Verifying Security Protocols in Tamarin

Ralf Sasse
Institute of Information Security
ETH Zurich

Tamarin Day 1, v.1
Jan 25, 2016

Organization of the course

- **No grade.**
  - ★ Participation required in lectures and exercises.
- **Web resources (slides, exercises, and protocols).**
  - ★ Download Link???
  - ★ Slides available now.
  - ★ https://github.com/tamarin-prover/tamarin-prover

Resources

- **Literature.**
  - ★ Franz Baader and Tobias Nipkow. *Term Rewriting and All That*. Cambridge University Press, 1998.
  - ★ Cas Cremers and Sjouke Mauw. *Operational Semantics and Verification of Security Protocols*. Springer, 2012.
    http://link.springer.com/book/10.1007/978-3-540-78636-8/page/1
  - ★ Benedikt Schmidt. *Formal analysis of key exchange protocols and physical protocols*. Ph.D. thesis, ETH Zürich, 2012.
    http://dx.doi.org/10.3929/ethz-a-009898924

Contents

module	content
1	Introduction to Security Protocols, Protocol Specification
2	Term Rewriting; Protocol Syntax and Semantics
3	Security Properties and Algorithmic Verification
4	Observational Equivalence
5	Applications: HISP, ARPKI

Formal Methods in Information Security

A historical perspective (with focus on security protocols):

1983	Symbolic attacker model [Dolev-Yao]
	Undecidability of secrecy problem for security protocols [Even-Goldreich]
1989	BAN logic for security protocol analysis [Burrows, Abadi, and Needham]
1995	MITM attack on Needham-Schroeder Public Key Protocol using the automatic verification tool Casper/FDR [Lowe]
1998	Inductive approach to verifying security protocols [Paulson]
2001	ProVerif: Efficient symbolic protocol verification tool [Blanchet]
	Constraint-based decision procedure for bounded sessions [Millen-Shmatikov]
	NP-Completeness of protocol insecurity for finite number of sessions [Rusinowitch-Turuani]
2005	Avispa protocol verification tools (OFMC, SATMC, ...) [Armando et al.]
2006	Scyther protocol verification tool [Cremers]
2012	Tamarin protocol verification tool [Meier, Schmidt]

Outline

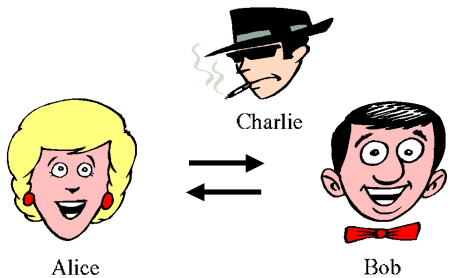
Outline

Motivation

- RSA, ECC, AES, . . . provide provably very good cryptographic primitives.
- How can we construct secure distributed applications with these primitives? E.g.:
  - ★ E-commerce
  - ★ E-banking
  - ★ E-voting
  - ★ Mobile communication
  - ★ Digital contract signing
- Even if cryptography is hard to break, this is not a trivial task.

Motivation

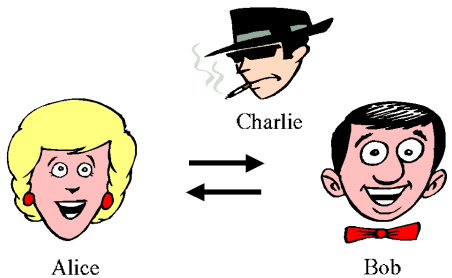
Example: Securing an e-banking application



$A \rightarrow B$: "Send \$10,000 to account X"
$B \rightarrow A$: "I'll transfer it now"

Motivation

Example: Securing an e-banking application

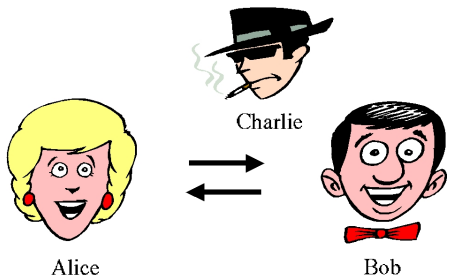


$A \rightarrow B$: "Send \$10,000 to account X"
$B \rightarrow A$: "I'll transfer it now"

- **How does B know the message originated from A?**
- **How does B know A just said it?**
- Confidentiality, integrity, accountability, non-repudiation, ...?

Motivation

Example: Securing an e-banking application



$A \rightarrow B$: "Send \$10,000 to account X"
$B \rightarrow A$: "I'll transfer it now"

- **How does B know the message originated from A?**
- **How does B know A just said it?**
- Confidentiality, integrity, accountability, non-repudiation, ...?

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

What is a 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** (or **cryptographic**) protocols use cryptographic mechanisms to achieve their **security goals**.
  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**.
- "Three-line programs that people still manage to get wrong"

Outline

Building a key establishment protocol

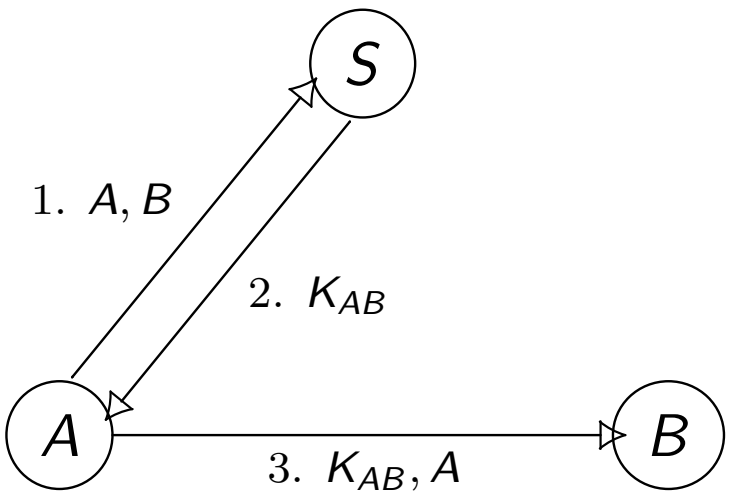
- An attempt to design a good protocol (from first principles).
- We choose one common scenario:
  - ★ A **set of users**, any 2 of whom may wish to establish a new **session key** for subsequent secure communications.
    Note: Users are not necessarily honest! (More later)
  - ★ There is an **honest server**.
    Note: Often called "trusted server", but trust $\neq$ honesty! We assume that an honest server never cheats and never gives out user secrets.
- We thus consider in this scenario a protocol with three **roles**: initiator role A, responder role B, and server role S.

Preliminaries

- In a concrete execution of a protocol, the roles are **played by agents** a.k.a. principals: *a*, *b*, *c* (charly), *s*, *i* (intruder), ...
- We use *i* as the name of an agent whose long-term keys are known to the adversary. No agent in our model knows that *i*'s long-term keys are known to the adversary.
- **Security goals of the protocol:**
  - ★ Key secrecy: At the end of the protocol, the session key K_{AB} is known to A and B, and possibly S, but to no other parties.
  - ★ Key freshness: A and B know that K_{AB} is freshly generated.
- Formalization questions (that we will consider later):
  - ★ How do we formalize the protocol steps and goals?
  - ★ How do we formalize "knowledge", "secrecy", "freshness"?

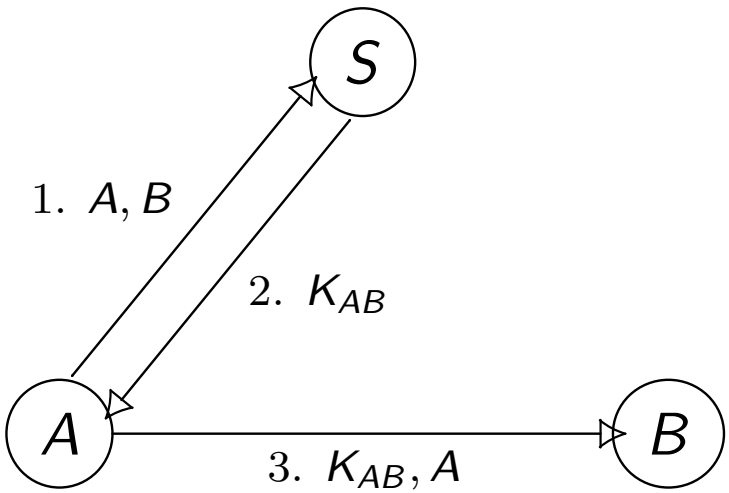
First attempt

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.

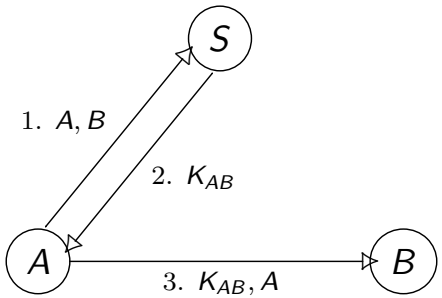
Alice&Bob notation



1. $A \rightarrow S : \quad A,\ B$
2. $S \rightarrow A : \quad K_{AB}$
3. $A \rightarrow B : \quad K_{AB},\ A$

Note: sender-receiver pairs "$A \rightarrow B$" in Alice&Bob notation are not part of the communicated message.

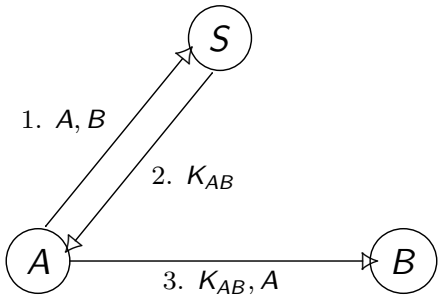
Protocol specification issues and conventions



1. $A \to S$: A, B
2. $S \to A$: K_{AB}
3. $A \to B$: K_{AB}, A

- K_{AB} does not contain any "information" about A and B. It is simply a name for the bit-string representing the session key.
- What if a message of the wrong format or no message at all is received? – Only messages passed in a successful protocol run are specified.
- No specification of internal actions of principals. (E.g., "create fresh K_{AB}", "store K_{AB} as a key for A and B".)
- Roles "know" what protocol run received messages are part of.

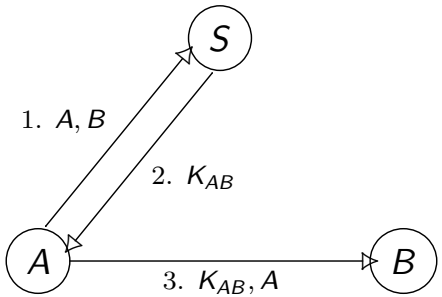
Security issues



1. $A \rightarrow S: \quad A, B$
2. $S \rightarrow A: \quad K_{AB}$
3. $A \rightarrow B: \quad K_{AB}, A$

- First problem with this protocol?

Security issues



1. $A \to S$: A, B
2. $S \to A$: K_{AB}
3. $A \to B$: K_{AB}, A

- First problem with this protocol? **Secrecy.**
  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:

Assumption (Threat Assumption 1)

The adversary is able to eavesdrop on all sent messages.

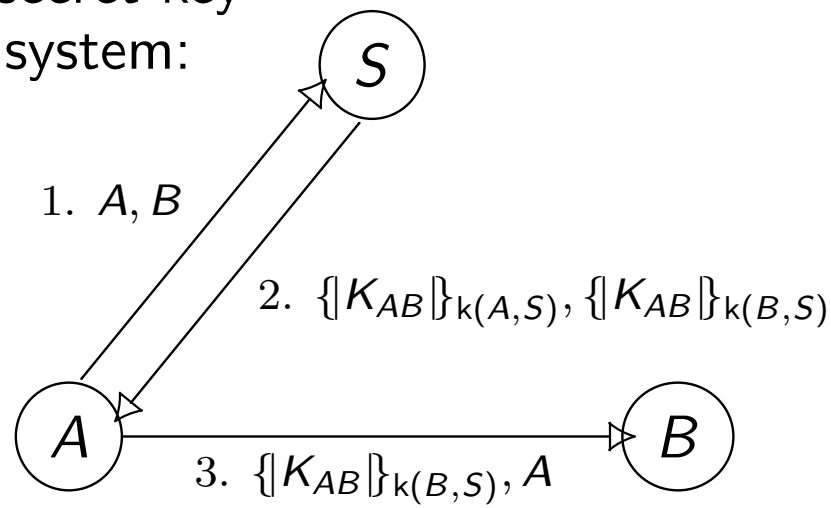
$\Rightarrow$ Use cryptography.

