Security Protocols

Alessandro Armando

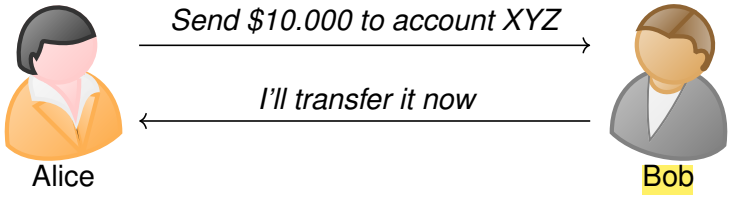
Computer Security Laboratory (CSec)
DIBRIS, Università di Genova

Computer Security

Outline

Motivation

Example: Securing an e-banking application.



- How does *Bob* know the message originated from *Alice*?
- How does *Bob* know *Alice* just said it?

attacker copia la req di alice, e dopo una settimana lo ritrasmette a Bob -> replay attack

Motivation – top down (II)

Other examples:

- Securing a sensor network
- Pairing of wireless devices
- An access control system for area-wide ski lifts
- Online payment systems

How would you build distributed algorithms for doing this?

Solutions involve protocols like: IPSec, SSH, PGP, SSL, Kerberos, etc.

We will focus on underlying ideas.

Motivation – top down (II)

Other examples:

- Securing a sensor network
- Pairing of wireless devices
- An access control system for area-wide ski lifts
- Online payment systems

How would you build distributed algorithms for doing this?

Solutions involve protocols like: IPSec, SSH, PGP, SSL, Kerberos, etc.

We will focus on underlying ideas.

Motivation – top down (II)

Other examples:

- Securing a sensor network
- Pairing of wireless devices
- An access control system for area-wide ski lifts
- Online payment systems

How would you build distributed algorithms for doing this?

Solutions involve protocols like: IPSec, SSH, PGP, SSL, Kerberos, etc.

We will focus on underlying ideas.

Outline

Definitions

- 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 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**.[1]

[1] https://www.cl.cam.ac.uk/~rja14/Papers/satan.pdf

Messages

- Message constructors are:

  Names: A, B or *Alice*, *Bob*, ...

  Keys: K and inverse keys K^{-1}

  Symmetric keys: $\{M\}_{K_{AB}}$, where K_{AB} is shared between (i.e. known only to) A and B

  Encryption: $\{M\}_K$, e.g. encryption with A's public key: $\{M\}_{K_A}$

  Signing: $\{M\}_{K^{-1}}$, e.g. "signing" with A's private key: $\{M\}_{K_A^{-1}}$

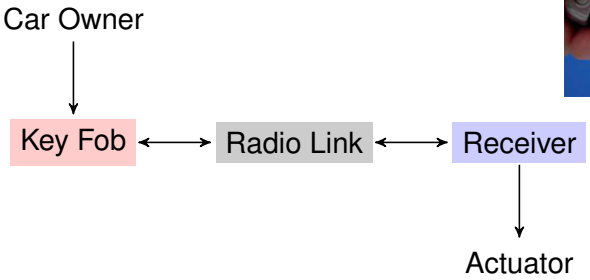
  Nonces: N_A, fresh data items used for challenge/response.

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

  Message concatenation: $\{M_1, M_2\}$, $M_1 \| M_2$, or $[M_1, M_2]$.

- Example: $\{A, T_A, K_{AB}\}_{K_B}$

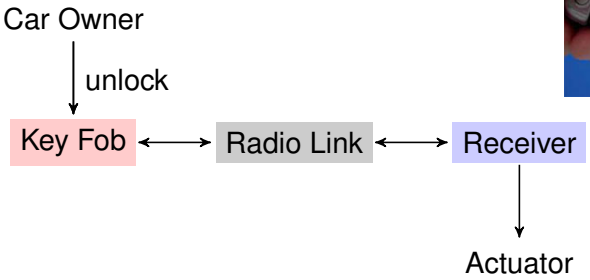
Example: Remote Keyless System



Security Goal (1st attempt)

Receiver sends unlock command to Actuator *only if* Car Owner *previously* pressed unlock button on Key Fob.

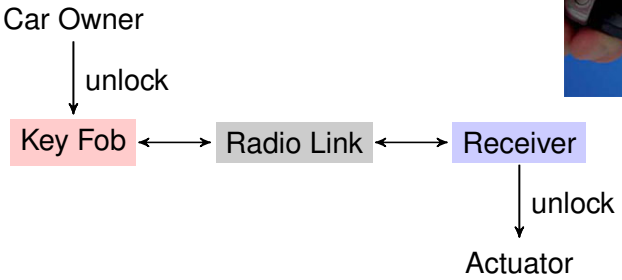
Example: Remote Keyless System



Security Goal (1st attempt)

Receiver sends unlock command to Actuator *only if* Car Owner *previously* pressed unlock button on Key Fob.

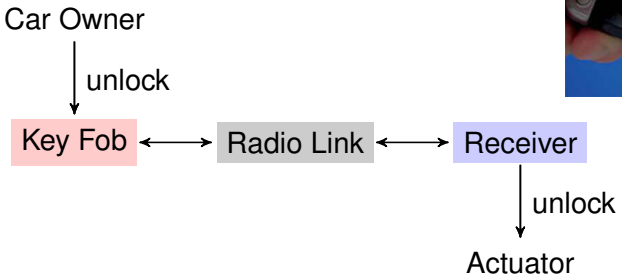
Example: Remote Keyless System



Security Goal (1st attempt)

Receiver sends unlock command to Actuator *only if* Car Owner *previously* pressed unlock button on Key Fob.

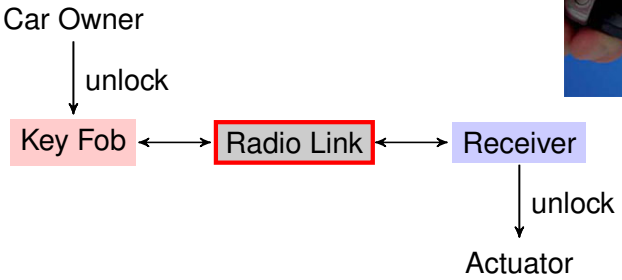
Example: Remote Keyless System



Security Goal (1st attempt)

Receiver sends unlock command to Actuator *only if* Car Owner *previously* pressed unlock button on Key Fob.

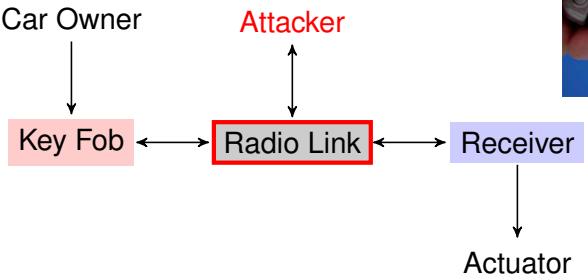
Example: Remote Keyless System



Security Goal (1st attempt)

Receiver sends unlock command to Actuator *only if* Car Owner *previously* pressed unlock button on Key Fob.

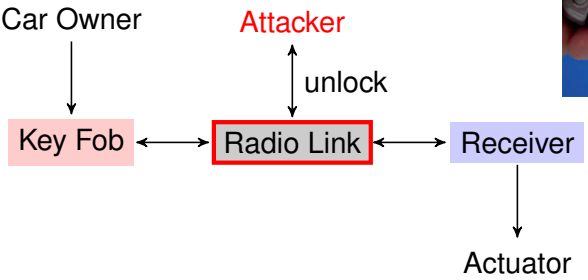
Example: Remote Keyless System



Security Goal (1st attempt)

Receiver sends unlock command to Actuator *only if* Car Owner *previously* pressed unlock button on Key Fob.

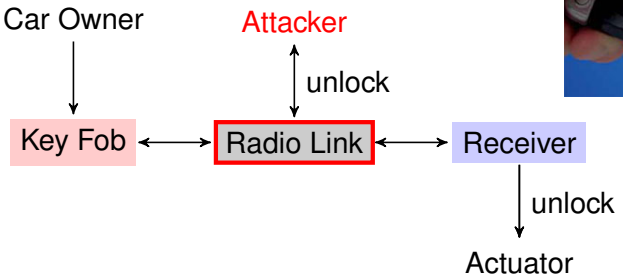
Example: Remote Keyless System



Security Goal (1st attempt)

Receiver sends unlock command to Actuator *only if* Car Owner *previously* pressed unlock button on Key Fob.

