CYBER CHALLENGE
CyberChallenge.IT

cini
Cybersecurity National Lab

SPONSOR PLATINUM

accenturesecurity

aizoon AUSTRALIA EUROPE USA TECHNOLOGY CONSULTING

B5

EY Building a better working world

eni

exprivia | ITALTEL

IBM

KPMG

LEONARDO

NTT DATA Trusted Global Innovator

NUMERA SISTEMI E INFORMATICA S.p.A.

Telsy

SPONSOR GOLD

bip.

cisco

MONTE DEI PASCHI DI SIENA BANCA DAL 1472

negg

NOVANEXT connecting the future

pwc

SPONSOR SILVER

DiGi ONE the leading digital company

ICT CYBER CONSULTING

Security Protocols

Alessandro Armando

Università di Genova

alessandro.armando@unige.it

CYBER CHALLENGE
CyberChallenge.IT

consorzio interuniversitario nazionale per l'informatica

Cyber Security National Lab

Outline

1 Motivation

2 Basic notions

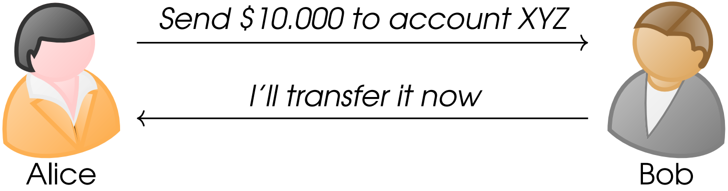
3 Needham-Schroeder Public Key Authentication Protocol

4 Needham-Schroeder Shared-Key Protocol

5 Kerberos

Motivation

Example: Securing an e-banking application.



Send $10.000 to account XYZ

I'll transfer it now

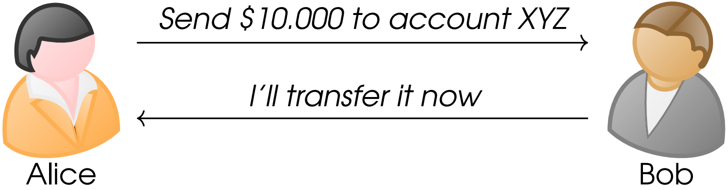
Alice

Bob

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

Motivation

Example: Securing an e-banking application.



Send $10.000 to account XYZ

I'll transfer it now

Alice

Bob

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

Motivation – top down (II)

Other examples:

- Securing a sensor network.
- A micropayment scheme for a parking company.
- An access control system for area-wide ski lifts.

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.
- A micropayment scheme for a parking company.
- An access control system for area-wide ski lifts.

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.
- A micropayment scheme for a parking company.
- An access control system for area-wide ski lifts.

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.
- A micropayment scheme for a parking company.
- An access control system for area-wide ski lifts.

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.
- A micropayment scheme for a parking company.
- An access control system for area-wide ski lifts.

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.
- A micropayment scheme for a parking company.
- An access control system for area-wide ski lifts.

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

1 Motivation

2 Basic notions

3 Needham-Schroeder Public Key Authentication Protocol

4 Needham-Schroeder Shared-Key Protocol

5 Kerberos

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.

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.

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.

Messages

- Message constructors are:
  Names: A, B or *Alice*, *Bob*, ...
  Keys: K and inverse keys K^{-1}
  Encryption: $\{M\}_K$; Example: encryption with A's public key: $\{M\}_{K_A}$
  Signing: $\{M\}_{K^{-1}}$; Example: signing with A's private key: $\{M\}_{K_A^{-1}}$
  Symmetric keys: $\{M\}_{K_{AB}}$
  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: $\{A, T_A, K_{AB}\}_{K_B}$

Messages

- Message constructors are:

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

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

  Encryption: $\{M\}_K$; Example: encryption with A's public key: $\{M\}_{K_A}$

  Signing: $\{M\}_{K^{-1}}$; Example: signing with A's private key: $\{M\}_{K_A^{-1}}$

  Symmetric keys: $\{M\}_{K_{AB}}$

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

Messages

- Message constructors are:

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

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

  Encryption: $\{M\}_K$; Example: encryption with A's public key: $\{M\}_{K_A}$

  Signing: $\{M\}_{K^{-1}}$; Example: signing with A's private key: $\{M\}_{K_A^{-1}}$