Second Attempt

Use cryptography

- Assume that S initially shares a secret key $\mathsf{k}(U, S)$ with each user U of the system:
  - ★ $\mathsf{k}(A, S)$ with A,
  - ★ $\mathsf{k}(B, S)$ with B,

  and encrypt message 2.

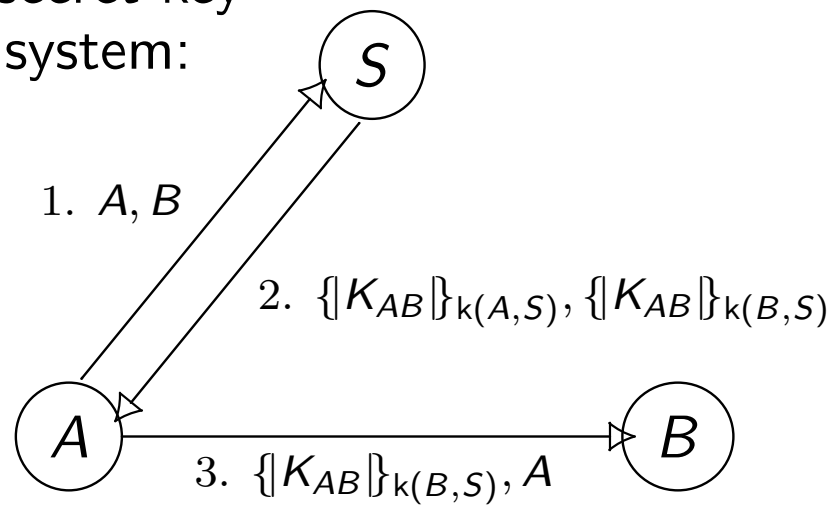


Second Attempt

Use cryptography

- Assume that S initially shares a secret key $\mathsf{k}(U, S)$ with each user U of the system:
  - ★ $\mathsf{k}(A, S)$ with A,
  - ★ $\mathsf{k}(B, S)$ with B,

  and encrypt message 2.
- Problems with protocol?
  - ★ Eavesdropping?

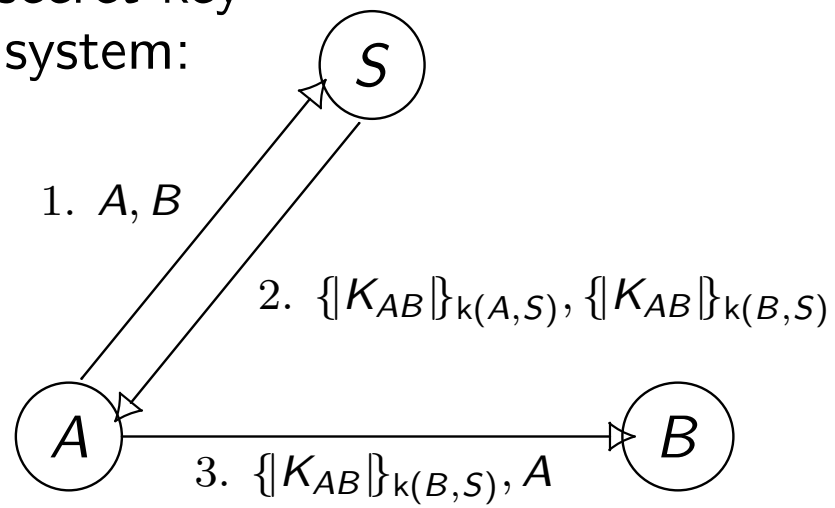


Second Attempt

Use cryptography

- Assume that S initially shares a secret key $\mathrm{k}(U, S)$ with each user U of the system:
  - ★ $\mathrm{k}(A, S)$ with A,
  - ★ $\mathrm{k}(B, S)$ with B,

  and encrypt message 2.
- Problems with protocol?
  - ★ Eavesdropping? – No.



Assumption (Perfect Cryptography)

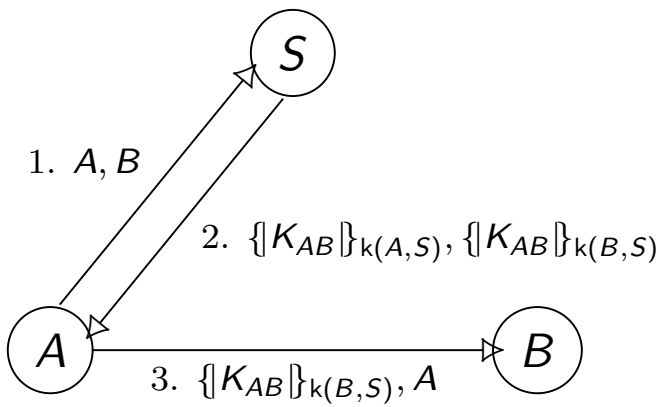
Encrypted messages may only be read by recipients who have the required decryption key.

Assuming that cryptography is perfect allows us to abstract away from the details of cryptographic algorithms.

Second Attempt

Security Issues

- Problem: information about who else has K_{AB} is unprotected, i.e. the principal's names are not bound to K_{AB}.



- Adversary may not only eavesdrop on sent messages, but also capture and modify them.

Second Attempt

Security Issues

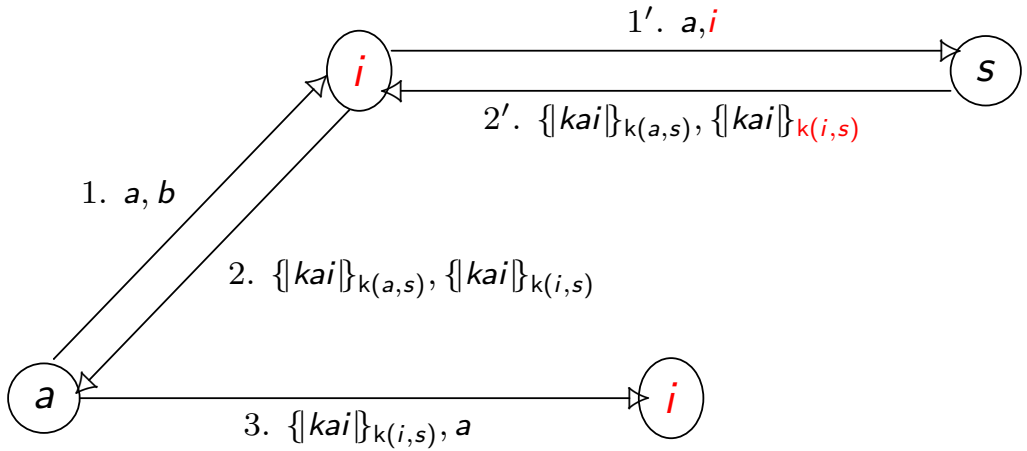
Assumption (Threat Assumption 2)

The adversary is able to intercept messages on the network and send to anybody (under any sender name) modified or new messages based on any information available.

- In other words: the adversary has complete control of the network channel(s) over which protocol messages flow.
  The adversary has complete control over the network.
- In contrast to ordinary communication protocols, we assume the worst-case network adversary.
  - ★ Although there may be no more than 4 or 5 messages involved in a legitimate session of the protocol, there are an infinite number of variations in which the adversary can participate.
  - ★ These variations involve an unbounded number of messages and each must satisfy the protocol's security requirements.

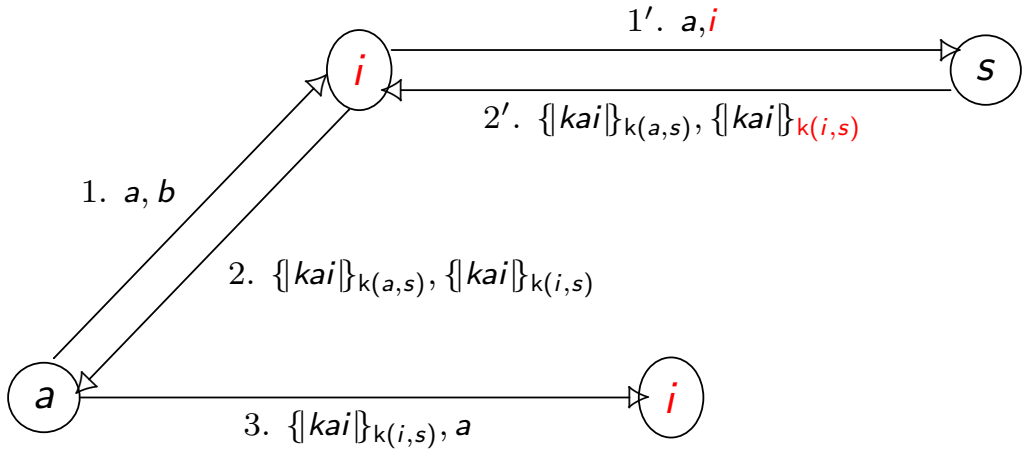
Binding Attack

There are several binding attacks on the second-attempt protocol, e.g.:



- Modify the message from a to s so that s generates a key kai for a and i and encrypts it with key $\mathsf{k}(i, s)$ known by the adversary.
- Since a cannot distinguish between encrypted messages meant for other agents she will not detect the modification.
- a will believe that the protocol has been successfully completed with b. However, the adversary knows kai and so can masquerade as b as well as learn all information that a sends intended for b.

Binding Attack



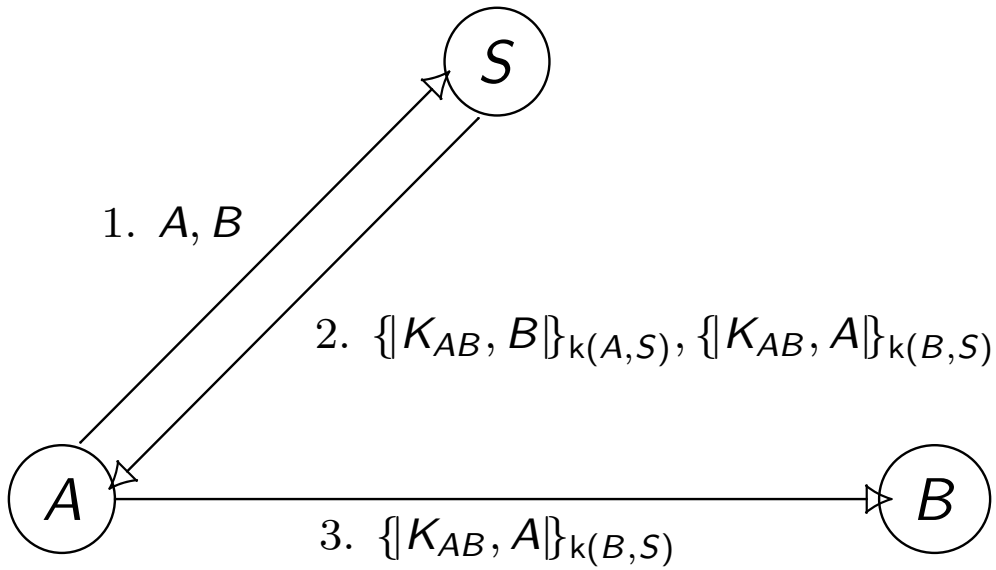
- Note: This attack will only succeed if i is a legitimate system user known to s, which is a realistic assumption:

Assumption (Threat Assumption 3)

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

Third Attempt

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



- Improvement: An adversary is unable to attack the protocol by eavesdropping or modifying the messages sent between honest parties (i.e., the previous two attacks fail).
- However, even now the protocol is not good enough to provide security in normal operating conditions.