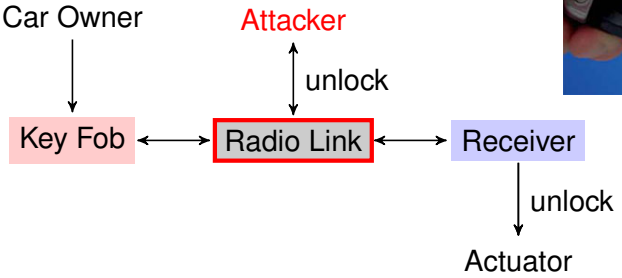
Example: Remote Keyless System



Security Goal (1st attempt)

Receiver sends unlock command to Actuator *only if* Car Owner *previously* pressed unlock button on Key Fob.

Example: Remote Keyless System



Security Goal (1st attempt)

Receiver sends unlock command to Actuator *only if* Car Owner *previously* pressed unlock button on Key Fob.

Remote Keyless System Protocol (1st attempt)

Assume Serial Number (SN)
is a secret shared between KF and R.

KF sends SN to R:

$$1.\ KF \rightarrow R : unlock, SN$$

- Bad idea: Attacker can easily overhear SN and *replay* it subsequently.
- Problems:
  - Secrecy of SN compromised.
  - R cannot check authenticity of request

Remote Keyless System Protocol (1st attempt)

Assume Serial Number (SN)
is a secret shared between KF and R.

KF sends SN to R:

$$1.\ KF \rightarrow R : unlock, SN$$

- Bad idea: Attacker can easily overhear SN and *replay* it subsequently.
- Problems:
  - Secrecy of SN compromised.
  - R cannot check authenticity of request

Remote Keyless System Protocol (1st attempt)

Assume Serial Number (SN)
is a secret shared between KF and R.

KF sends SN to R:

$$1.\ KF \rightarrow R : unlock, SN$$

- Bad idea: Attacker can easily overhear SN and *replay* it subsequently.
- Problems:
  - Secrecy of SN compromised.
  - R cannot check authenticity of request

Remote Keyless System Protocol (2nd attempt)

Idea: protect secrecy of SN

KF encrypts request with shared key (K) and sends the results to R.

1. $KF \rightarrow R : \{unlock, SN\}_K$

- Attacker can easily overhear encrypted request and replay it subsequently.
- Secrecy of SN ensured but attack possible anyway.
- R can check authenticity of request; but this is not enough...

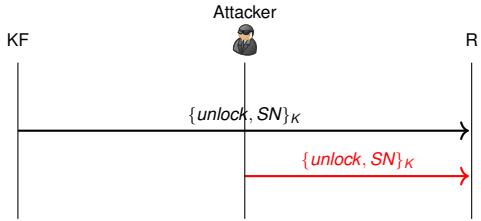
Remote Keyless System Protocol (2nd attempt)

Idea: protect secrecy of SN

KF encrypts request with shared key (K) and sends the results to R.

$$1.\ KF \rightarrow R : \{unlock, SN\}_K$$

- Attacker can easily overhear encrypted request and replay it subsequently.
- Secrecy of SN ensured but attack possible anyway.
- R can check authenticity of request; but this is not enough...



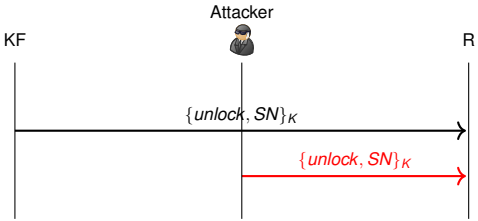
Remote Keyless System Protocol (2nd attempt)

Idea: protect secrecy of SN

KF encrypts request with shared key (K) and sends the results to R.

$$1.\ KF \rightarrow R : \{unlock, SN\}_K$$

- Attacker can easily overhear encrypted request and replay it subsequently.
- Secrecy of SN ensured but attack possible anyway.
- R can check authenticity of request; but this is not enough...



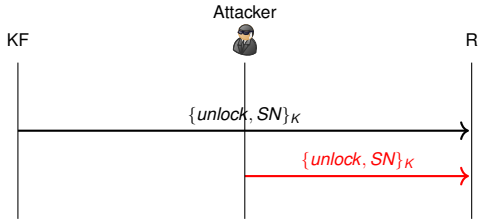
Remote Keyless System Protocol (2nd attempt)

Idea: protect secrecy of SN

KF encrypts request with shared key (K) and sends the results to R.

$$1.\ KF \rightarrow R : \{unlock, SN\}_K$$

- Attacker can easily overhear encrypted request and replay it subsequently.
- Secrecy of SN ensured but attack possible anyway.
- R can check authenticity of request; but this is not enough...

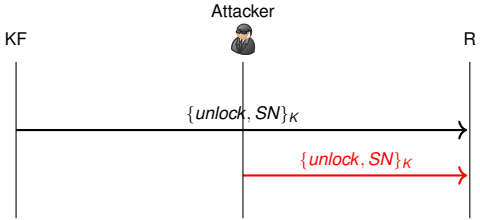


Security Goal: Freshness Requirement

The property:

Security Goal (1st attempt)

Receiver sends unlock command to Actuator *only if* Car Owner *previously* pressed unlock button on Key Fob.



is met by the protocol.

Yet, the protocol suffers from a *replay attack*.

Security Goal (revised)

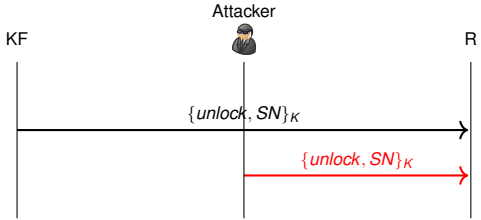
Receiver sends unlock command to Actuator *only if* Car Owner *recently* pressed unlock button on Key Fob.

Security Goal: Freshness Requirement

The property:

Security Goal (1st attempt)

Receiver sends unlock command to Actuator *only if* Car Owner *previously* pressed unlock button on Key Fob.



is met by the protocol.
Yet, the protocol suffers from a *replay attack*.

Security Goal (revised)

Receiver sends unlock command to Actuator *only if* Car Owner *recently* pressed unlock button on Key Fob.

Remote Keyless System Protocol (3rd attempt)

KF encrypts a timestamp with shared key K and sends result to R.

$$1.\ KF \rightarrow R : \{unlock, T\}_K$$

Remote Keyless System Protocol (3rd attempt)

KF encrypts a timestamp with shared key K and sends result to R.

$$1.\ KF \rightarrow R : \{unlock, T\}_K$$

- No need to send SN since KF can be identified by shared key.
- Timestamp prevents replay attack, but it requires synchronized clocks on KF and R.

Remote Keyless System Protocol (4th attempt)

Receiver (R) sends Key Fob (KF) a challenge (a nonce, N) and KF sends back N encrypted with shared key (K).

1. $KF \rightarrow R : hello$
2. $R \rightarrow KF : N$
3. $KF \rightarrow R : \{unlock, N\}_K$

Remote Keyless System Protocol (4th attempt)

Receiver (R) sends Key Fob (KF) a challenge (a nonce, N) and KF sends back N encrypted with shared key (K).

1. $KF \rightarrow R : hello$
2. $R \rightarrow KF : N$
3. $KF \rightarrow R : \{unlock, N\}_K$

- The usage of the nonce prevents replay attacks.
- Synchronized clocks on KF and R not needed, but additional steps are necessary.

Recommended Readings

- “How Remote Entry Works”, Howstuffworks. https://auto.howstuffworks.com/remote-entry2.htm (Accessed on Aug 25, 2019)
- “Hackers can steal a Tesla Model S in seconds by cloning its key fob”. Wired, 2018. https://www.wired.com/story/hackers-steal-tesla-model-s-seconds-key-fob/
- “Tesla Model S Stolen in 30 Seconds Using Keyless Hack”, PC Magazine, 2019. https://www.pcmag.com/news/370359/tesla-model-s-stolen-in-30-seconds-using-keyless-hack

Protocol Notation (Alice & Bob Notation)

- Fundamental events are communication between principals:

$$1.\ I \to R : \{I, T_I, K\}_{K_R}$$
$$2.\ R \to I : \{R, I\}_K$$

- I (initiator) and R (responder) name roles.
  Can be instantiated by any principal playing in the role, e.g.

$$1.\ a \to b : \{a, t_a, k\}_{k_b}$$
$$2.\ b \to a : \{b, a\}_k$$

$$1.\ b \to c : \{b, t_b, k'\}_{k_c}$$
$$2.\ c \to b : \{c, b\}_{k'}$$

- Communication is asynchronous
- Protocol specifies actions of principals.
  Equivalently, protocol defines a set of event sequences (traces).