  Symmetric keys: $\{M\}_{K_{AB}}$

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

Messages

- Message constructors are:

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

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

  Encryption: $\{M\}_K$; Example: encryption with A's public key: $\{M\}_{K_A}$

  Signing: $\{M\}_{K^{-1}}$; Example: signing with A's private key: $\{M\}_{K_A^{-1}}$

  Symmetric keys: $\{M\}_{K_{AB}}$

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

Messages

- Message constructors are:

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

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

  Encryption: $\{M\}_K$; Example: encryption with A's public key: $\{M\}_{K_A}$

  Signing: $\{M\}_{K^{-1}}$; Example: signing with A's private key: $\{M\}_{K_A^{-1}}$

  Symmetric keys: $\{M\}_{K_{AB}}$

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

Messages

- Message constructors are:

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

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

  Encryption: $\{M\}_K$; Example: encryption with A's public key: $\{M\}_{K_A}$

  Signing: $\{M\}_{K^{-1}}$; Example: signing with A's private key: $\{M\}_{K_A^{-1}}$

  Symmetric keys: $\{M\}_{K_{AB}}$

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

Messages

- Message constructors are:

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

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

  Encryption: $\{M\}_K$; Example: encryption with A's public key: $\{M\}_{K_A}$

  Signing: $\{M\}_{K^{-1}}$; Example: signing with A's private key: $\{M\}_{K_A^{-1}}$

  Symmetric keys: $\{M\}_{K_{AB}}$

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

Messages

- Message constructors are:

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

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

  Encryption: $\{M\}_K$; Example: encryption with A's public key: $\{M\}_{K_A}$

  Signing: $\{M\}_{K^{-1}}$; Example: signing with A's private key: $\{M\}_{K_A^{-1}}$

  Symmetric keys: $\{M\}_{K_{AB}}$

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

Messages

- Message constructors are:

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

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

  Encryption: $\{M\}_K$; Example: encryption with A's public key: $\{M\}_{K_A}$

  Signing: $\{M\}_{K^{-1}}$; Example: signing with A's private key: $\{M\}_{K_A^{-1}}$

  Symmetric keys: $\{M\}_{K_{AB}}$

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

Messages

- Message constructors are:

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

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

  Encryption: $\{M\}_K$; Example: encryption with A's public key: $\{M\}_{K_A}$

  Signing: $\{M\}_{K^{-1}}$; Example: signing with A's private key: $\{M\}_{K_A^{-1}}$

  Symmetric keys: $\{M\}_{K_{AB}}$

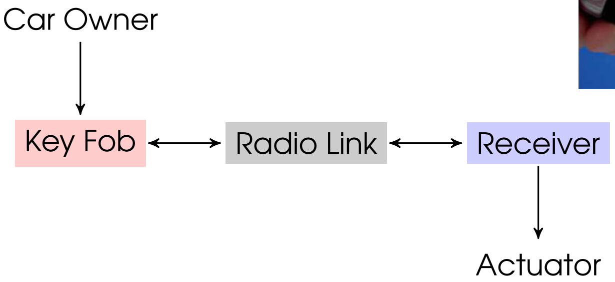
  Nonces: N_A, fresh data items used for challenge/response.

  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



Car Owner

Key Fob

Radio Link

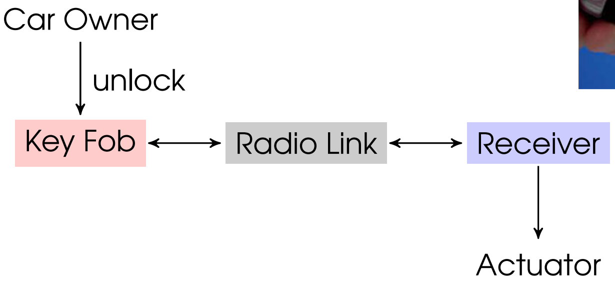
Receiver

Actuator

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



Car Owner

unlock

Key Fob

Radio Link

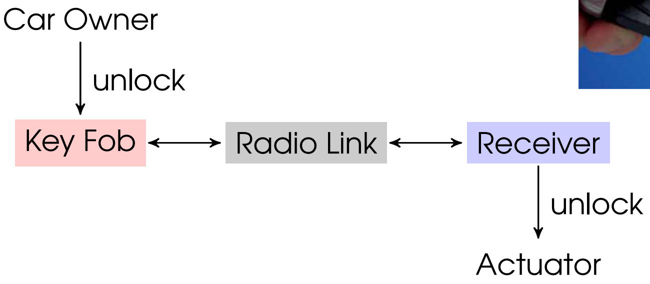
Receiver

Actuator

Security Goal (1st attempt)