Third Attempt

Security Issues

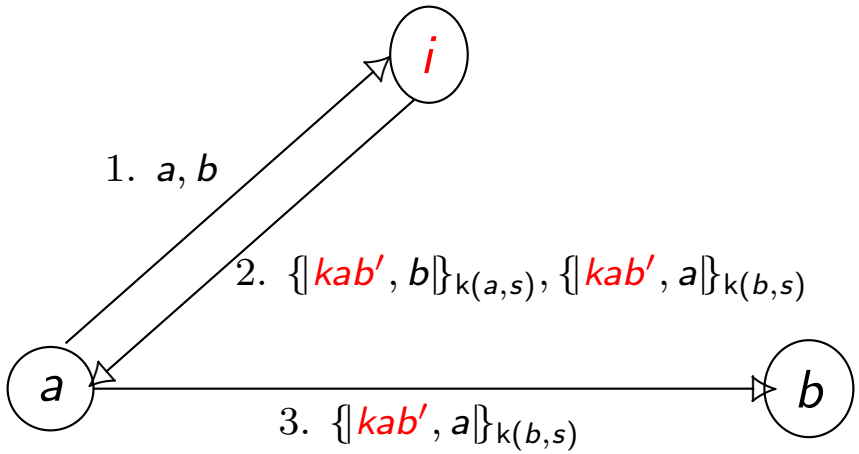
- A whole class of attacks becomes possible when old keys (or other security-relevant data) may be **replayed** in a later session. (Replay is possible by Threat Assumptions 1 & 2.)
- Additional problem: difference in quality between long-term keys and the session keys K_{AB} generated for each protocol session.

Assumption (Threat 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.

- Reasons for using session keys (as opposed to long-term keys):
  - ★ Key distribution problem: $\mathcal{O}(n^2)$ keys needed for n principals.
  - ★ Encrypted messages are 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.

Replay Attack and Session Key Compromisation



- *i* masquerades as *s* and replays *kab′*, an old key used by *a* and *b* in a previous session.
  - ★ By Threat Assumptions 1 & 2, the adversary can be expected to know and replay the encrypted messages in which *kab′* was transported to *a* and *b*.
- After the protocol run, the adversary can decrypt, modify, or insert information encrypted with *kab′* (no confidentiality, no integrity).
  - ★ By Threat Assumption 4, the adversary can be expected to know *kab′*.

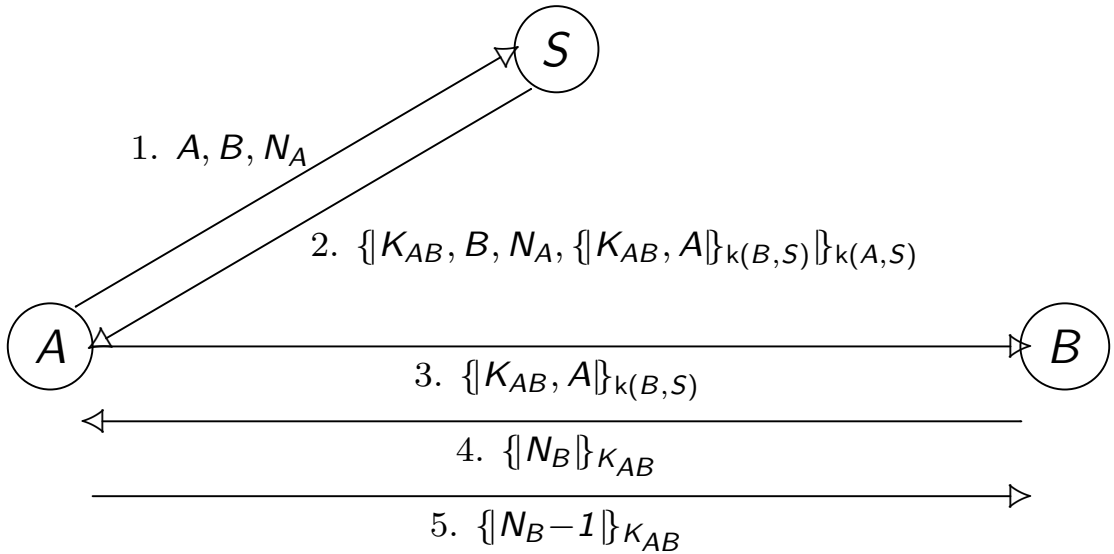
Thwarting the Replay Attack

- The replay attack can still be regarded as successful even if the adversary has not obtained the value of kab':
  - ★ Adversary succeeds 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.
- 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 guard against replay of session key, e.g., the **challenge-response** method:

Definition

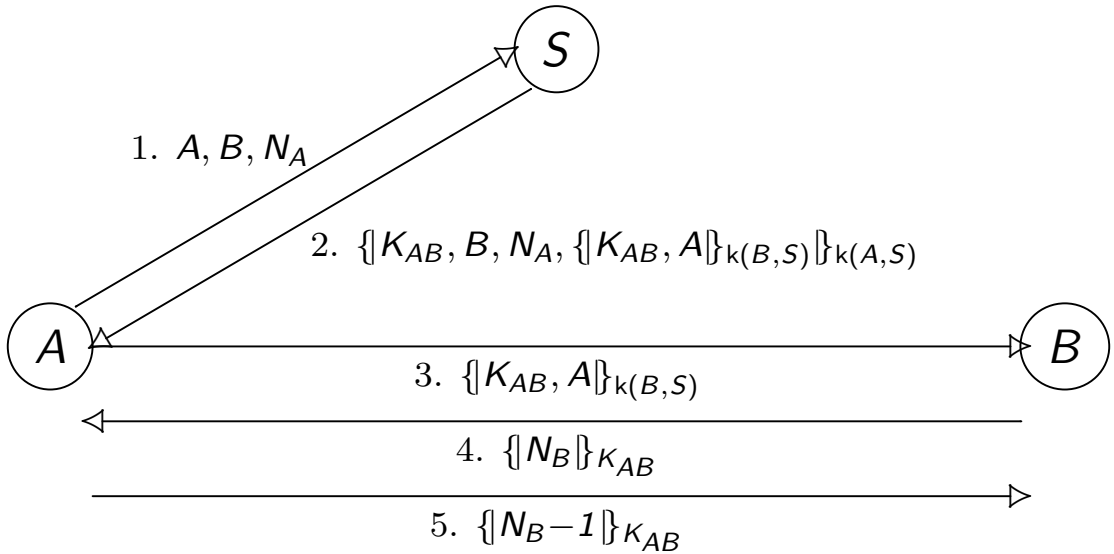
Def.: A **nonce** ("a number used only once") is a random value generated by one principal and returned to that principal to show that a message is newly generated.

Fourth Attempt



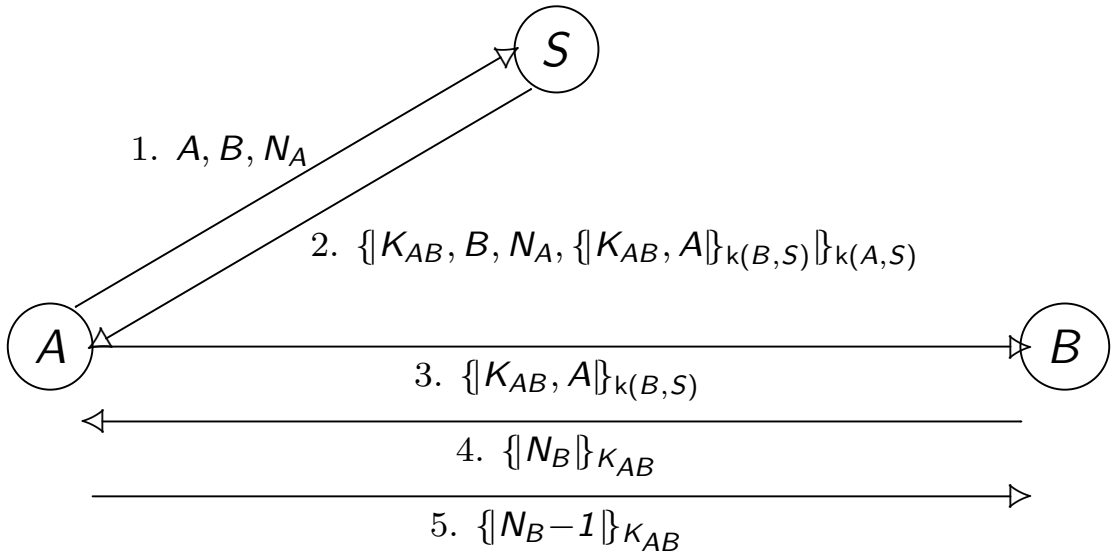
- A sends her nonce N_A to S with the request for a new key.
  - ★ Note: N_A is just a name for a number; nothing in N_A identifies who created it.
- If this same value is received with the session key, then A can deduce that the key has not been replayed.
  This reasoning is valid if session key and nonce are bound together cryptographically in such a way that only S can form such a message.

Fourth Attempt



- 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}.
  - ★ A uses K_{AB} to send a reply to B ("-1" to avoid replay of message 4).

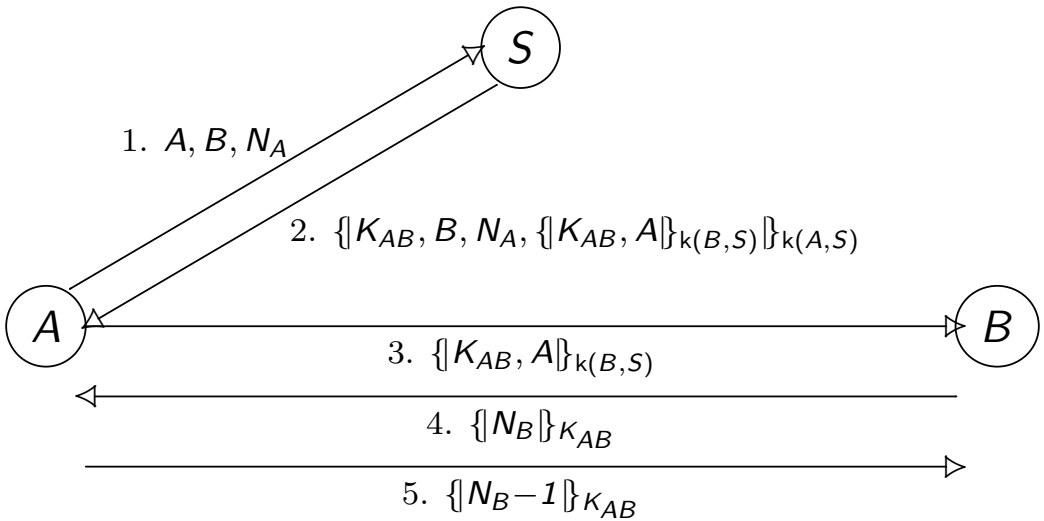
The NSCK Protocol



- This is actually a famous security protocol: Needham Schroeder with Conventional Keys [Clark-Jacob §6.3.1]
- Published by Needham and Schroeder in 1978, it has been the basis for a whole class of related protocols.

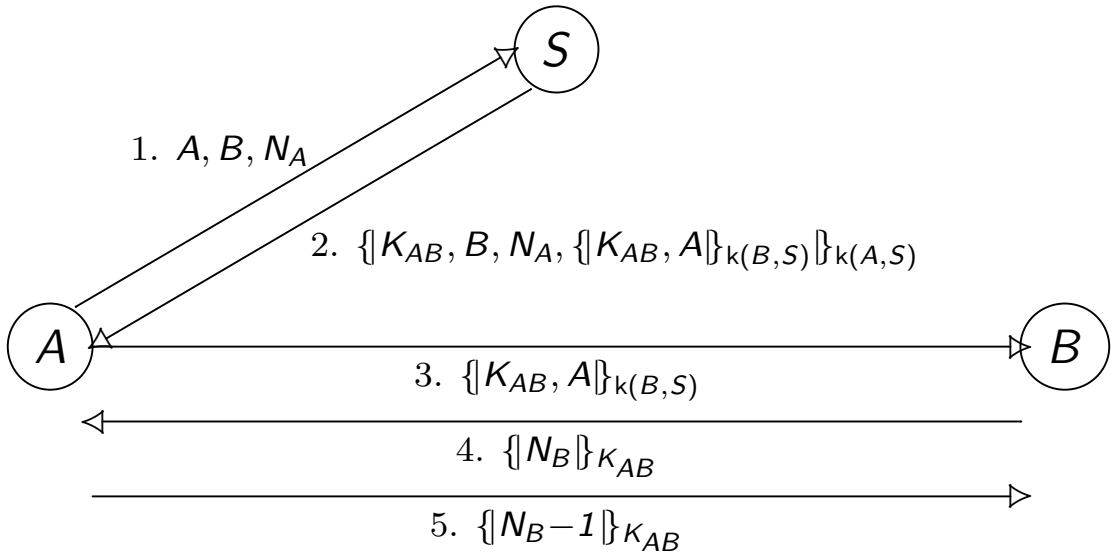
The NSCK Protocol

Security Issue

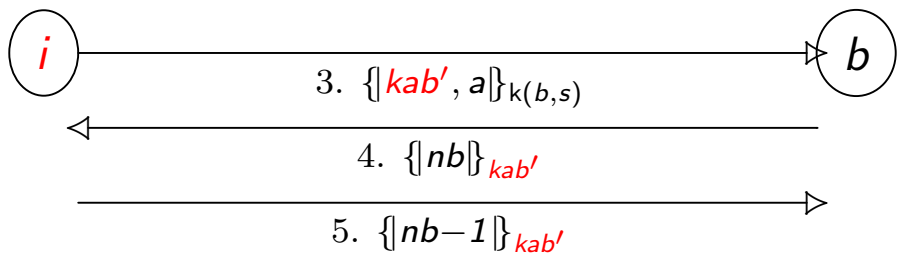


- Unfortunately, this protocol is vulnerable to a famous attack due to Denning and Sacco.
  - ★ Problem: assumption that only A can form correct reply to message 4 from B.
- Since the adversary can be expected to know the value of an old session key, this assumption is unrealistic.

