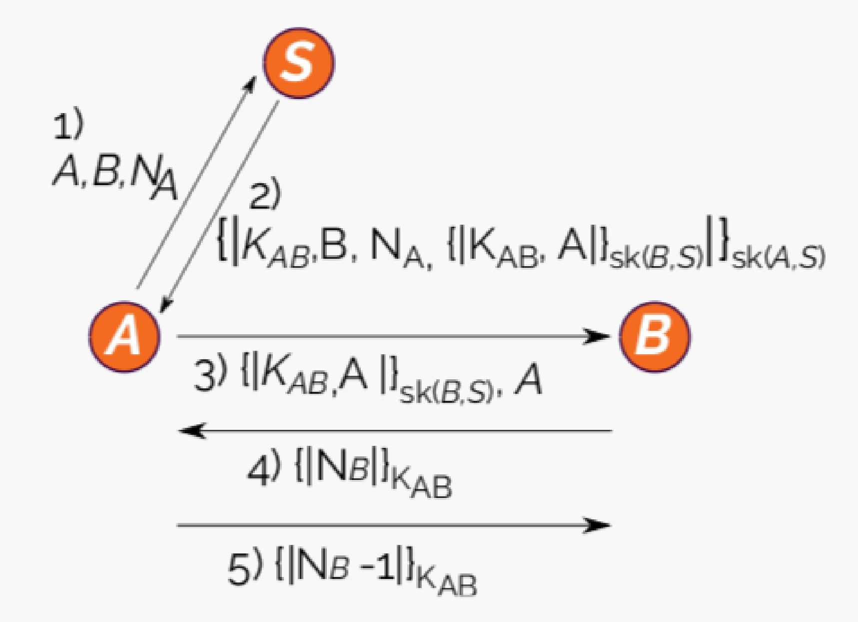


Building A Key Establishment Protocol

Fourth Attempt: Attack

- The problem is the assumption that only *A* will be able to form a correct reply to message 4 from *B*
- Since the intruder *i* can be expected to know the value of an old session key, this assumption is unrealistic
- *i* masquerades as *a* and convinces *b* to use old key kab'





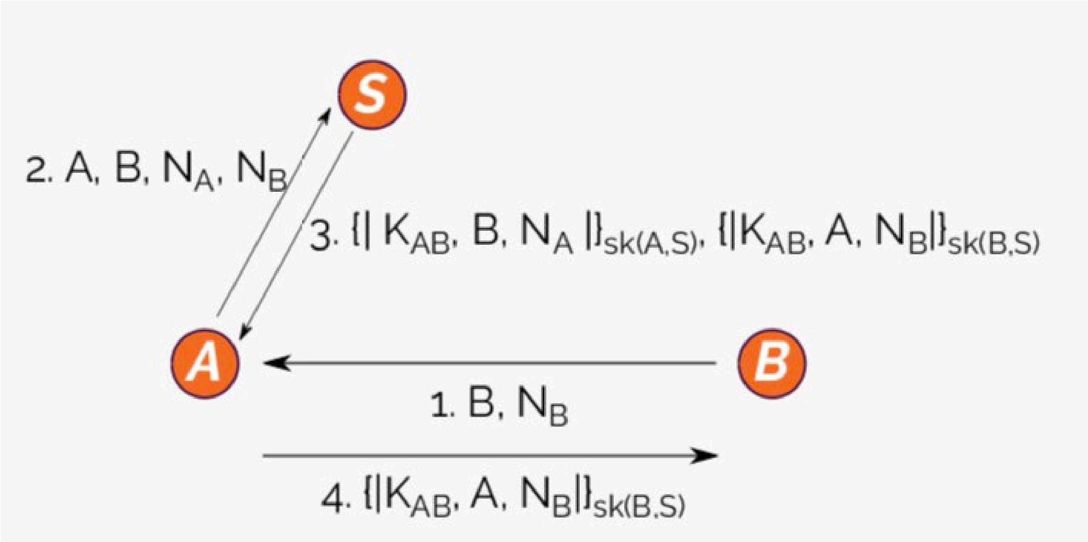
Building A Key Establishment Protocol

Fifth Attempt

- **Idea**
  Let's throw away the assumption that it is inconvenient for both *B* and *A* to send their challenges to *S*