Protocol Notation (Alice & Bob Notation)

- Fundamental events are communication between principals:

$$1.\ I \to R : \{I, T_I, K\}_{K_R}$$
$$2.\ R \to I : \{R, I\}_K$$

- I (initiator) and R (responder) name roles.
  Can be instantiated by any principal playing in the role, e.g.

$$1.\ a \to b : \{a, t_a, k\}_{k_b}$$
$$2.\ b \to a : \{b, a\}_k$$

$$1.\ b \to c : \{b, t_b, k'\}_{k_c}$$
$$2.\ c \to b : \{c, b\}_{k'}$$

- Communication is asynchronous
- Protocol specifies actions of principals.
  Equivalently, protocol defines a set of event sequences (traces).

Assumptions and Goals

Assumptions (for principals):

- Principals know their private keys and public keys of others
- Principals can generate/check nonces and timestamps, encrypt and decrypt with known keys
- (Honest) Principals correctly implement the protocol

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).

Assumptions: Attacker

How do we model the attacker? Possibilities:

- He knows the protocol but cannot break crypto. (Standard)
- He is passive but overhears all communications.
- He is active and can intercept and generate messages.
- He might even be one of the principals running the protocol!

Standard Attacker Model

- The attacker is active. Namely:
  - 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.
- Sometimes called the Dolev-Yao attacker model.
- Strongest possible assumptions about the attacker

$\Longrightarrow$

correct protocols function in the largest range of environments.

Kinds of attack

- Replay (or freshness) attack: reuse parts of previous messages.
- Man-in-the-middle (or parallel sessions) attack: $A \leftrightarrow \mathcal{M} \leftrightarrow B$.
- Reflection attack send transmitted information back to originator.
- Type flaw attack: substitute a different type of message field. Example: use a name (or a key or ...) as a nonce.

Outline

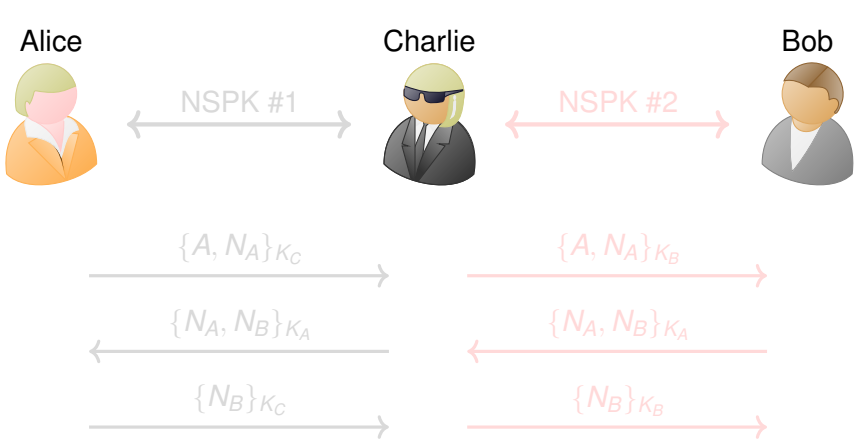
NSPK Protocol

1. $A \rightarrow B : \{A, N_A\}_{K_B}$
2. $B \rightarrow A : \{N_A, N_B\}_{K_A}$
3. $A \rightarrow B : \{N_B\}_{K_B}$

- **Goal:** mutual (entity) authentication.
- Correctness argument (informal).
  1. This is Alice and I have chosen a nonce N_{Alice}.
  2. Here is your Nonce N_{Alice}. Since I could read it, I must be Bob. I also have a challenge N_{Bob} for you.
  3. You sent me N_{Bob}. Since only Alice can read this and I sent it back, I must be Alice.
- Recall principals can be involved in multiple runs. Goal should hold in all interleaved protocol runs.

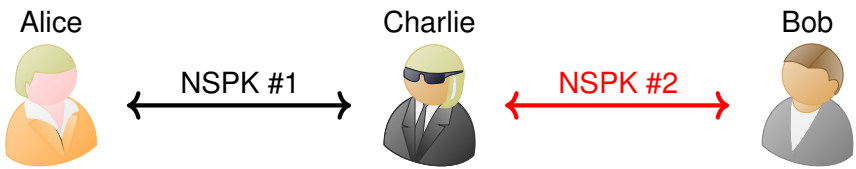
Protocol proposed in 1970s and used for decades.

Attack on NSPK



1. $A \rightarrow B : \{A, N_A\}_{K_B}$
2. $B \rightarrow A : \{N_A, N_B\}_{K_A}$
3. $A \rightarrow B : \{N_B\}_{K_B}$

Attack on NSPK



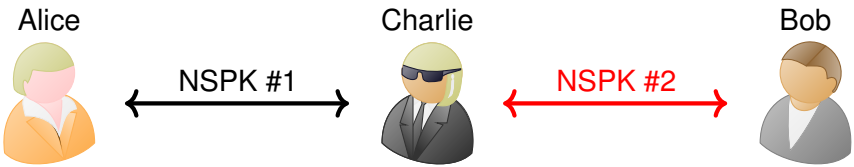
1. $A \to B : \{A, N_A\}_{K_B}$
2. $B \to A : \{N_A, N_B\}_{K_A}$
3. $A \to B : \{N_B\}_{K_B}$

$\{A, N_A\}_{K_C}$ $\quad$ $\{A, N_A\}_{K_B}$

$\{N_A, N_B\}_{K_A}$ $\quad$ $\{N_A, N_B\}_{K_A}$

$\{N_B\}_{K_C}$ $\quad$ $\{N_B\}_{K_B}$

Attack on NSPK



1. $A \to B : \{A, N_A\}_{K_B}$
2. $B \to A : \{N_A, N_B\}_{K_A}$
3. $A \to B : \{N_B\}_{K_B}$