Attack on NSCK

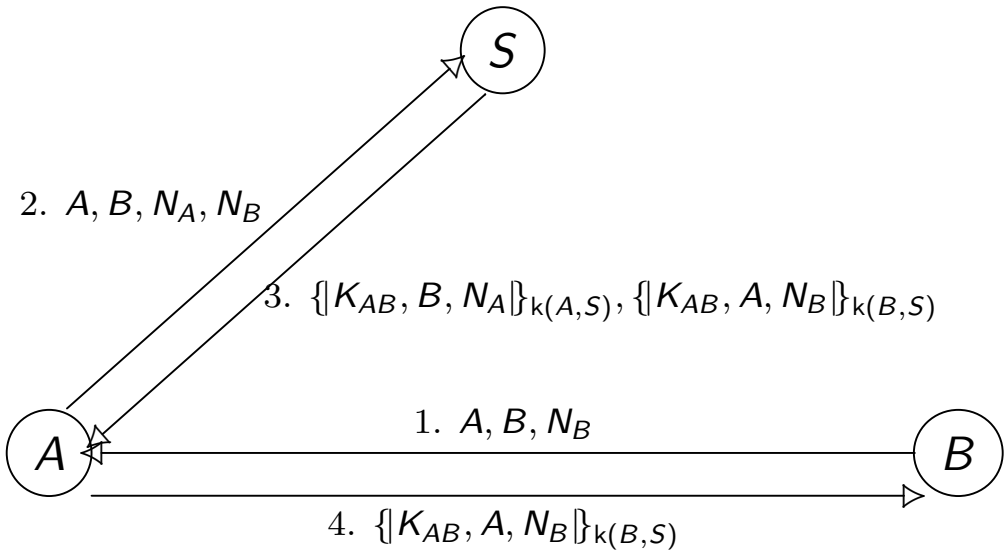


Adversary masquerades as a and convinces b to use old key: kab':



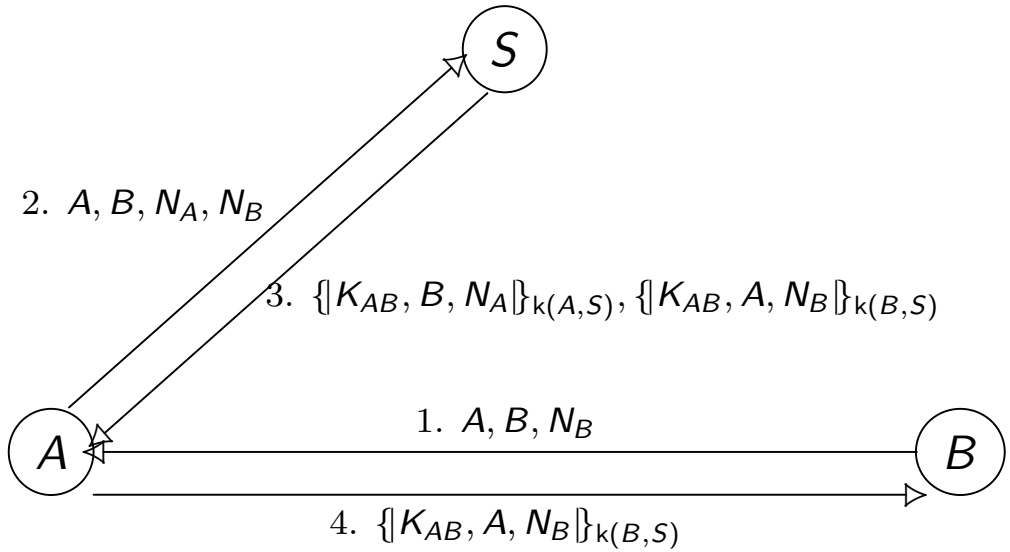
Fifth and Final Attempt

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



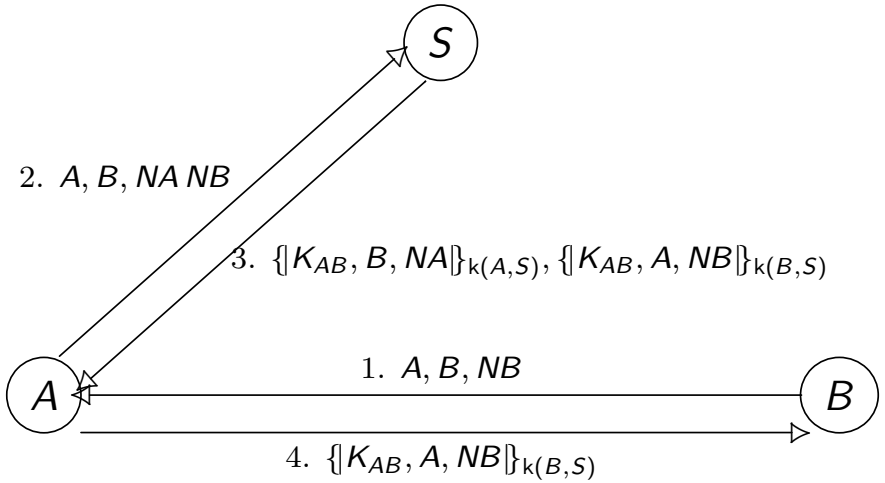
- The protocol is now initiated by B who sends his nonce N_B first to A.
- A adds her nonce N_A and sends both to S, who now sends K_{AB} in separate messages for A and B, which can be verified as fresh by the respective recipients.

Fifth and Final Attempt



- 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 $\{\lceil N_B \rceil\}_{K_{AB}}$ 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}. (Problem?)

Is it Secure?



- 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 a means to formalize protocols and goals.

Exercise

Can you improve on these protocols using Diffie-Hellman?
Why is that a much better idea?

Summary: adversary, attacks, and defenses

The **adversary** must be expected to

(TA 1) eavesdrop on messages, but cannot break cryptography,

(TA 2) completely control the network, i.e.,

- immediately intercept, modify, and fake messages,
- compose/decompose messages with the available keys,

(TA 3) participate in the protocol (as insider or outsider), and

(TA 4) be able to obtain old session keys.

TA 1-3: worst-case assumption of network adversary (after Dolev-Yao)

Attacks and defenses:

- Eavesdropping: encrypt session keys using long-term keys
- Binding attack: cryptographically bind names to session keys
- Replay attack: use challenge-response based on nonces

Outline

An authentication protocol

The Needham-Schroeder Public Key protocol (NSPK, 1978):

$$
\begin{array}{lll}
1. & A \to B: & \{NA, A\}_{\mathsf{pk}(B)} \\
2. & B \to A: & \{NA, NB\}_{\mathsf{pk}(A)} \\
3. & A \to B: & \{NB\}_{\mathsf{pk}(B)}
\end{array}
$$

Security goal: mutual authentication of A and B.

How the protocol is executed

1. $A \rightarrow B$: $\{NA, A\}_{\mathrm{pk}(B)}$
2. $B \rightarrow A$: $\{NA, NB\}_{\mathrm{pk}(A)}$
3. $A \rightarrow B$: $\{NB\}_{\mathrm{pk}(B)}$

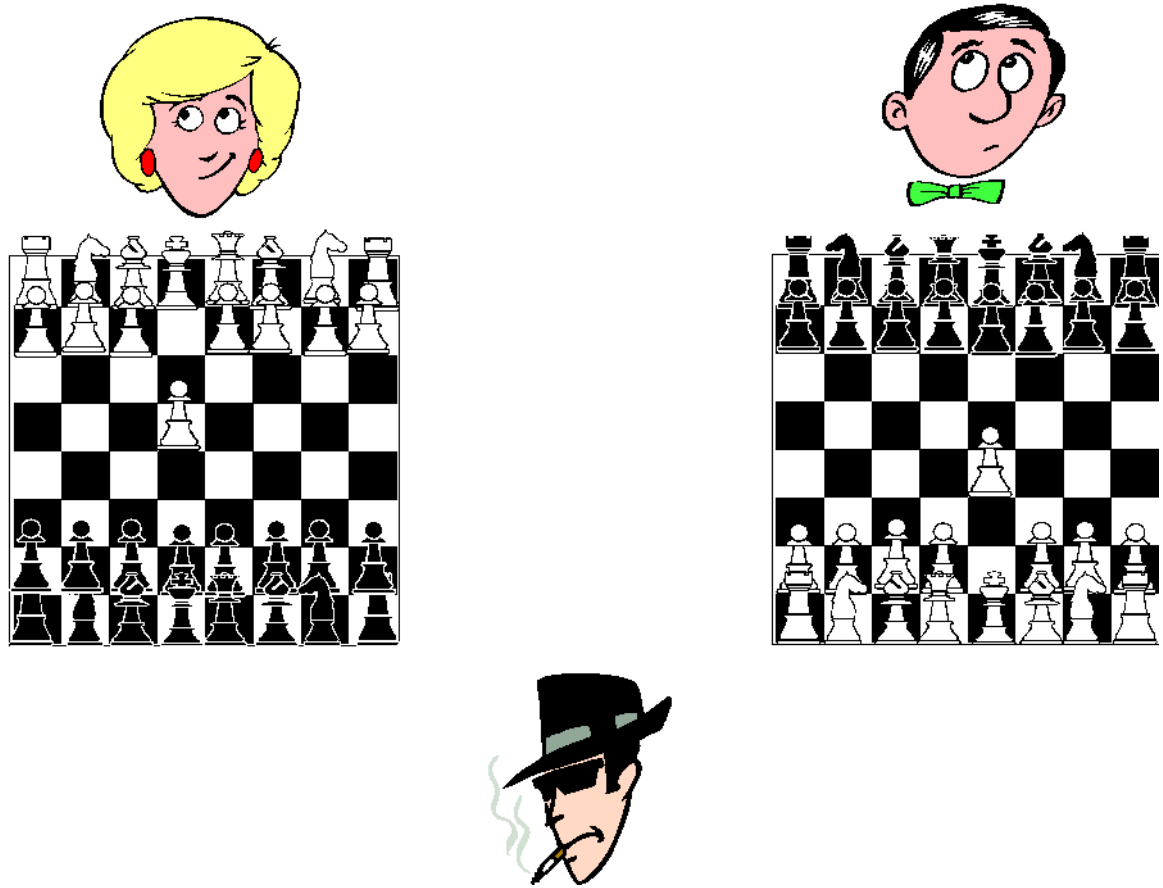
Role A:

1. Construct and send message 1.
   - ★ Generate nonce NA, concatenate it with name A, and encrypt with $\mathrm{pk}(B)$.
   - ★ Send $\{NA, A\}_{\mathrm{pk}(B)}$ to B.
2. Receive a message M and check that it is message 2.
   Q: how to do this when running multiple sessions (or protocols) in parallel?
   - ★ Decrypt M with $\mathrm{sk}(A)$, call it M'. If decryption fails, reject M.
     Q: how to detect wrong decryption?
     Q: what to do about rejected messages?
   - ★ Split the message into two nonces NA' and NB. If not possible, reject M.
     Q: how to check this?
   - ★ Check that $NA' = NA$; if not, reject M.
3. Construct and send message 3.
   - ★ Encrypt NB with $\mathrm{pk}(B)$
   - ★ Send $\{NB\}_{\mathrm{pk}(B)}$ to B.