- The protocol is now initiated by *B* who sends his nonce N_B first to *A*

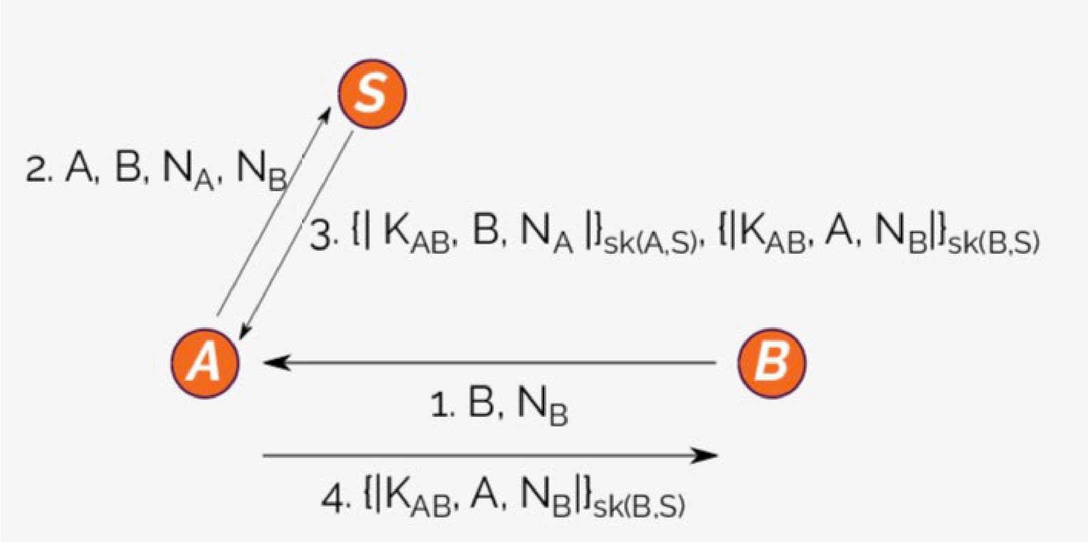
- *A* adds her nonce N_A and sends both to *S*, who is now able to return K_{AB} in separate messages for *A* and *B*, which can each be verified as fresh by their respective recipients



Building A Key Establishment Protocol

Fifth Attempt: Observations

- It may seem that we have achieved more than the previous protocol using fewer messages, but in fact…
  - In the NSCK, *A* could verify 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 $\{|\mathrm{N}_B|\}_{K_{AB}}$ cannot be formed without the 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}