$\{A, N_A\}_{K_C}$

$\{N_A, N_B\}_{K_A}$

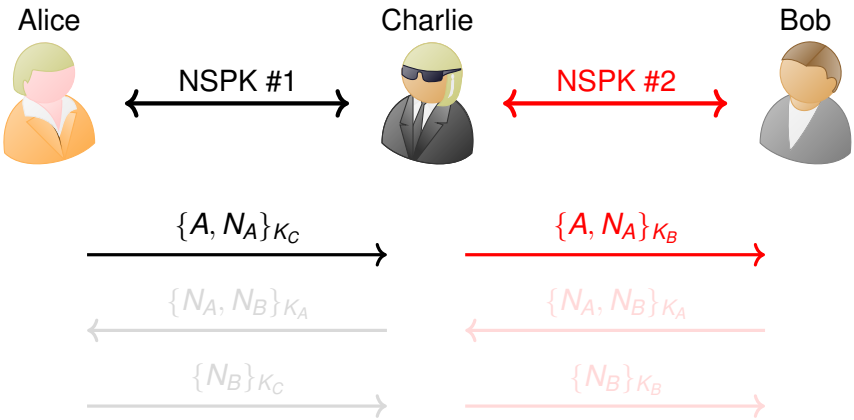
$\{N_B\}_{K_C}$

$\{A, N_A\}_{K_B}$

$\{N_A, N_B\}_{K_A}$

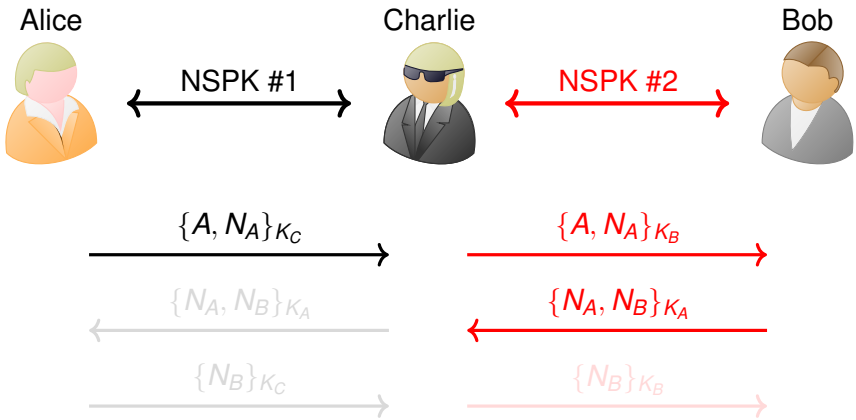
$\{N_B\}_{K_B}$

Attack on NSPK



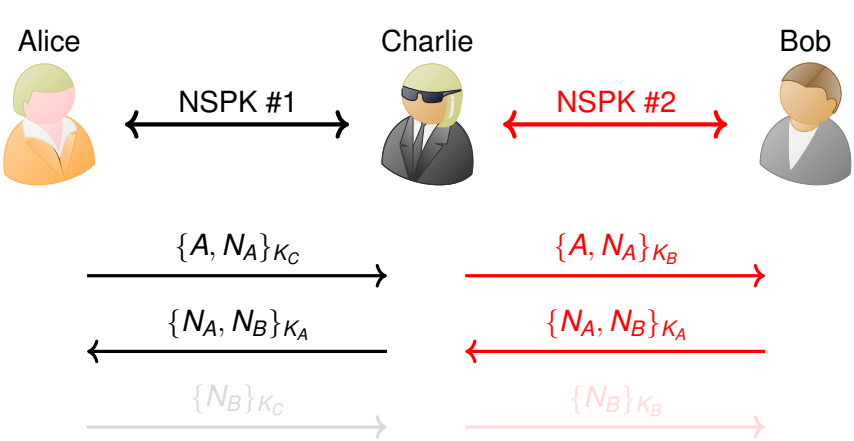
1. $A \to B : \{A, N_A\}_{K_B}$
2. $B \to A : \{N_A, N_B\}_{K_A}$
3. $A \to B : \{N_B\}_{K_B}$

Attack on NSPK



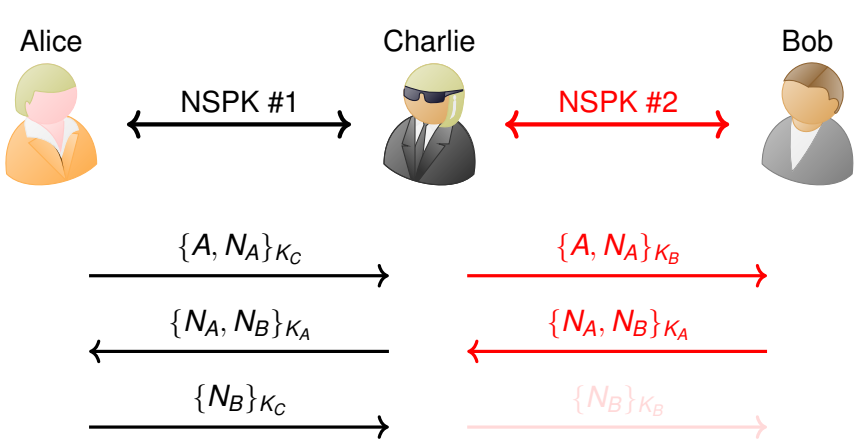
1. $A \rightarrow B : \{A, N_A\}_{K_B}$
2. $B \rightarrow A : \{N_A, N_B\}_{K_A}$
3. $A \rightarrow B : \{N_B\}_{K_B}$

Attack on NSPK



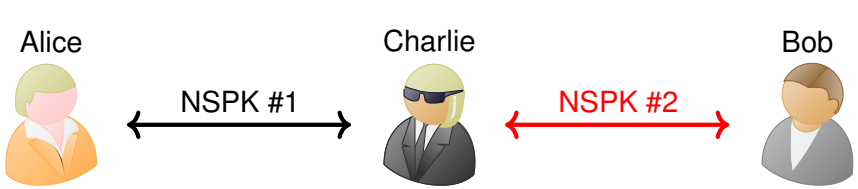
1. $A \to B : \{A, N_A\}_{K_B}$
2. $B \to A : \{N_A, N_B\}_{K_A}$
3. $A \to B : \{N_B\}_{K_B}$

Attack on NSPK



1. $A \to B : \{A, N_A\}_{K_B}$
2. $B \to A : \{N_A, N_B\}_{K_A}$
3. $A \to B : \{N_B\}_{K_B}$

Attack on NSPK



1. $A \to B : \{A, N_A\}_{K_B}$
2. $B \to A : \{N_A, N_B\}_{K_A}$
3. $A \to B : \{N_B\}_{K_B}$

$\{A, N_A\}_{K_C}$ $\quad$ $\{A, N_A\}_{K_B}$

$\{N_A, N_B\}_{K_A}$ $\quad$ $\{N_A, N_B\}_{K_A}$

$\{N_B\}_{K_C}$ $\quad$ $\{N_B\}_{K_B}$

B believes he is speaking with *A*!

NSPK Protocol with Lowe's Fix

How can we protect against the previous attack?

1. $A \to B : \{A, N_A\}_{K_B}$
2. $B \to A : \{N_A, N_B, B\}_{K_A}$
3. $A \to B : \{N_B\}_{K_B}$

NSPK Protocol with Lowe's Fix

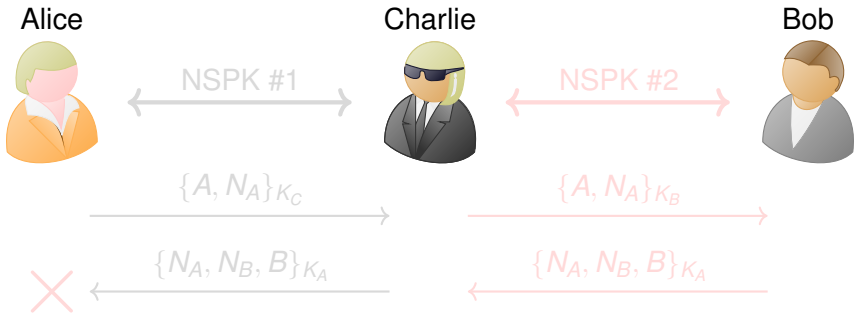
How can we protect against the previous attack?

1. $A \to B : \{A, N_A\}_{K_B}$
2. $B \to A : \{N_A, N_B, B\}_{K_A}$
3. $A \to B : \{N_B\}_{K_B}$

NSPK Protocol with Lowe's Fix

How can we protect against the previous attack?

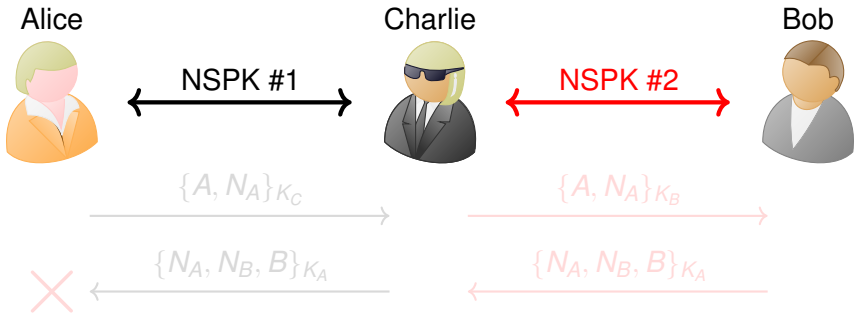
1. $A \to B : \{A, N_A\}_{K_B}$
2. $B \to A : \{N_A, N_B, B\}_{K_A}$
3. $A \to B : \{N_B\}_{K_B}$



NSPK Protocol with Lowe's Fix

How can we protect against the previous attack?

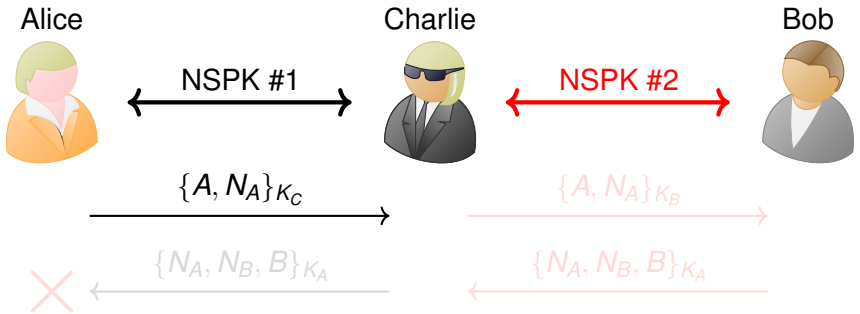
1. $A \to B : \{A, N_A\}_{K_B}$
2. $B \to A : \{N_A, N_B, B\}_{K_A}$
3. $A \to B : \{N_B\}_{K_B}$



NSPK Protocol with Lowe's Fix

How can we protect against the previous attack?