Informal correctness

1. $A \rightarrow B : \{NA, A\}_{\mathsf{pk}(B)}$ "This is Alice and I have chosen a nonce NA."
2. $B \rightarrow A : \{NA, NB\}_{\mathsf{pk}(A)}$ "Here is your nonce NA. Since I could read it, I must be Bob. I also have a challenge NB for you."
3. $A \rightarrow B : \{NB\}_{\mathsf{pk}(B)}$ "You sent me NB. Since only Alice can read this and I sent it back, I must be Alice."

How not to lose against a grandmaster in chess



Man-in-the-middle attack

NSPK (1978):

$$\begin{array}{lll} 1. & A \to B: & \{NA, A\}_{\mathsf{pk}(B)} \\ 2. & B \to A: & \{NA, NB\}_{\mathsf{pk}(A)} \\ 3. & A \to B: & \{NB\}_{\mathsf{pk}(B)} \end{array}$$

Attack (Lowe 1996):

$$\begin{array}{lll} 1. & a \to i: & \{na, a\}_{\mathsf{pk}(i)} \\ & & \qquad\qquad 1.' \quad i(a) \to b: \quad \{na, a\}_{\mathsf{pk}(b)} \\ & & \qquad\qquad 2.' \quad b \to i(a): \quad \{na, nb\}_{\mathsf{pk}(a)} \\ 2. & i \to a: & \{na, nb\}_{\mathsf{pk}(a)} \\ 3. & a \to i: & \{nb\}_{\mathsf{pk}(i)} \\ & & \qquad\qquad 3.' \quad i(a) \to b: \quad \{nb\}_{\mathsf{pk}(b)} \end{array}$$

Property that "b authenticates a" is violated:
b believes to be talking to a, where he is in fact talking to i.

What went wrong?

- Problem in step 2.

$$B \to A : \{NA, NB\}_{\mathsf{pk}(A)}$$

  This message does not say where it comes from!
  (A cannot check her assumption on partner)
- Agent B should also give his name: $\{NA, NB, B\}_{\mathsf{pk}(A)}$.
  Known as Lowe's Fix.
- Is the improved version now correct?

Outline

Types of protocol attacks

- **Man-in-the-middle 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.

Note that these attack types are not formally defined and there may be overlaps between them.

Diffie-Hellman: man-in-the-middle attack

- Textbook Diffie-Hellman can be attacked:

$$
\begin{array}{llll}
1. & a \to i(b): & \exp(g,x) & \\
 & & & 1'. \quad i(a) \to b: \quad \exp(g,z) \\
 & & & 2'. \quad b \to i(a): \quad \exp(g,y) \\
2. & i(b) \to a: & \exp(g,z) &
\end{array}
$$

- a believes to share key $\exp(\exp(g,z),x)$ with b.
- b believes to share key $\exp(\exp(g,z),y)$ with a.
- The adversary knows both keys.

Diffie-Hellman: man-in-the-middle attack

- A "half" man-in-the-middle attack is possible, too:

$$\begin{array}{lll} 1. & a \to i(b): & \exp(g, x) \\ 2. & i(b) \to a: & \exp(g, z) \end{array}$$

- Countermeasure: authenticate the half-keys, e.g., with digital signatures:

$$\begin{array}{lll} 1. & A \to B: & \{\exp(g, X)\}_{\mathsf{sk}(A)} \\ 2. & B \to A: & \{\exp(g, Y)\}_{\mathsf{sk}(B)} \end{array}$$

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

Example of a reflection attack

This challenge-response authentication protocol

$$\begin{array}{lll} \text{M1.} & A \to B: & \{|NA|\}_{\mathsf{k}(A,B)} \\ \text{M2.} & B \to A: & \{|NA+1|\}_{\mathsf{k}(A,B)} \end{array}$$

admits a reflection attack (with 'oracle'):

Example of a reflection attack

This challenge-response authentication protocol

$$\begin{array}{lll} \text{M1.} & A \to B: & \{|NA|\}_{\mathsf{k}(A,B)} \\ \text{M2.} & B \to A: & \{|NA+1|\}_{\mathsf{k}(A,B)} \end{array}$$

admits a reflection attack (with 'oracle'):

$$\text{M1.1.} \quad a \to i(b): \{|na|\}_{\mathsf{k}(a,b)}$$

$$\text{M2.1.} \quad i(b) \to a: \{|na|\}_{\mathsf{k}(a,b)}$$

$$\text{M2.2.} \quad a \to i(b): \{|na+1|\}_{\mathsf{k}(a,b)}$$

$$\text{M1.2.} \quad i(b) \to a: \{|na+1|\}_{\mathsf{k}(a,b)}$$

a works on behalf of the adversary: a acts as an 'oracle', since she provides the correct answer to her own question.
a believes (at least) that b is operational, while b may no longer exist.
Fix: add A's name to message M1 or use different keys for each direction (i.e., $\mathsf{k}(a,b) \neq \mathsf{k}(b,a)$).

Example of a reflection attack

Reflection attack (with 'oracle'):

M1.1. $a \to i(b) : \{|na|\}_{k(a,b)}$

M1.2. $i(b) \to a : \{|na+1|\}_{k(a,b)}$

M2.1. $i(b) \to a : \{|na|\}_{k(a,b)}$

M2.2. $a \to i(b) : \{|na+1|\}_{k(a,b)}$

This attack requires that a executes several protocol runs in parallel.

Assumption (Threat Assumption 5)

The adversary may start any number of parallel protocol runs between any principals including different runs involving the same principals and with principals taking the same or different protocol roles.

Hence, our formal protocol model will allow for an unbounded number of protocol runs of arbitrary roles and with arbitrary participating principals.

Type flaw attacks

- A message consists of a sequence of submessages.
  Examples: a principal's name, a nonce, a key, ...
- Real messages are bit strings without type information.

  1011 0110 0010 1110 0011 0111 1010 0000

- **Type flaw** is when $A \rightarrow B : M$ and B accepts M as valid but parses it differently. I.e., B interprets the bits differently than A.
- Let's consider a couple examples.


"HONEY... HAND ME THE HAIR DRYER..."

The Otway-Rees protocol

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

M1. $A \to B : \quad M, A, B, \{\!|NA, M, A, B|\!\}_{\mathrm{k}(A,S)}$
M2. $B \to S : \quad M, A, B, \{\!|NA, M, A, B|\!\}_{\mathrm{k}(A,S)}, \{\!|NB, M, A, B|\!\}_{\mathrm{k}(B,S)}$
M3. $S \to B : \quad M, \{\!|NA, K_{AB}|\!\}_{\mathrm{k}(A,S)}, \{\!|NB, K_{AB}|\!\}_{\mathrm{k}(B,S)}$
M4. $B \to A : \quad M, \{\!|NA, K_{AB}|\!\}_{\mathrm{k}(A,S)}$

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

Why should(n't) it have the above properties?

Type flaw attack on the Otway-Rees protocol

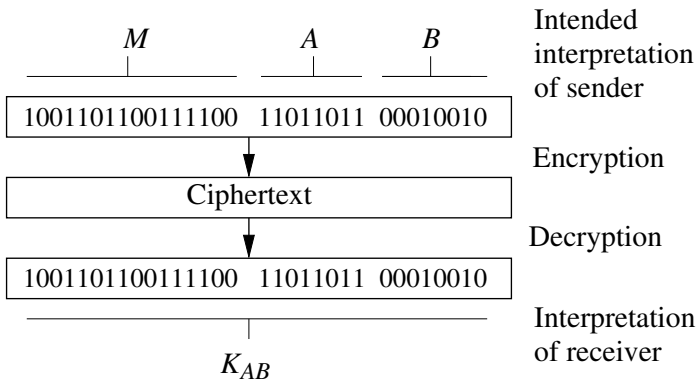
M1. $A \rightarrow B : \quad M, A, B, \{|NA, M, A, B|\}_{k(A,S)}$

M2. $B \rightarrow S : \quad M, A, B, \{|NA, M, A, B|\}_{k(A,S)}, \{|NB, M, A, B|\}_{k(B,S)}$

M3. $S \rightarrow B : \quad M, \{|NA, K_{AB}|\}_{k(A,S)}, \{|NB, K_{AB}|\}_{k(B,S)}$

M4. $B \rightarrow A : \quad M, \{|NA, K_{AB}|\}_{k(A,S)}$

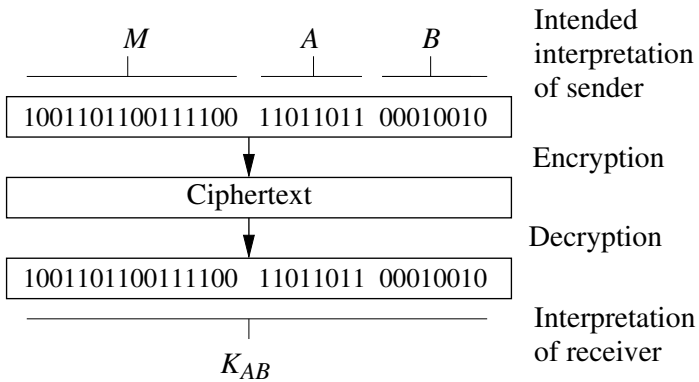
- Suppose $|M, A, B| = |K_{AB}|$, e.g., M is 32 bits, A and B are 16 bits, and K_{AB} is 64 bits.



Type flaw attack on the Otway-Rees protocol

M1. $A \to B : \quad M, A, B, \{|NA, M, A, B|\}_{k(A,S)}$
M2. $B \to S : \quad M, A, B, \{|NA, M, A, B|\}_{k(A,S)}, \{|NB, M, A, B|\}_{k(B,S)}$
M3. $S \to B : \quad M, \{|NA, K_{AB}|\}_{k(A,S)}, \{|NB, K_{AB}|\}_{k(B,S)}$
M4. $B \to A : \quad M, \{|NA, K_{AB}|\}_{k(A,S)}$

- Suppose $|M, A, B| = |K_{AB}|$, e.g., M is 32 bits, A and B are 16 bits, and K_{AB} is 64 bits.



- Attack 1 (Reflection/type-flaw): i replays parts of message 1 as message 4 (omitting steps 2 and 3).

M1. $a \to i(b) : \quad m, a, b, \{|na, m, a, b|\}_{k(a,s)}$
M4. $i(b) \to a : \quad m, \{|na, \underbrace{m, a, b}_{\text{mistaken as } kab}|\}_{k(a,s)}$

Type flaw attack on the Otway-Rees protocol (cont.)

M1. $A \to B : \quad M, A, B, \{|NA, M, A, B|\}_{\mathsf{k}(A,S)}$
M2. $B \to S : \quad M, A, B, \{|NA, M, A, B|\}_{\mathsf{k}(A,S)}, \{|NB, M, A, B|\}_{\mathsf{k}(B,S)}$
M3. $S \to B : \quad M, \{|NA, K_{AB}|\}_{\mathsf{k}(A,S)}, \{|NB, K_{AB}|\}_{\mathsf{k}(B,S)}$
M4. $B \to A : \quad M, \{|NA, K_{AB}|\}_{\mathsf{k}(A,S)}$

Attack 2: The adversary can play the role of S in M2 and M3 by reflecting the encrypted components of M2 back to B. Namely:

M1. $a \to b : \quad m, a, b, \{|na, m, a, b|\}_{\mathsf{k}(a,s)}$
M2. $b \to i(s) : \quad m, a, b, \{|na, m, a, b|\}_{\mathsf{k}(a,s)}, \{|nb, m, a, b|\}_{\mathsf{k}(b,s)}$
M3. $i(s) \to b : \quad m, \{|na, m, a, b|\}_{\mathsf{k}(a,s)}, \{|nb, m, a, b|\}_{\mathsf{k}(b,s)}$
M4. $b \to a : \quad m, \{|na, m, a, b|\}_{\mathsf{k}(a,s)}$

Type flaw attack on the Otway-Rees protocol (cont.)