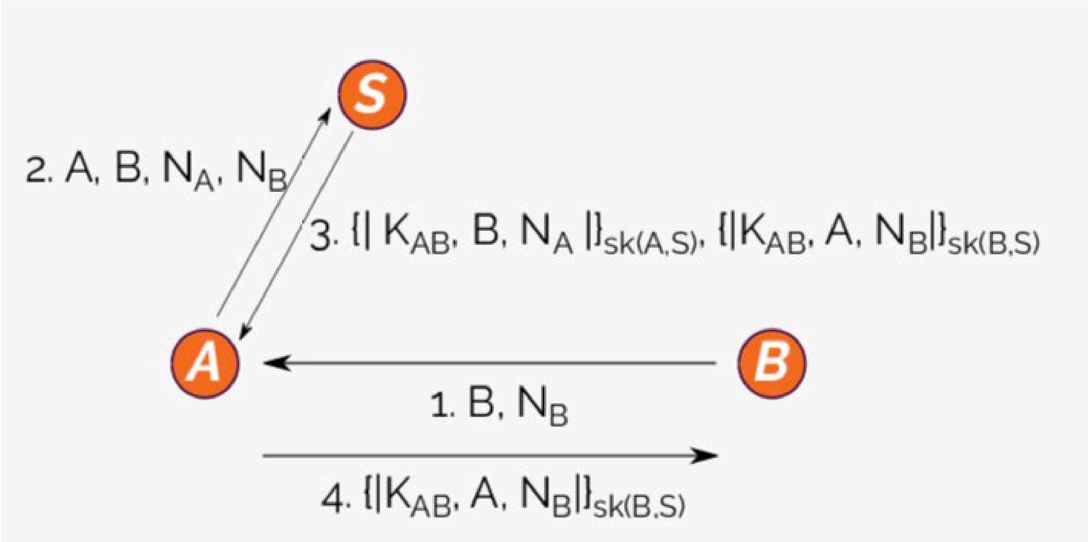
Building A Key Establishment Protocol

Summary

- 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

- No rush 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



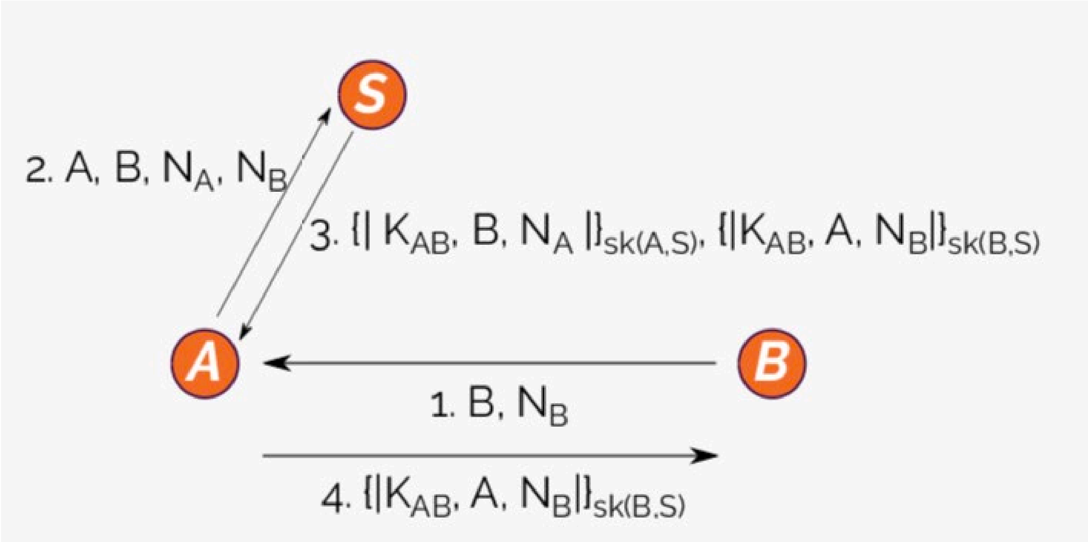
Building A Key Establishment Protocol
Summary

- 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

- No rush 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

➡ We need “means” to formalize protocols and goals



Notation Used In Protocol Modeling

Notation: Messages

Roles: *A*, *B* or *Alice*, *Bob*

Agents: *a*, *b*, *i*

Symmetric Keys: *K*, K_{AB}, …; $\mathrm{sk}(A, S)$

Symmetric Encryption: $\{|M|\}_K$

Public Keys: $K, \mathrm{pk}(A)$

Private Keys: $\mathrm{inv}(K),\ \mathrm{inv}(\mathrm{pk}(A))$

Asymmetric Encryption: $\{M\}_K$

Signing: $\{M\}_{\mathrm{inv}\ K}$

Nonces: $\mathrm{N}A$, $\mathrm{N}1$ fresh data items used for challenge/response

Timestamps: *T* denotes time, e.g., used for key expiration

Message concatenation: M_1, M_2, M_3

Notation Used In Protocol Modeling

Notation: Communication

- Fundamental event is communication between principals

$$A \longrightarrow B : \{A, T_1, K_{AB}\}_{\mathrm{pk}(B)}$$

- *A* and *B* name roles
  Can be instantiated by any principal playing in the role