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

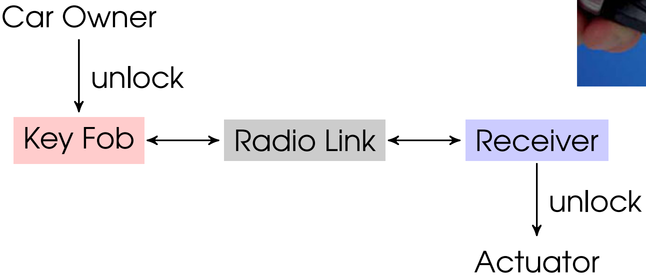
Example: Remote Keyless System



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



Car Owner

unlock

Key Fob ⟷ Radio Link ⟷ Receiver

unlock

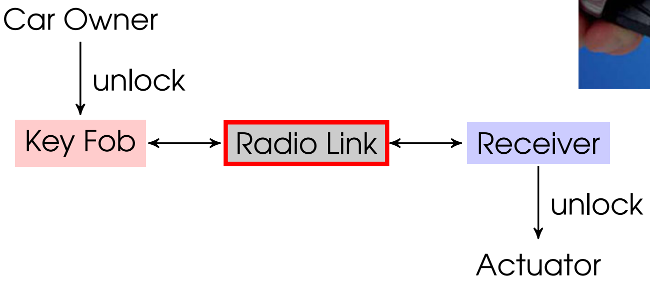
Actuator

✓

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



Car Owner

unlock

Key Fob ⟷ Radio Link ⟷ Receiver

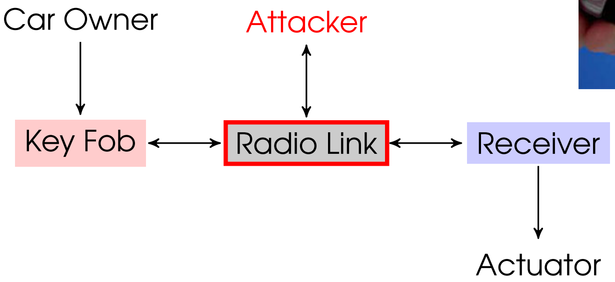
unlock

Actuator

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



Car Owner

Attacker

Key Fob

Radio Link

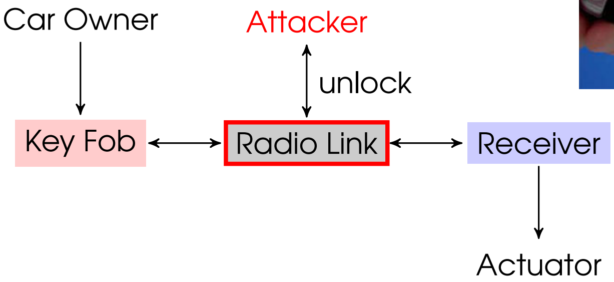
Receiver

Actuator

Security Goal (1st attempt)

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

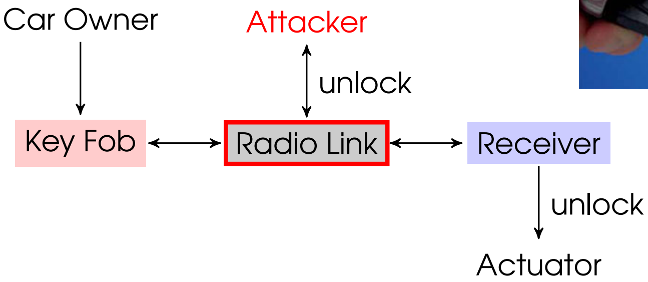
Example: Remote Keyless System



Security Goal (1st attempt)