M1. $A \to B : \quad M, A, B, \{|NA, M, A, B|\}_{k(A,S)}$
M2. $B \to S : \quad M, A, B, \{|NA, M, A, B|\}_{k(A,S)}, \{|NB, M, A, B|\}_{k(B,S)}$
M3. $S \to B : \quad M, \{|NA, K_{AB}|\}_{k(A,S)}, \{|NB, K_{AB}|\}_{k(B,S)}$
M4. $B \to A : \quad M, \{|NA, K_{AB}|\}_{k(A,S)}$

Attack 2: The adversary can play the role of S in M2 and M3 by reflecting the encrypted components of M2 back to B. Namely:

M1. $a \to b : \quad m, a, b, \{|na, m, a, b|\}_{k(a,s)}$
M2. $b \to i(s) : \quad m, a, b, \{|na, m, a, b|\}_{k(a,s)}, \{|nb, m, a, b|\}_{k(b,s)}$
M3. $i(s) \to b : \quad m, \{|na, m, a, b|\}_{k(a,s)}, \{|nb, m, a, b|\}_{k(b,s)}$
M4. $b \to a : \quad m, \{|na, m, a, b|\}_{k(a,s)}$

$\Rightarrow$ a and b accept wrong key and i can decrypt their subsequent communication! So key authentication (and secrecy) fails!

Prudent engineering of security protocols

- Principles proposed by Abadi and Needham (1994, 1995):
  - ★ Every message should say what it means.
  - ★ Specify clear 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, ...
    e.g., $\{X, Y\}_{\mathsf{sk}(K)}$ versus $\{X\}_{\mathsf{sk}(K)}, \{Y\}_{\mathsf{sk}(K)}$
  - ★ Be clear on what properties you are assuming.
  - ★ Beware of clock variations (for timestamps).
  - ★ ... and more ...

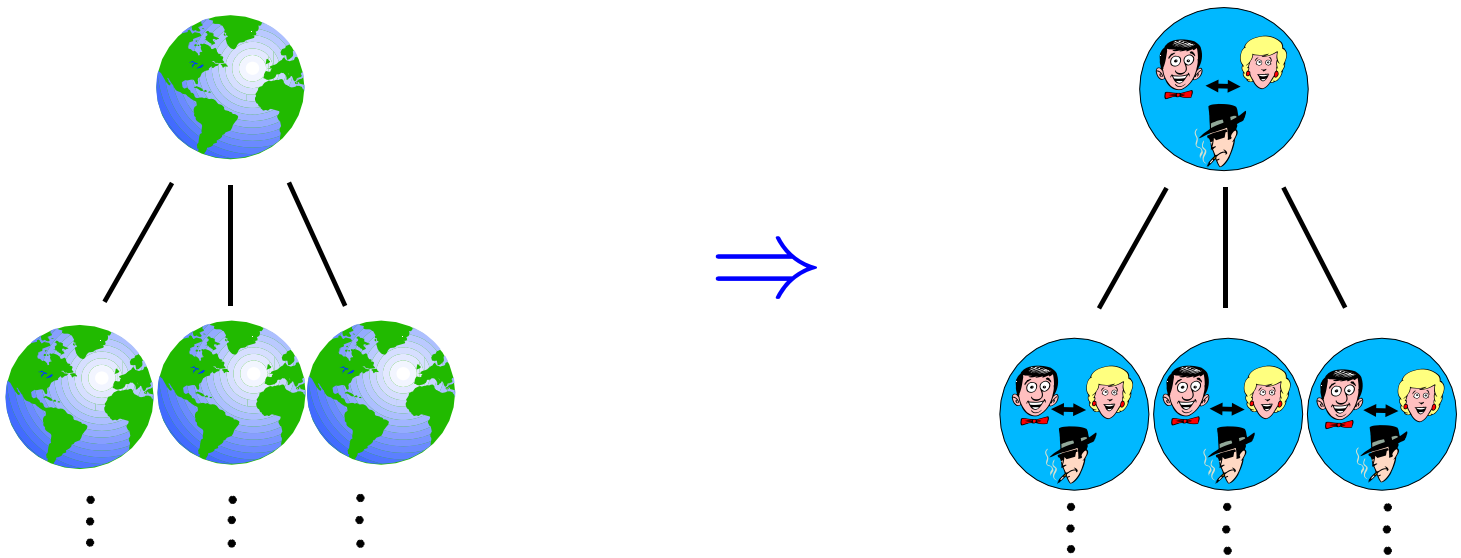
- Good advice, but
  - ★ Is the protocol guaranteed to be secure then?
  - ★ Is it optimal and/or minimal then?
  - ★ Have you considered all types of attacks?

Outline

Formal modeling and analysis of protocols

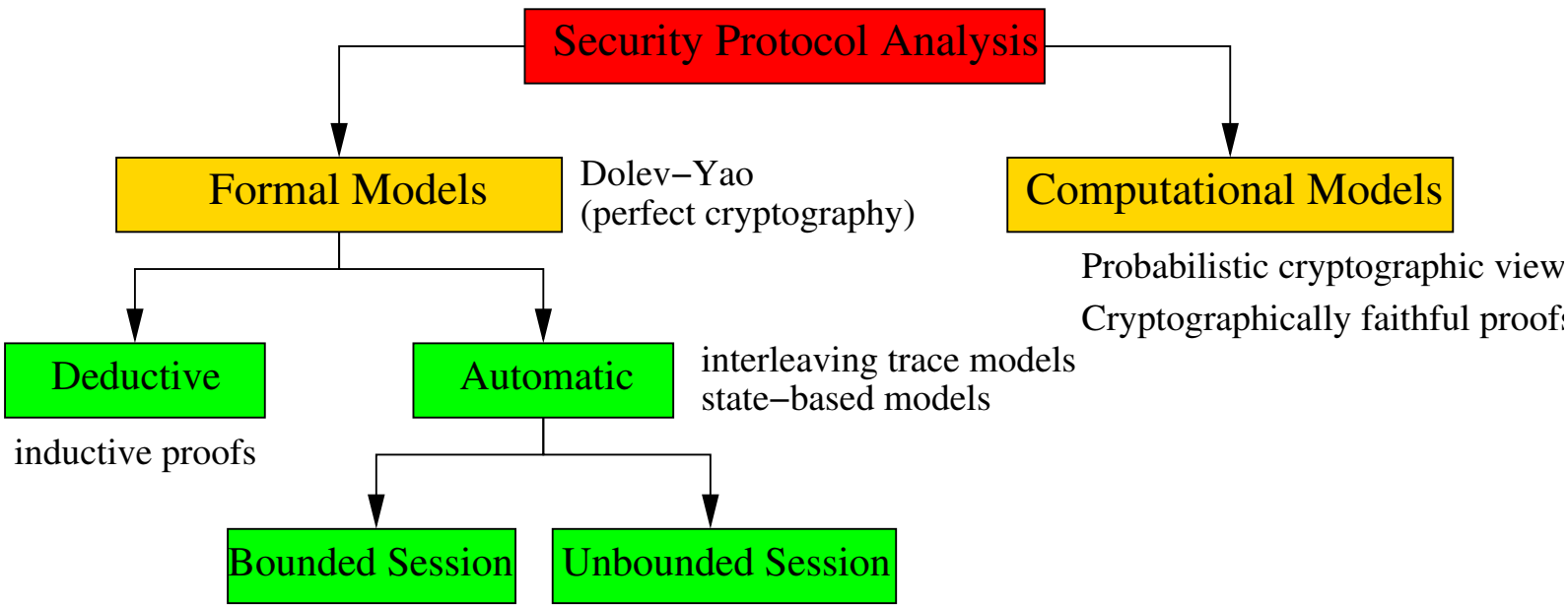
Goal: formally model protocols and their properties and provide a mathematically sound means for reasoning about these models.

Basis: suitable abstraction of protocols:

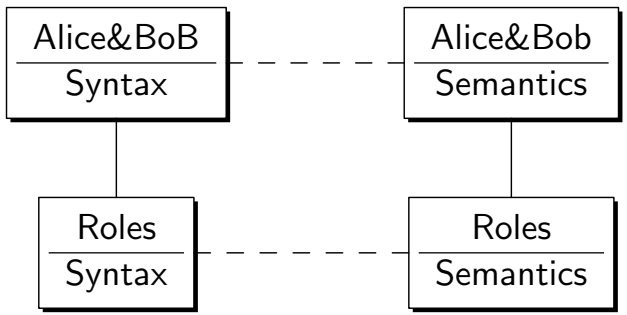


Analysis: with formal methods based on mathematics and logic.

Formal analysis of security protocols



Formal modeling of security protocols



Security protocol models

- Preliminaries: **Term rewriting**
- **Syntax**: Alice&Bob ($A \to B : M$) and role-based (send & receive rules)
- **Dolev-Yao adversary**: message derivation

Security properties

- **Semantics**: transition system, event traces
- **Security Goals**: Secrey, authentication

Syntax and semantics of security protocols

Syntax

- Fundamental event is communication between principals.
- Alice&Bob: protocol is a sequence of A&B events: $A \rightarrow B : M$.
- Role-based: protocol is a set of roles, each role is a sequence of send and receive rules.

Semantics

- Semantics of a role-based specification is a transition system.
- Threads are executing role instances; they keep a local store of received messages.
- Trace: records past events (send, receive, ...)
- Send rule: thread passes a message to adversary (i.e., network).
- Receive rule: thread obtains an adversary-derivable message.
- Alice&Bob protocol specifications: meaning defined in terms of translation to role-based specifications.

Why is security protocol analysis difficult?

Infinite state space for several reasons:

- Messages: adversary can produce messages of arbitrary size
- Sessions: unbounded number of parallel sessions
- Nonces: unbounded number of nonces (if sessions unbounded)

Undecidability

- secrecy problem for security protocols is undecidable (Even & Goldreich, 1983)
- even if the number of nonces or the message size is bounded

Approaches that work well in practice

- symbolic analysis methods: avoid state enumeration
- sophisticated search strategies to avoid non-termination
- abstraction techniques: over-approximate reachable states

Summary

- Security protocols can achieve properties that cryptographic primitives alone cannot offer, e.g., authentication, freshness, ...
- A protocol without explicit goals and assumptions is useless.
- Even three-liners show how difficult the art of correct design is.
- Protocol without a proof of its properties is probably wrong.
- Formal modeling and analysis of protocols is required.
  Formal analysis is non-trivial (even assuming perfect cryptography).

Outline

Real-world protocol standards: ISO/IEC 9798

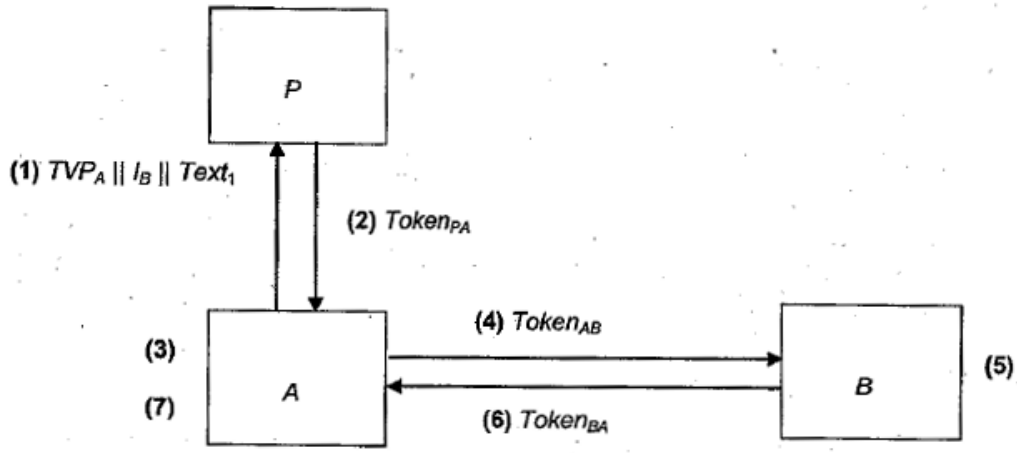


Figure 5 — Mechanism 5 — Four-pass authentication

The form of the token ($Token_{PA}$), sent by P to A, is:

$$Token_{PA} = Text_4 \| e_{K_{AP}}(TVP_A \| K_{AB} \| I_B \| Text_3) \| e_{K_{BP}}(TN_P \| K_{AB} \| I_A \| Text_2)$$

The form of the token ($Token_{AB}$), sent by A to B, is:

$$Token_{AB} = Text_6 \| e_{K_{BP}}(TN_P \| K_{AB} \| I_A \| Text_2) \| e_{K_{AB}}(TN_A \| I_B \| Text_5)$$

The form of the token ($Token_{BA}$), sent by B to A, is:

$$Token_{BA} = Text_8 \| e_{K_{AB}}(TN_B \| I_A \| Text_7)$$

The choice of using either time stamps or sequence numbers in this mechanism depends on the capabilities of the entities involved as well as on the environment.