- Communication is asynchronous

- Sender/receiver names "$A \longrightarrow B$" are not part of the message

- Protocol specifies actions of principals
  Alternatively, protocol defines a set of event sequences (traces)

Notation Used In Protocol Modeling

Notation: Protocols

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

  1. $A \longrightarrow B : \{\mathrm{N}A, A\}_{\mathrm{pk}(B)}$
  2. $\mathrm{B} \longrightarrow A : \{\mathrm{N}A, \mathrm{N}B\}_{\mathrm{pk}(A)}$
  3. $A \longrightarrow B : \{\mathrm{N}B\}_{\mathrm{pk}(B)}$

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

- What does a message $A \longrightarrow B : M$ 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

Protocol Execution

- Role A:

1. $A \longrightarrow B : \{\mathrm{N}A, A\}_{\mathrm{pk}(B)}$
2. $B \longrightarrow A : \{\mathrm{N}A, \mathrm{N}B\}_{\mathrm{pk}(A)}$
3. $A \longrightarrow B : \{\mathrm{N}B\}_{\mathrm{pk}(B)}$

1. Generate nonce N*A*, concatenate it with name, and encrypt with pk(B)
2. Receive a message *M*
   1. Decrypt *M* with inv(pk*(A))*, call it *M'*. If decryption fails, reject *M'*
   2. Split the message into two nonces N*A'* and N*B*. If that is not possible, reject *M*
   3. Check that N*A'* = N*A; if not*, reject *M*
3. Encrypt N*B* with pk*(B)* and send to B

Protocol Attacks

Different Kind of Attacks

Man-in-the-middle (or parallel sessions) attack: $A \leftrightarrow i \leftrightarrow B$

Replay (or freshness) attack: Reuse parts of previous messages

Masquerading attack: Pretend to be another principal

Reflection attack: Send transmitted information back to originator

Oracle attack: Take advantage of normal protocol responses as encryption and decryption "services"

Binding attack: Using messages in a different context/for a different purpose than originally intended

Type flaw attack: Substitute a different type of message field

Protocol Attacks

Different Kind of Attacks

Man-in-the-middle (or parallel sessions) attack: $A \leftrightarrow i \leftrightarrow B$

Replay (or freshness) attack: Reuse parts of previous messages

Masquerading attack: Pretend to be another principal

Reflection attack: Send transmitted information back to originator

Oracle attack: Take advantage of normal protocol responses as encryption and decryption “services”

Binding attack: Using messages in a different context/for a different purpose than originally intended

Type flaw attack: Substitute a different type of message field

* These attack types are not formally defined and there may be overlaps between these types

Man-in-the-Middle Attack

Diffie-Hellman (1/3)

- **Diffie-Hellman Key Exchange**
  - A and B agree on a DH group (g, p)
  - A generates large x and sends half-key $X = g^x \; mod \; p$ to B
  - B generates large y and sends half-key $Y = g^y \; mod \; p$ to A
  - A and B compute key $k = Y^x \; mod \; p = X^y mod \; p$

Security depends on the difficulty of computing the discrete logarithm of an exponentiated number modulo a large prime number

Man-in-the-Middle Attack

Diffie-Hellman (2/3)

- **Diffie-Hellman** (without authentication of the half-keys) can be attacked

1. $a \longrightarrow i\ (b): \exp(g, x)$
    1. $i\ (a) \longrightarrow b : \exp\ (g, z)$
    2. $b \longrightarrow i\ (a): \exp\ (g, y)$
2. $i(b) \longrightarrow i\ (a): \exp(g, z)$

a believes to share key $\exp(\exp(g, x), z)$ with b,
b believes to share key $\exp(\exp(g, y), z)$ with a,
i knows both keys…

Man-in-the-Middle Attack

Diffie-Hellman (3/3)

- **Prevention:** Authenticate the half keys, e.g., with digital signatures
  1. $A \longrightarrow B$: $\exp(g, x)_{\mathrm{inv}(\mathrm{pk}(A))}$
  2. $B \longrightarrow A$: $\exp(g, y)_{\mathrm{inv}\,(\mathrm{pk}(B))}$

Today many protocols are based on Diffie-Hellman, which is not a bad idea!