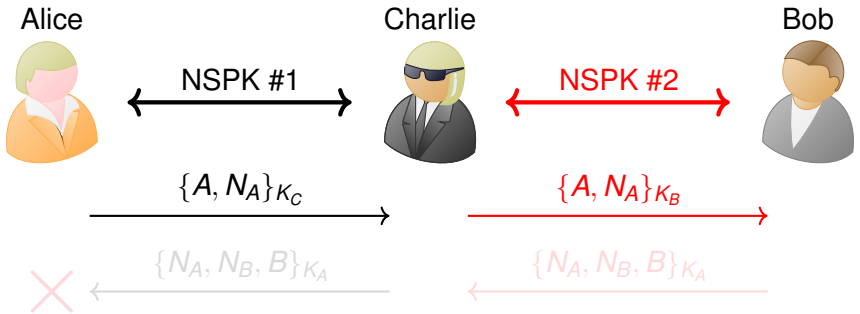
1. $A \to B : \{A, N_A\}_{K_B}$
2. $B \to A : \{N_A, N_B, B\}_{K_A}$
3. $A \to B : \{N_B\}_{K_B}$



NSPK Protocol with Lowe's Fix

How can we protect against the previous attack?

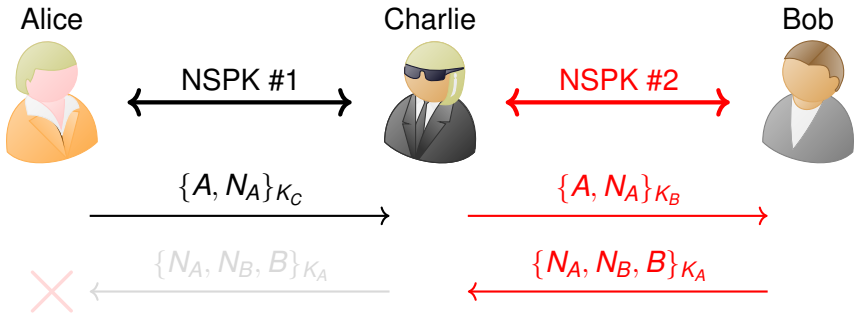
1. $A \to B : \{A, N_A\}_{K_B}$
2. $B \to A : \{N_A, N_B, B\}_{K_A}$
3. $A \to B : \{N_B\}_{K_B}$



NSPK Protocol with Lowe's Fix

How can we protect against the previous attack?

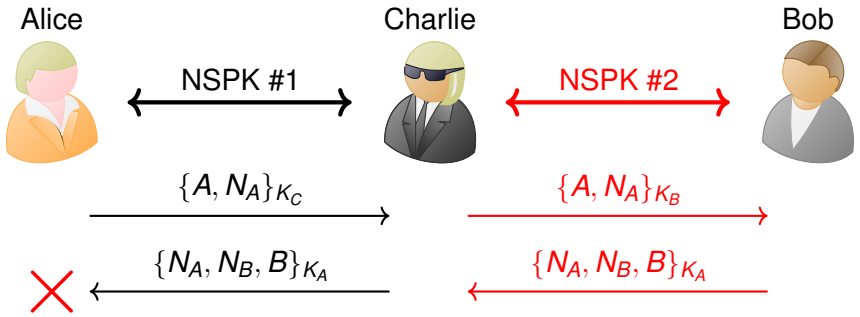
1. $A \to B : \{A, N_A\}_{K_B}$
2. $B \to A : \{N_A, N_B, B\}_{K_A}$
3. $A \to B : \{N_B\}_{K_B}$



NSPK Protocol with Lowe's Fix

How can we protect against the previous attack?

1. $A \to B : \{A, N_A\}_{K_B}$
2. $B \to A : \{N_A, N_B, B\}_{K_A}$
3. $A \to B : \{N_B\}_{K_B}$



Type flaw attacks

- A message consists of a sequence of submessages.
  Examples: an principal's name, a nonce, a key, ...
- Messages sent as bit strings. No 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.

Outline

The Otway-Rees protocol

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

$$
\begin{array}{lll}
\text{M1.} & A \rightarrow B: & I, A, B, \{N_A, I, A, B\}_{K_{AS}} \\
\text{M2.} & B \rightarrow S: & I, A, B, \{N_A, I, A, B\}_{K_{AS}}, \{N_B, I, A, B\}_{K_{BS}} \\
\text{M3.} & S \rightarrow B: & I, \{N_A, K_{AB}\}_{K_{AS}}, \{N_B, K_{AB}\}_{K_{BS}} \\
\text{M4.} & B \rightarrow A: & I, \{N_A, K_{AB}\}_{K_{AS}}
\end{array}
$$

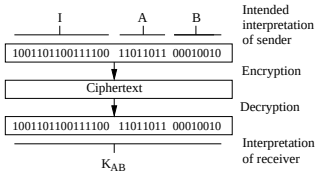
where server keys already known and I is protocol run identifier (e.g., an integer).

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

Type flaw attack on the Otway-Rees protocol

$$
\begin{array}{lll}
\text{M1.} & A \to B: & I, A, B, \{N_A, I, A, B\}_{K_{AS}} \\
\text{M2.} & B \to S: & I, A, B, \{N_A, I, A, B\}_{K_{AS}}, \{N_B, I, A, B\}_{K_{BS}} \\
\text{M3.} & S \to B: & I, \{N_A, K_{AB}\}_{K_{AS}}, \{N_B, K_{AB}\}_{K_{BS}} \\
\text{M4.} & B \to A: & I, \{N_A, K_{AB}\}_{K_{AS}}
\end{array}
$$

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



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

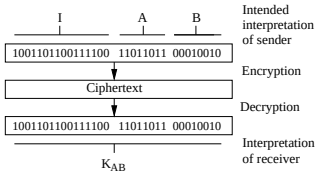
$$
\begin{array}{lll}
\text{M1.} & A \to \mathcal{M}(B): & I, A, B, \{N_A, I, A, B\}_{K_{AS}} \\
\text{M4.} & \mathcal{M}(B) \to A: & I, \{N_A, I, A, B\}_{K_{AS}}
\end{array}
$$

A sees N_A and wrongly accepts $\{I, A, B\}$ as the session key.

Type flaw attack on the Otway-Rees protocol

$$
\begin{array}{lll}
\text{M1.} & A \to B: & I, A, B, \{N_A, I, A, B\}_{K_{AS}} \\
\text{M2.} & B \to S: & I, A, B, \{N_A, I, A, B\}_{K_{AS}}, \{N_B, I, A, B\}_{K_{BS}} \\
\text{M3.} & S \to B: & I, \{N_A, K_{AB}\}_{K_{AS}}, \{N_B, K_{AB}\}_{K_{BS}} \\
\text{M4.} & B \to A: & I, \{N_A, K_{AB}\}_{K_{AS}}
\end{array}
$$

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



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

$$
\begin{array}{lll}
\text{M1.} & A \to \mathcal{M}(B): & I, A, B, \{N_A, I, A, B\}_{K_{AS}} \\
\text{M4.} & \mathcal{M}(B) \to A: & I, \{N_A, I, A, B\}_{K_{AS}}
\end{array}
$$

A sees N_A and wrongly accepts $\{I, A, B\}$ as the session key.