Real-world protocol specifications: IKE RFC

When using encryption for authentication, Main Mode is defined as follows.

```
     Initiator                          Responder
     -----------                        -----------
     HDR, SA                    -->
                                <--     HDR, SA
     HDR, KE, [ HASH(1), ]
       <IDii_b>PubKey_r,
         <Ni_b>PubKey_r         -->
                                                    HDR, KE, <IDir_b>PubKey_i,
                                <--                      <Nr_b>PubKey_i
     HDR*, HASH_I               -->
                                <--     HDR*, HASH_R
```

Aggressive Mode authenticated with encryption is described as follows:

```
     Initiator                          Responder
     -----------                        -----------
     HDR, SA, [ HASH(1),] KE,
       <IDii_b>Pubkey_r,
        <Ni_b>Pubkey_r          -->
                                                    HDR, SA, KE, <IDir_b>PubKey_i,
                                <--                      <Nr_b>PubKey_i, HASH_R
     HDR, HASH_I                -->
```

Real-world protocol specifications: IKE RFC


```
  For signatures:            SKEYID = prf(Ni_b | Nr_b, g^xy)
  For public key encryption: SKEYID = prf(hash(Ni_b | Nr_b), CKY-I |
CKY-R)
  For pre-shared keys:       SKEYID = prf(pre-shared-key, Ni_b |
Nr_b)
```

The result of either Main Mode or Aggressive Mode is three groups of authenticated keying material:

```
   SKEYID_d = prf(SKEYID, g^xy | CKY-I | CKY-R | 0)
   SKEYID_a = prf(SKEYID, SKEYID_d | g^xy | CKY-I | CKY-R | 1)
   SKEYID_e = prf(SKEYID, SKEYID_a | g^xy | CKY-I | CKY-R | 2)
```

and agreed upon policy to protect further communications. The values of 0, 1, and 2 above are represented by a single octet. The key used for encryption is derived from SKEYID_e in an algorithm-specific manner (see appendix B).

To authenticate either exchange the initiator of the protocol generates HASH_I and the responder generates HASH_R where:

```
 HASH_I = prf(SKEYID, g^xi | g^xr | CKY-I | CKY-R | SAi_b | IDii_b )
 HASH_R = prf(SKEYID, g^xr | g^xi | CKY-R | CKY-I | SAi_b | IDir_b )
```

For authentication with digital signatures, HASH_I and HASH_R are signed and verified; for authentication with either public key encryption or pre-shared keys, HASH_I and HASH_R directly authenticate the exchange. The entire ID payload (including ID type, port, and protocol but excluding the generic header) is hashed into both HASH_I and HASH_R.

What are formal models?

- A **language** is **formal** when it has a well-defined syntax and semantics. Additionally there is often a deductive system for determining the truth of statements.
- **Examples:**

What are formal models?

- A **language** is **formal** when it has a well-defined syntax and semantics. Additionally there is often a deductive system for determining the truth of statements.
- **Examples:** propositional logic, first-order logic.
- A **model** (or **construction**) is **formal** when it is specified in a formal language.
- Standard protocol notation is not formal.
- We will see how to formalize such notations.

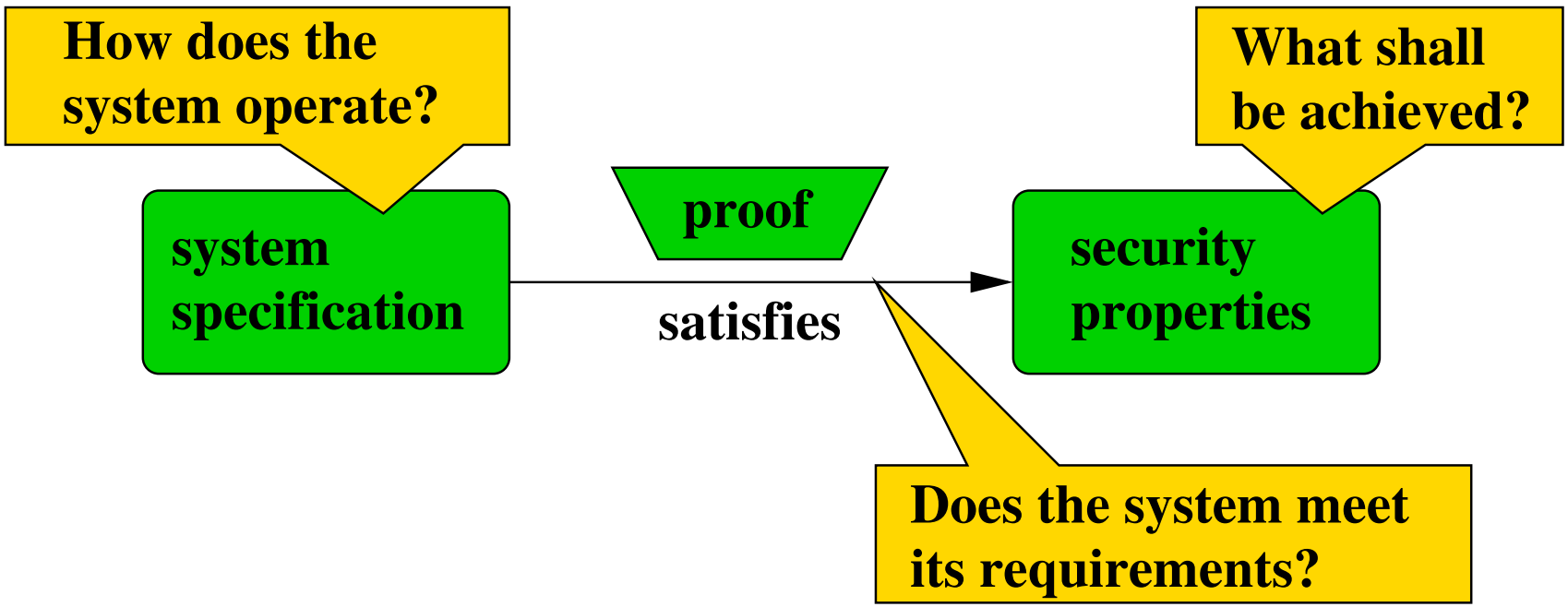
Formal modeling and analysis of protocols

Goal: formally model protocols and their properties and provide a mathematically sound means for reasoning about these models.

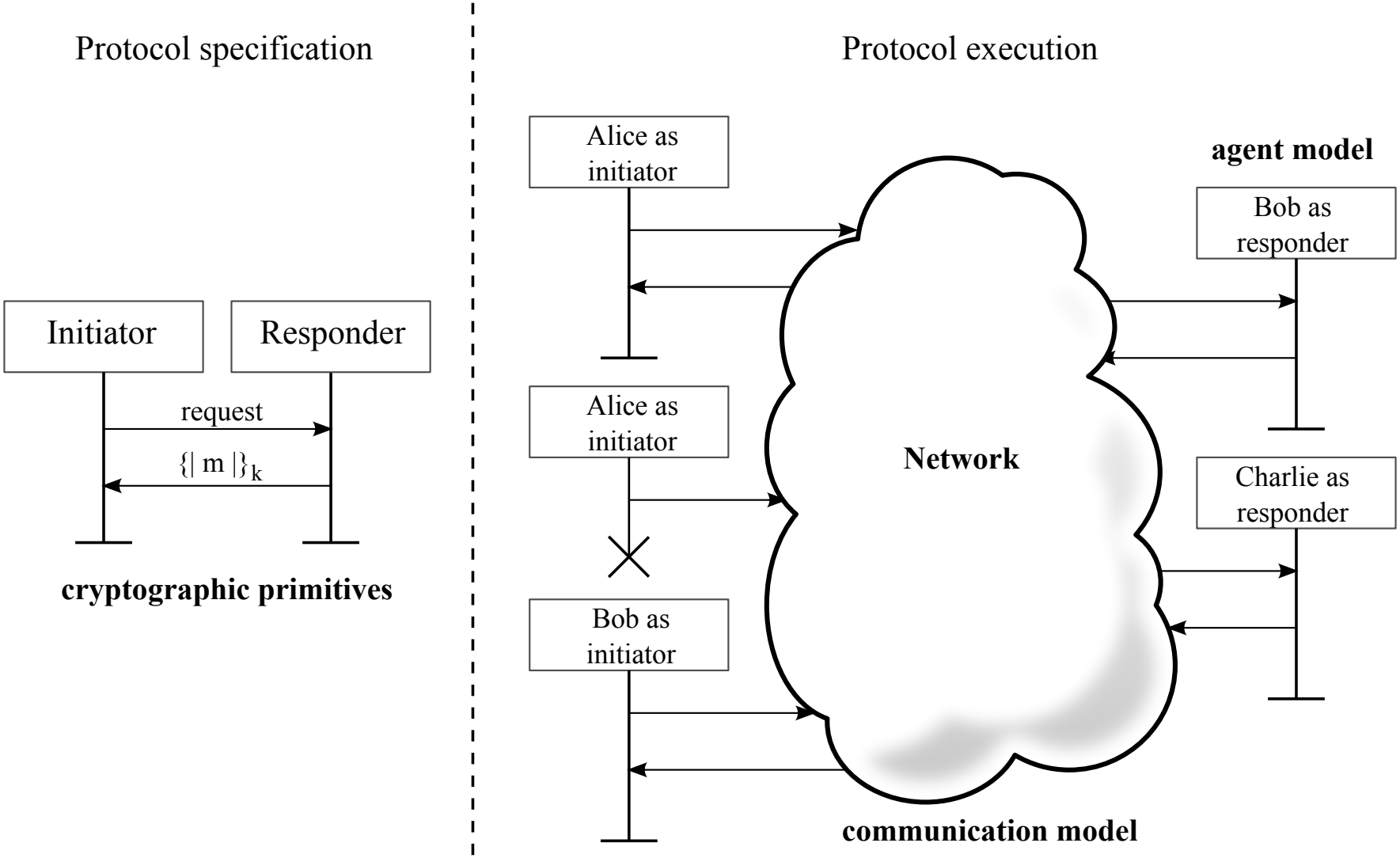
Basis: suitable abstraction of protocols.

Analysis: with formal methods based on mathematics and logic.

Formal Methods



From protocol sequence charts to protocol execution



Outline

A (Semi-)Formal Alice & Bob Language

Actions :

$$
\begin{array}{lcll}
A & \rightarrow & B & : \{NA, A\}_{\mathsf{pk}(B)} \\
B & \rightarrow & A & : \{NA, NB\}_{\mathsf{pk}(A)} \\
A & \rightarrow & B & : \{NB\}_{\mathsf{pk}(B)}
\end{array}
$$

A (Semi-)Formal Alice & Bob Language

Protocol : *NSPK*

Actions :

$$
\begin{array}{lcll}
A & \rightarrow & B & : \{NA, A\}_{\mathsf{pk}(B)} \\
B & \rightarrow & A & : \{NA, NB\}_{\mathsf{pk}(A)} \\
A & \rightarrow & B & : \{NB\}_{\mathsf{pk}(B)}
\end{array}
$$

First, let's give it a name.

A (Semi-)Formal Alice & Bob Language

Protocol : *NSPK*

Types :

Agent A, B;

Number NA, NB;

Function pk;

Actions :

$$A \rightarrow B : \{NA, A\}_{\mathsf{pk}(B)}$$
$$B \rightarrow A : \{NA, NB\}_{\mathsf{pk}(A)}$$
$$A \rightarrow B : \{NB\}_{\mathsf{pk}(B)}$$

Specify the types of all identifiers.

NB: we do not necessarily consider types in the analysis!