Type Flaw Attacks
Definition

- A message consists of a sequence of sub-messages, e.g., a principal's name, a nonce, a key, …

- Real messages are bit strings without type information, e.g.,

  1011 0110 0010 1110 0011 0111 1010 0000

- Type flaw is when $A \longrightarrow B : M$ and B accepts M as valid but parses it differently

  ⟹ B interprets the bits differently than A

Type Flaw Attacks

The Otway-Rees Protocol (1/2)

- Server-based protocol providing authenticated key distribution (with key authentication and key freshness) but without entity authentication or key confirmation

  1. $A \rightarrow B : M, A, B, \{|\mathrm{N}A, M, A, B|\}_{\mathrm{sk}(A,S)}$
  2. $B \rightarrow S : M, A, B, \{|\mathrm{N}A, M, A, B|\}_{\mathrm{sk}(A,S)}, \{|\mathrm{N}B, M, A, B|\}_{\mathrm{sk}(B,S)}$
  3. $S \rightarrow B : M, \{|\mathrm{N}A, K_{AB}|\}_{\mathrm{sk}(A,S)}, \{|\mathrm{N}B, K_{AB}|\}_{\mathrm{sk}(B,S)}$
  4. $B \rightarrow A : M, \{|\mathrm{N}A, K_{AB}|\}_{\mathrm{sk}(A,S)}$

- Server keys already known and M is a session id (e.g., an integer)

Type Flaw Attacks

The Otway-Rees Protocol (2/2)

- Suppose that $|M, A, B| = |K_{AB}|$

- Possible Attack (reflection/type flaw)

i replays parts of message 1 as message 4 (omitting steps 2 and 3)

1. $a \rightarrow i(b) : m, a, b, \{|na\ m, a, b|\}_{\mathrm{sk}(a,s)}$
2. $i(b) \rightarrow a : m, \{|na, \underbrace{m, a, b}_{\text{mistaken as } kab}|\}_{\mathrm{sk}(a,s)}$

Prudent Engineering of Security Protocols

- Principles proposed by Abadi and Needham (1994, 1995)
  - Every message should say what it means
  - Specify clearly conditions for a message to be acted on
  - Mention names explicitly if they are essential to the meaning
  - Be clear as to why encryption is being done: Confidentiality, message authentication, binding of messages, …
  - Be clear on what properties you are assuming
  - Beware of clock variations (for timestamps)
  - And other necessary things…

- Good advice, but
  - Is the protocol guaranteed to be secure then?
  - Is it optimal and/or minimal then?
  - Have you considered all types of attacks?
  - So on…

Summary

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

- Assumptions/Intruder model (following Dolev and Yao)
  - Can control the network
  - Can participate in the protocol
  - Can compose/decompose messages with the keys he has
  - Cannot break cryptography

  => Worst case assumption of an intruder

- 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, …)

Reading List

- Ross J. Anderson. Security Engineering: A Guide to Building Dependable Distributed Systems. John Wiley & Sons, Inc., New York, NY, USA, 1st edition, 2001.
  - The complete book is available at: http://www.cl.cam.ac.uk/~rja14/book.html
- Alfred J. Menezes, Scott A. Vanstone, and Paul C. Van Oorschot. Handbook of Applied Cryptography. CRC Press, Inc., Boca Raton, FL, USA, 5th edition, 2001.
  - The complete book is available at: http://cacr.uwaterloo.ca/hac/
- Bruce Schneier. Applied Cryptography. John Wiley & Sons, Inc., 2nd edition, 1996.

Thanks for your attention!

- Any questions or remarks?