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

$$
\begin{array}{lll}
\text{M1.} & A \to B: & I, A, B, \{N_A, I, A, B\}_{K_{AS}} \\
\text{M2.} & B \to S: & I, A, B, \{N_A, I, A, B\}_{K_{AS}}, \{N_B, I, A, B\}_{K_{BS}} \\
\text{M3.} & S \to B: & I, \{N_A, K_{AB}\}_{K_{AS}}, \{N_B, K_{AB}\}_{K_{BS}} \\
\text{M4.} & B \to A: & I, \{N_A, K_{AB}\}_{K_{AS}}
\end{array}
$$

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

$$
\begin{array}{lll}
\text{M1.} & A \to B: & I, A, B, \{N_A, I, A, B\}_{K_{AS}} \\
\text{M2.} & B \to \mathcal{M}(S): & I, A, B, \{N_A, I, A, B\}_{K_{AS}}, \{N_B, I, A, B\}_{K_{BS}} \\
\text{M3.} & \mathcal{M}(S) \to B: & I, \{N_A, I, A, B\}_{K_{AS}}, \{N_B, I, A, B\}_{K_{BS}} \\
\text{M4.} & B \to A: & I, \{N_A, I, A, B\}_{K_{AS}}
\end{array}
$$

$\Rightarrow$ A and B accept wrong key and $\mathcal{M}$ can decrypt their subsequent communication! So key authentication (and secrecy) fails!

How might we protect against this kind of attack?

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

$$
\begin{array}{lll}
\text{M1.} & A \to B: & I, A, B, \{N_A, I, A, B\}_{K_{AS}} \\
\text{M2.} & B \to S: & I, A, B, \{N_A, I, A, B\}_{K_{AS}}, \{N_B, I, A, B\}_{K_{BS}} \\
\text{M3.} & S \to B: & I, \{N_A, K_{AB}\}_{K_{AS}}, \{N_B, K_{AB}\}_{K_{BS}} \\
\text{M4.} & B \to A: & I, \{N_A, K_{AB}\}_{K_{AS}}
\end{array}
$$

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

$$
\begin{array}{lll}
\text{M1.} & A \to B: & I, A, B, \{N_A, I, A, B\}_{K_{AS}} \\
\text{M2.} & B \to \mathcal{M}(S): & I, A, B, \{N_A, I, A, B\}_{K_{AS}}, \{N_B, I, A, B\}_{K_{BS}} \\
\text{M3.} & \mathcal{M}(S) \to B: & I, \{N_A, I, A, B\}_{K_{AS}}, \{N_B, I, A, B\}_{K_{BS}} \\
\text{M4.} & B \to A: & I, \{N_A, I, A, B\}_{K_{AS}}
\end{array}
$$

$\Rightarrow$ A and B accept wrong key and $\mathcal{M}$ can decrypt their subsequent communication! So key authentication (and secrecy) fails!

How might we protect against this kind of attack?

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

$$
\begin{array}{lll}
\text{M1.} & A \to B: & I, A, B, \{N_A, I, A, B\}_{K_{AS}} \\
\text{M2.} & B \to S: & I, A, B, \{N_A, I, A, B\}_{K_{AS}}, \{N_B, I, A, B\}_{K_{BS}} \\
\text{M3.} & S \to B: & I, \{N_A, K_{AB}\}_{K_{AS}}, \{N_B, K_{AB}\}_{K_{BS}} \\
\text{M4.} & B \to A: & I, \{N_A, K_{AB}\}_{K_{AS}}
\end{array}
$$

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

$$
\begin{array}{lll}
\text{M1.} & A \to B: & I, A, B, \{N_A, I, A, B\}_{K_{AS}} \\
\text{M2.} & B \to \mathcal{M}(S): & I, A, B, \{N_A, I, A, B\}_{K_{AS}}, \{N_B, I, A, B\}_{K_{BS}} \\
\text{M3.} & \mathcal{M}(S) \to B: & I, \{N_A, I, A, B\}_{K_{AS}}, \{N_B, I, A, B\}_{K_{BS}} \\
\text{M4.} & B \to A: & I, \{N_A, I, A, B\}_{K_{AS}}
\end{array}
$$

$\Rightarrow$ A and B accept wrong key and $\mathcal{M}$ can decrypt their subsequent communication! So key authentication (and secrecy) fails!

How might we protect against this kind of attack?

Outline

The Andrew Secure RPC protocol

Goal: Exchange a fresh, authenticated, secret, shared key, between two principals sharing a symmetric key.

$$
\begin{array}{lll}
\text{M1.} & A \to B: & A, \{N_A\}_{K_{AB}} \\
\text{M2.} & B \to A: & \{N_A + 1, N_B\}_{K_{AB}} \\
\text{M3.} & A \to B: & \{N_B + 1\}_{K_{AB}} \\
\text{M4.} & B \to A: & \{K'_{AB}, N'_B\}_{K_{AB}}
\end{array}
$$

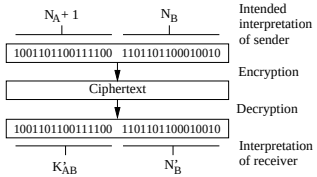
Establishes session key K'_{AB} plus nonce N'_B (for a future session).

Type flaw attack on Andrew Secure RPC protocol

M1.	$A \to B:$	$A, \{N_A\}_{K_{AB}}$	M1.	$A \to B:$	$A, \{N_A\}_{K_{AB}}$
M2.	$B \to A:$	$\{N_A + 1, N_B\}_{K_{AB}}$	M2.	$B \to A:$	$\{N_A + 1, N_B\}_{K_{AB}}$
M3.	$A \to B:$	$\{N_B + 1\}_{K_{AB}}$	M3.	$A \to \mathcal{M}(B):$	$\{N_B + 1\}_{K_{AB}}$
M4.	$B \to A:$	$\{K'_{AB}, N'_B\}_{K_{AB}}$	M4.	$\mathcal{M}(B) \to A:$	$\{N_A + 1, N_B\}_{K_{AB}}$

Assume: nonces and keys are represented as bit sequences of the same length (e.g. 64 bits).

Mallory can then record M2, intercept M3, and replay M2 as M4.



A is fooled into accepting the nonce value $N_A + 1$ as the new session key. This key is not authenticated.

This attack doesn't violate secrecy though (cf. Otway-Rees).

Outline

Key exchange with CA (Denning & Sacco)

$$
\begin{aligned}
A \rightarrow S: &\quad A, B \\
S \rightarrow A: &\quad C_A, C_B \\
A \rightarrow B: &\quad C_A, C_B, \{\{T_A, K_{AB}\}_{K_A^{-1}}\}_{K_B}
\end{aligned}
$$

Explanations:

S**:** Server for a certification authority.

K_{AB}**:** Secret session key.

K_B**:** Public key of B, the intended communication partner.

K_A^{-1}**:** Signing with private key of A.

C_A, C_B**:** Certificates for A and B (name, public key, ...)

T_A**:** A timestamp generated by A.

Denning & Sacco (II)

$$
\begin{aligned}
A \to S: &\quad A, B \\
S \to A: &\quad C_A, C_B \\
A \to B: &\quad C_A, C_B, \{\{T_A, K_{AB}\}_{K_A^{-1}}\}_{K_B}
\end{aligned}
$$

Intended purpose:

- K_{AB} is only known to A and B (session key)
- B can be sure that it was sent by A:
  - he can decrypt it using the public key K_A.
  - K_A is bound to A by the certificate C_A.
- B knows that the message was intended for him/her because his/her public key is used.
- A knows ...?
- Timestamp may be used to limit use of session key.

Denning & Sacco (III)

- The attack

$$
\begin{aligned}
A \to S: &\quad A, B \\
S \to A: &\quad C_A, C_B \\
A \to B: &\quad C_A, C_B, \{\{T_A, K_{AB}\}_{K_A^{-1}}\}_{K_B}
\end{aligned}
$$

$$
\begin{aligned}
alice \to charlie: &\quad C_{alice}, C_{charlie}, \{\{T_{alice}, K_{alice\ charlie}\}_{K_{alice}^{-1}}\}_{K_{charlie}} \\
charlie \to bob: &\quad C_{alice}, C_{bob}, \{\{T_{alice}, K_{alice\ charlie}\}_{K_{alice}^{-1}}\}_{K_{bob}}
\end{aligned}
$$

- Bob believes the last message was sent by Alice
  - When he later uses $K_{alice\ charlie}$, Charlie can listen in.
  - So key is neither authenticated nor secret!
- How can we defend against this man-in-the-middle attack?
  Be explicit about purpose: name principals in last step.

$$A \to B: \quad C_A, C_B, \{\{A, B, T_A, K_{AB}\}_{K_A^{-1}}\}_{K_B}$$

Denning & Sacco (III)