A (Semi-)Formal Alice & Bob Language

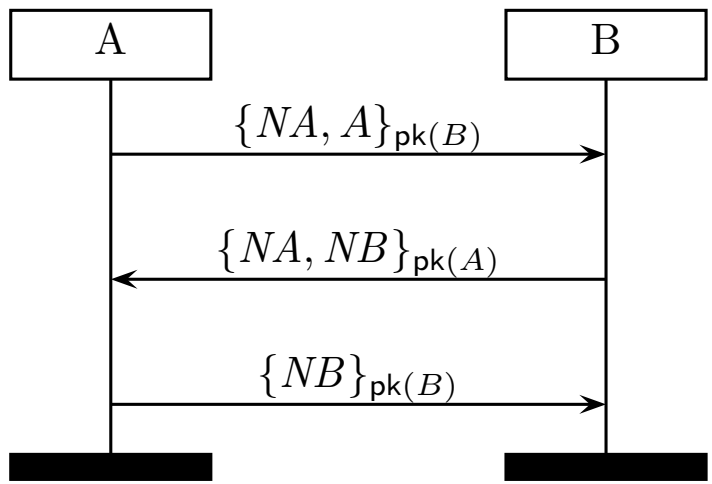
Protocol : *NSPK*
Types :
  Agent A, B;
  Number NA, NB;
  Function pk;
Knowledge :
  A : $A, B, \mathsf{pk}, \mathsf{sk}(A)$;
  B : $B, \mathsf{pk}, \mathsf{sk}(B)$;
Actions :

$$\begin{array}{lcll} A & \rightarrow & B & : \{NA, A\}_{\mathsf{pk}(B)} \\ B & \rightarrow & A & : \{NA, NB\}_{\mathsf{pk}(A)} \\ A & \rightarrow & B & : \{NB\}_{\mathsf{pk}(B)} \end{array}$$

Initial knowledge of each role uses only variables of type Agent.
Every other variable: freshly created by the agent who first uses it.

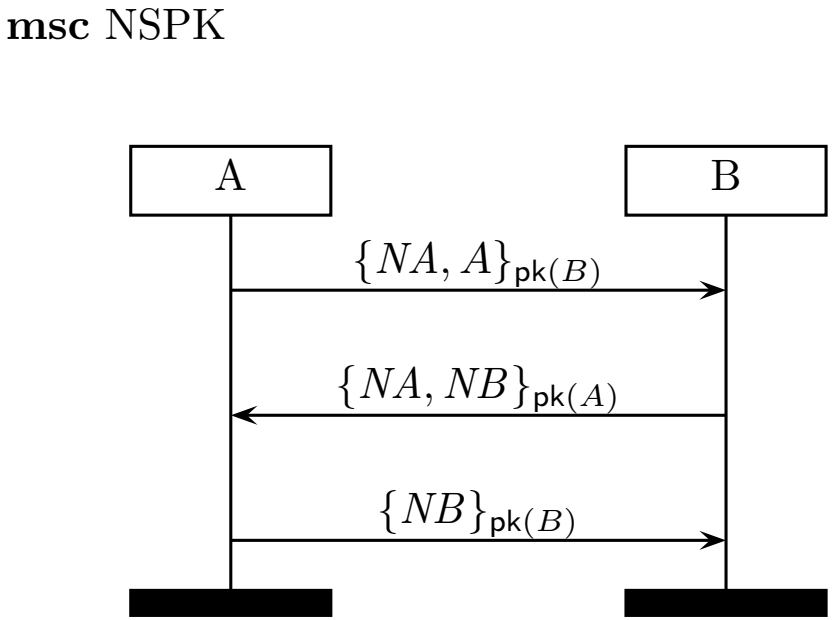
Message Sequence Charts

msc NSPK



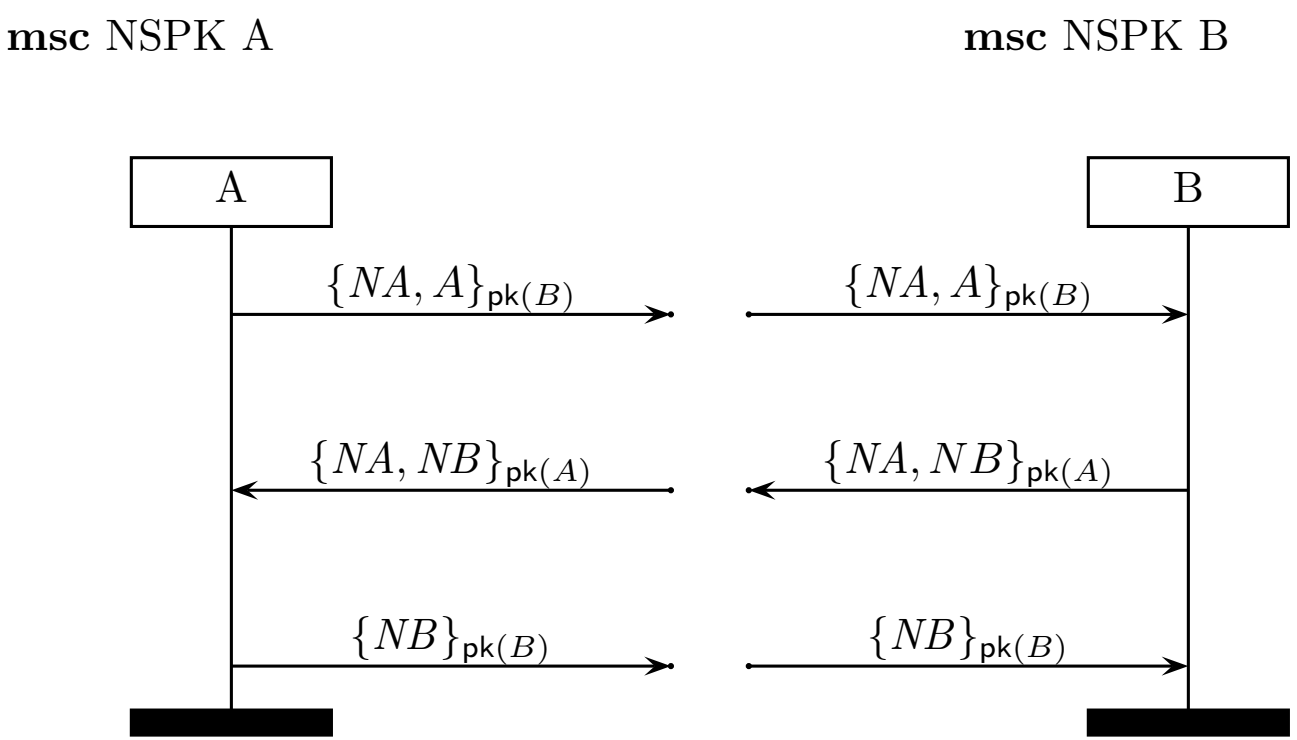
Towards the Meaning of an AnB Specification

Split a message sequence chart into single roles
(aka chords, symbolic strands, role scripts):



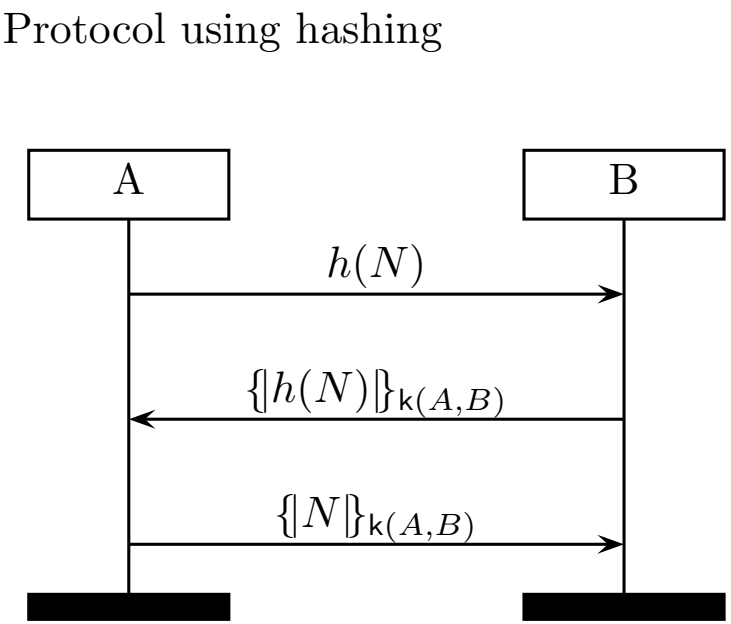
Towards the Meaning of an AnB Specification

Split a message sequence chart into single roles
(aka chords, symbolic strands, role scripts):



Towards the Meaning of an AnB Specification

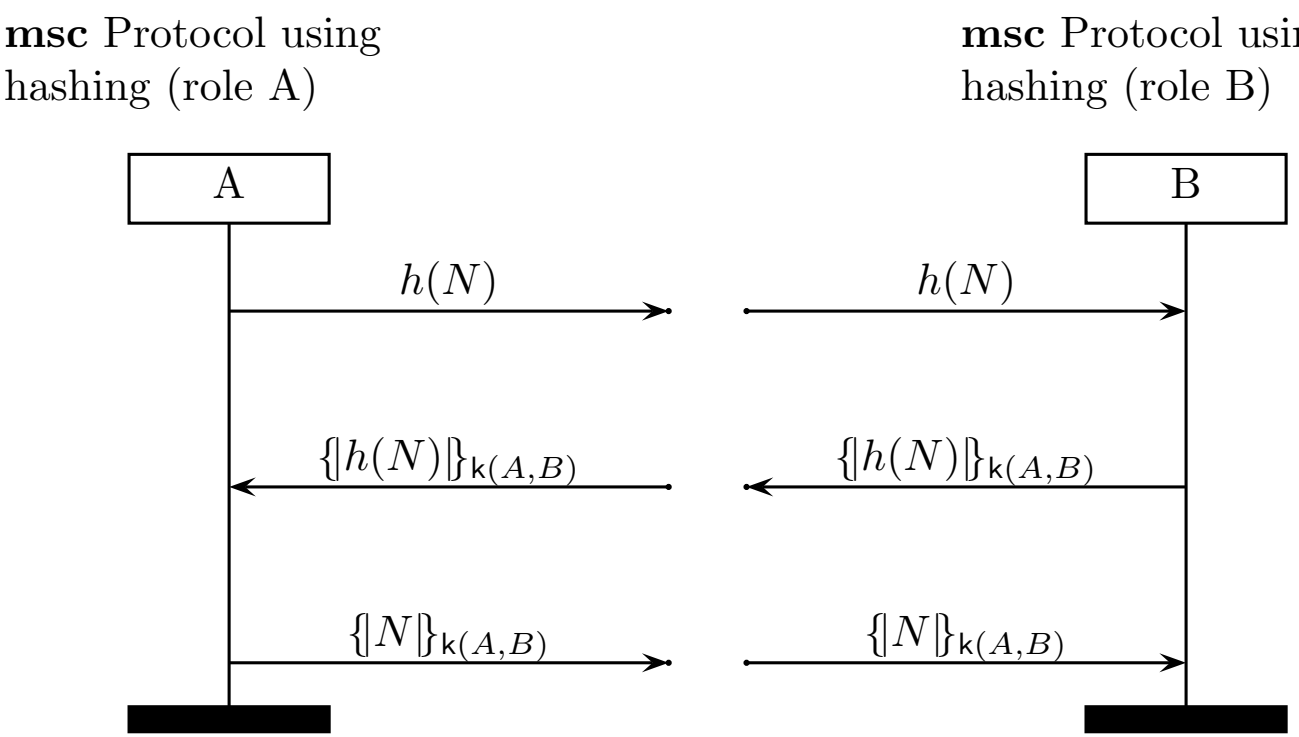
Not trivial for some protocols:



Here, $\mathsf{k}(A, B)$ is a shared key of A and B and N is fresh.

Towards the Meaning of an AnB Specification

Not trivial for some protocols:



This is wrong: B cannot check the format of the first message... before receiving the third!

Bibliography

- Martín Abadi and Roger Needham. *Prudent Engineering Practice for Cryptographic Protocols*. IEEE Transactions on Software Engineering, 22(1):2-15, 1996.
- Ross Anderson and Roger Needham. *Programming Satan's Computer.* In Computer Science Today, vol. 1000 of LNCS, p. 426-440. Springer, 1995.
- Cas Cremers and Sjouke Mauw. *Operational Semantics and Verification of Security Protocols*. Springer, 2012.
- John Clark and Jeremy Jacob. *A survey of authentication protocol literature*, 1997. http://www.cs.york.ac.uk/~jac/PublishedPapers/reviewV1_1997.pdf
- Catherine Meadows. *Open Issues in Formal Methods for Cryptographic Protocol Analysis*. Proceedings of DISCEX'00, 2000.