- The attack

$$
\begin{aligned}
A \to S: &\quad A, B \\
S \to A: &\quad C_A, C_B \\
A \to B: &\quad C_A, C_B, \{\{T_A, K_{AB}\}_{K_A^{-1}}\}_{K_B}
\end{aligned}
$$

$$
\begin{aligned}
alice \to charlie: &\quad C_{alice}, C_{charlie}, \{\{T_{alice}, K_{alice\ charlie}\}_{K_{alice}^{-1}}\}_{K_{charlie}} \\
charlie \to bob: &\quad C_{alice}, C_{bob}, \{\{T_{alice}, K_{alice\ charlie}\}_{K_{alice}^{-1}}\}_{K_{bob}}
\end{aligned}
$$

- Bob believes the last message was sent by Alice
  - When he later uses $K_{alice\ charlie}$, Charlie can listen in.
  - So key is neither authenticated nor secret!
- How can we defend against this man-in-the-middle attack?
  Be explicit about purpose: name principals in last step.

$$A \to B: \quad C_A, C_B, \{\{A, B, T_A, K_{AB}\}_{K_A^{-1}}\}_{K_B}$$

Outline

Prudent engineering of security protocols

Principles proposed by Abadi and Needham (1994, 1995):

1. Every message should say what it means.
2. The conditions for a message to be acted on should be stated.
3. Mention the principal's name explicitly in the message if it is essential to the meaning.
4. Be clear as to why encryption is being done.
   Confidentiality, message authentication, binding of messages,
   e.g, $\{X, Y\}_{K^{-1}}$ versus $\{X\}_{K^{-1}}, \{Y\}_{K^{-1}}$

Prudent engineering (II)

6. Be clear on what properties you are assuming about nonces.
7. Predictable quantities used for challenge-response should be protected from replay.
8. Timestamps must take into account local clock variation and clock maintenance mechanisms.
9. A key may have been used recently, yet be old.
10. If an encoding is used to present the meaning of a message, then it should be possible to tell which encoding is being used.
11. The protocol designer should know which trust relations his protocol depends on.

Prudent engineering (III)

Good advice, but ...

- Are you sure when you have followed all of them?
- Is the protocol guaranteed to be secure then?
- Is it optimal and/or minimal?

Prudent engineering: Conclusion

- Security protocols can achieve properties that cryptographic primitives alone cannot offer.
- The examples shown are simple, but the ideas are general.
  More advanced protocols coming shortly.
- Even three liners show how difficult the art of correct design is.
  Can we raise this to a science?

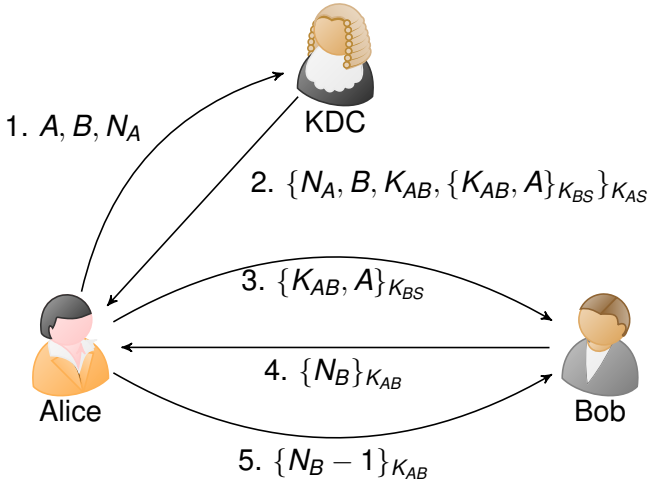
Outline

Automated Analysis of Security Protocols

- Security protocols are notoriously difficult to get right.
- Traditional verification techniques (e.g. human inspection, testing) do not ensure the needed level of assurance.
- Automated techniques can
  - systematically explore the state space of the system and guarantee a complete coverage of its behaviors
  - support the implementation of secure solutions/mechanisms
- A number of tools for the automatic analysis of security protocols has emerged over the years: Casper/FDR, NRL, AVISPA Tool (comprising OFMC, CL-AtSe, SATMC, and TA4SP), AVANTSSAR Platform, SPaCIoS Tool, Proverif, Scyther, ...

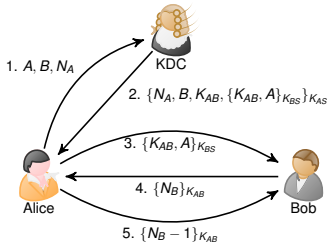
Outline

Needham-Schroeder Shared-Key Protocol



Security Goal: Authenticated key exchange.

Weakness of NSSK



- B cannot check freshness of message sent at step 3.
- Steps 4 and 5 prevent ensure (to B) that A is currently committed to using the session key K_{AB}.
- Yet, if a previous session key, say K'_{AB} gets compromised, then the attacker makes B accept K_{AB} again by replaying $\{K'_{AB}, A\}_{K_{BS}}$.
- This weakness can be fixed by the inclusion of a timestamp.

Outline

Kerberos

- Protocol for authentication/access control for client/server applications.
- In Greek mythology, Kerberos is 3-headed dog guarding entrance to Hades.
  Modern Kerberos intended to have three components to guard a network's gate: authentication, accounting, and audit.
  Last two heads never implemented.
- Developed as part of Project Athena (MIT, 1980's).
  - Version V (1993) now standard.
  - Widely used, e.g., Microsoft Windows and Microsoft Active Directory.

Kerberos Authentication Protocol: Aims (1/2)

(From: `https://www.kerberos.org/software/tutorial.html`)

- The user's password must never travel over the network;
- The user's password must never be stored in any form on the client machine: it must be immediately discarded after being used;
- The user's password should never be stored in an unencrypted form even in the authentication server database;
- Single Sign-On: The user is asked to enter a password only once per work session. Therefore users can transparently access all the services they are authorized for without having to re-enter the password during this session;

Kerberos Authentication Protocol: Aims (2/2)

- Authentication information resides on the authentication server only. The application servers must not contain the authentication information for their users. This is essential for obtaining the following results:
  - The administrator can disable the account of any user by acting in a single location without having to act on the several application servers;
  - When a user changes its password, it is changed for all services at the same time;
  - There is no redundancy of authentication information which would otherwise have to be safeguarded in various places;
- Mutual Authentication: not only do the users have to demonstrate that they are who they say, but, when requested, the application servers must prove their authenticity to the client as well.
- Following the completion of authentication and authorization, the client and server must be able to establish an encrypted connection.

Kerberos – requirements

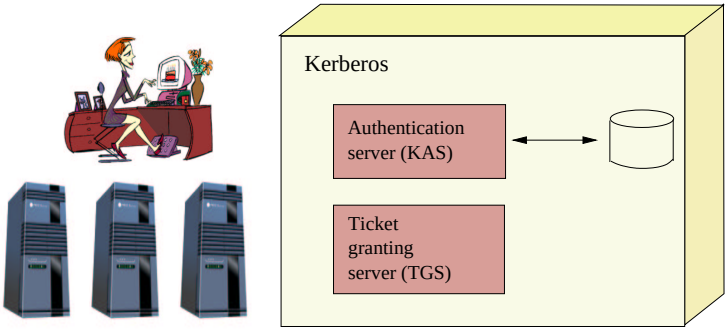
Secure: An eavesdropper should not be able to obtain information to impersonate a user.

Reliable: As many services depend on Kerberos for access control, it must be highly reliable and support a distributed architecture, where one system can back up another.

Transparent: Each user should enter a single password to obtain network services; otherwise he should be unaware of underlying protocols.

Scalable: The system should scale to support large numbers of users and servers; this suggests a modular, distributed architecture.

Kerberos version IV: Architecture

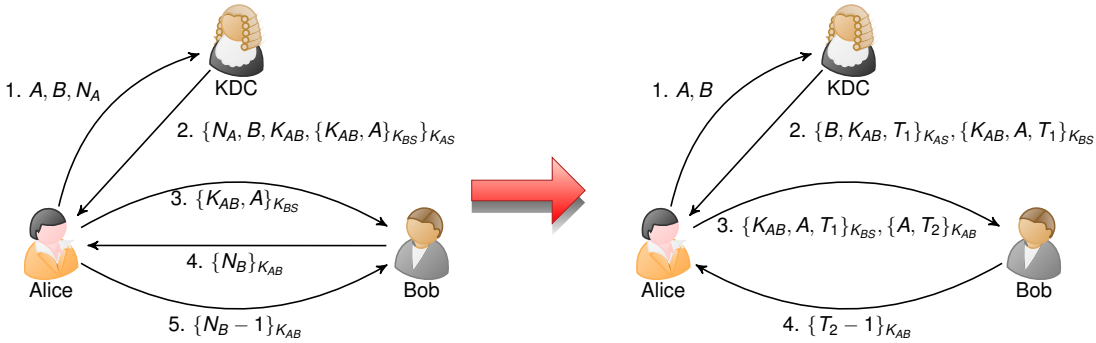


Authentication using KAS, the Kerberos Authentication Server.
Authorization using TGS, the Ticket Granting Server.
Access Control where servers check TGS tickets.

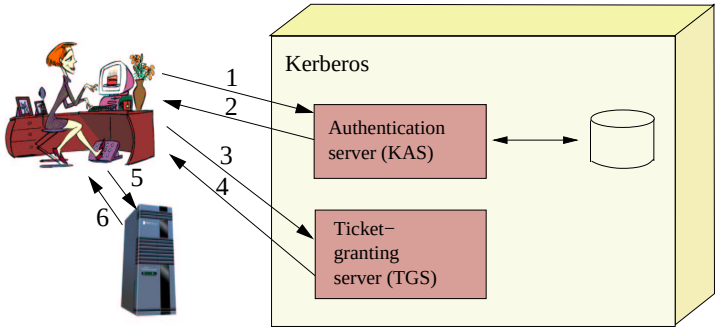
Kerberos Authentication Protocol

Loosely based on the Needham-Schroeder Shared-Key Protocol:

- Timestamps instead of nonces to assure freshness of session keys.
- Removal of nested encryption.



Kerberos IV: protocol



Authentication	messages 1 and 2.	Once per user login session.
Authorization	messages 3 and 4.	Once per type of service.
Service	messages 5 and 6.	Once per service session.

We present the three parts below (slightly simplified).

Authentication phase

1. $A \rightarrow KAS \quad : \quad A, TGS$
2. $KAS \rightarrow A \quad : \quad \{K_{A,TGS}, TGS, \mathcal{T}_1\}_{K_{AS}}, \underbrace{\{A, TGS, K_{A,TGS}, \mathcal{T}_1\}_{K_{KAS,TGS}}}_{AuthTicket}$

- A logs onto workstation and requests network resources.
- KAS accesses database and sends A a session key $K_{A,TGS}$ and an encrypted ticket $AuthTicket$.
- $K_{A,TGS}$ has lifetime of several hours (depending on application).
- K_{AS} is derived from the user's password, i.e. $K_{AS} = h(Password_A \| A)$.
- Both user and server keys must be registered in database.
- A types password on client to decrypt results. The ticket and session key are saved. The user's password is forgotten. A is logged out when $K_{A,TGS}$ expires.

Authorization phase

3. $\mathrm{A} \rightarrow \mathrm{TGS}: \underbrace{\{A,\ TGS,\ K_{A,TGS},\ \mathcal{T}_1\}_{K_{KAS,TGS}}}_{AuthTicket}, \underbrace{\{A,\ \mathcal{T}_2\}_{K_{A,TGS}}}_{authenticator}, \mathrm{B}$

4. $\mathrm{TGS} \rightarrow \mathrm{A}: \{K_{AB}, B, \mathcal{T}_3\}_{K_{A,TGS}}, \underbrace{\{A,\ B,\ K_{AB},\ \mathcal{T}_3\}_{K_{BS}}}_{ServTicket}$

Before A's first access of network resource B:

- A presents *AuthTicket* from message 2 to TGS together with a new *authenticator*, with short (seconds) lifetime.
  - Role of authenticator? Short validity prevent replay attacks.
  - Servers store recent authenticators to prevent immediate replay.
- TGS issues A a new session key K_{AB} (lifetime of few minutes) and a new ticket *ServTicket*. K_{BS} is key shared between TGS and network resource.

Authorization phase

3. $\mathrm{A} \rightarrow \mathrm{TGS}: \underbrace{\{A,\ TGS,\ K_{A,TGS},\ \mathcal{T}_1\}_{K_{KAS,TGS}}}_{AuthTicket}, \underbrace{\{A,\ \mathcal{T}_2\}_{K_{A,TGS}}}_{authenticator}, \mathrm{B}$

4. $\mathrm{TGS} \rightarrow \mathrm{A}: \{K_{AB}, B, \mathcal{T}_3\}_{K_{A,TGS}}, \underbrace{\{A,\ B,\ K_{AB},\ \mathcal{T}_3\}_{K_{BS}}}_{ServTicket}$

Before A's first access of network resource B:

- A presents *AuthTicket* from message 2 to TGS together with a new *authenticator*, with short (seconds) lifetime.
  - Role of authenticator? Short validity prevent replay attacks.
  - Servers store recent authenticators to prevent immediate replay.
- TGS issues A a new session key K_{AB} (lifetime of few minutes) and a new ticket *ServTicket*. K_{BS} is key shared between TGS and network resource.

Service phase

5. $A \rightarrow B: \underbrace{\{A,\ B,\ K_{AB},\ \mathcal{T}_3\}_{K_{BS}}}_{ServTicket}, \underbrace{\{A, \mathcal{T}_4\}_{K_{AB}}}_{authenticator}$
6. $B \rightarrow A: \{\mathcal{T}_4 + 1\}_{K_{AB}}$

For A to access network resource B:

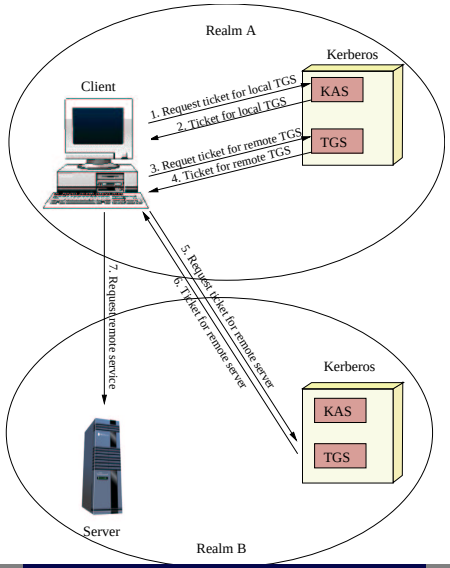
- A presents K_{AB} from 4 to B along with new *authenticator*.

  In practice, other information for server might be sent too.
- B replies, authenticating service.

Scalability – Multiple Realms/Kerberi

- A realm is defined by a Kerberos server.
  Server stores user and application server passwords for realm.
- Large networks may be divided into administrative realms.
- Kerberos supports inter-realm protocols.
  - Servers register with each other.
  - For A to access B in another realm, the TGS in A's realm receives request and grants ticket to access TGS in B's realm.
- Protocol extension simple: two additional steps.
  But for n realms, $O(n^2)$ key distribution problem (in Version IV).

Inter-realm communication



Limitations of Kerberos IV

- M1: encryption is not needed, but attacker can flood S

$$A \rightarrow KAS: \ A, \ TGS$$

- M2: nested encryption unnecessary; eliminated in Kerberos V.

$$KAS \rightarrow A: \ \{K_{A,TGS}, \ TGS, \ \mathcal{T}_1, \ \{A, \ TGS, \ K_{A,TGS}, \ \mathcal{T}_1\}_{K_{KAS,TGS}}\}_{K_{AS}}$$

- Relies on (roughly) synchronized and uncompromised clocks.
  If the host is compromised, then the clock can be compromised and replay is easy.

Some of these limitations apply also to Kerberos V.

Obtaining and using Kerberos

- Obtaining Kerberos:
  - MIT: `http://web.mit.edu/kerberos/www/`.
  - Distributions for Linux and Windows 2000.
- Many protocols/applications have been “kerberized” including:
  - Mobile IP (SeaMoby), Resource Reservation Protocol (RSVP),
  - Telnet, FTP, SNMP, TLS,
  - DHCP, GSS-API, SASL, DMS (via TKEY),
  - Windows 2000 for all authentication procedures, and
  - IKE (running IKE Phase 2, over Artifical Kerberos IKE SA),

Conclusions

Security protocols are

- key to secure distributed systems
- ubiquitous: they span virtually all application domains and the whole protocol stack
- (deceptively) simple: use established protocols when available/possible, but be ready to design/develop/use a new one if necessary needed.