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

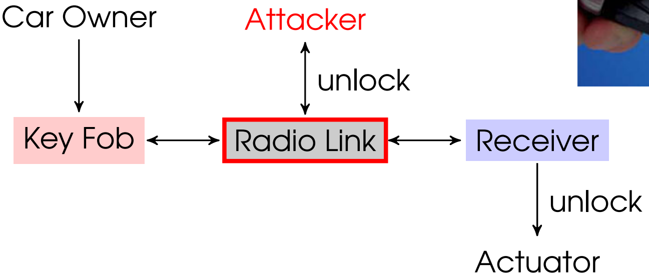
Example: Remote Keyless System



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

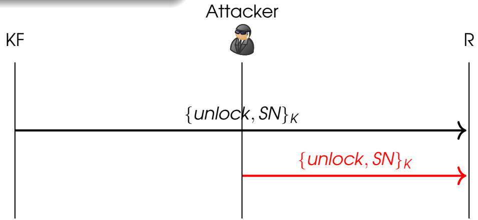
Remote Keyless System Protocol (2nd attempt)

Idea: protect secrecy of SN

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

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

- 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...



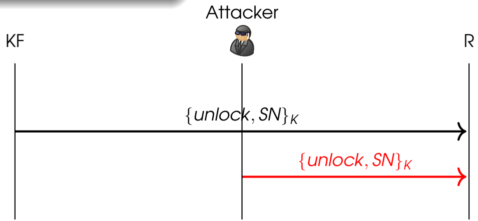
Remote Keyless System Protocol (2nd attempt)

Idea: protect secrecy of SN

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

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

- 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...



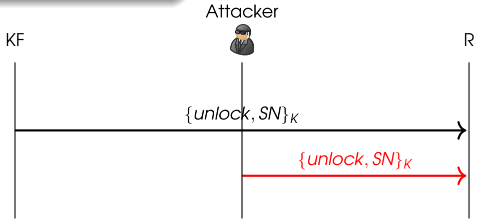
Remote Keyless System Protocol (2nd attempt)

Idea: protect secrecy of SN

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

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

- 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...



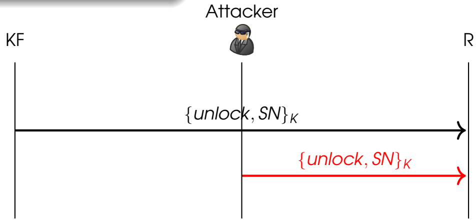
Remote Keyless System Protocol (2nd attempt)

Idea: protect secrecy of SN

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

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

- 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...

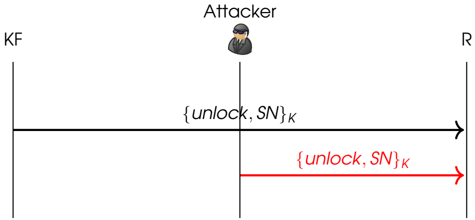


Security Goal: Freshness Requirement

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)

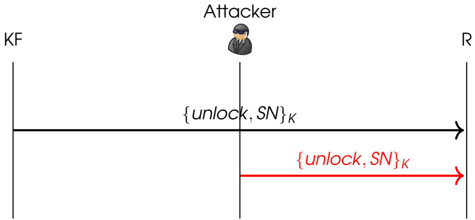
Receiver sends unlock command to Actuator *only if* Car Owner *recently* pressed unlock button on Key Fob.

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 (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.

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.

Alice & Bob Notation

- Fundamental events are communication between principals:

$$
\begin{array}{ll}
1. & A \rightarrow B : \{A, T_A, K\}_{K_B} \\
2. & B \rightarrow A : \{B, A\}_K
\end{array}
$$

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

Alice & Bob Notation

- Fundamental events are communication between principals:

1. $A \rightarrow B : \{A, T_A, K\}_{K_B}$
2. $B \rightarrow A : \{B, A\}_K$

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

Alice & Bob Notation

- Fundamental events are communication between principals:

$$
\begin{aligned}
&1.\ A \rightarrow B : \{A, T_A, K\}_{K_B}\\
&2.\ B \rightarrow A : \{B, A\}_K
\end{aligned}
$$

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

Alice & Bob Notation

- Fundamental events are communication between principals:

$$
\begin{aligned}
&1.\quad A \rightarrow B : \{A, T_A, K\}_{K_B}\\
&2.\quad B \rightarrow A : \{B, A\}_K
\end{aligned}
$$

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

Alice & Bob Notation

- Fundamental events are communication between principals:

1. $A \rightarrow B : \{A, T_A, K\}_{K_B}$
2. $B \rightarrow A : \{B, A\}_K$

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

Alice & Bob Notation

- Fundamental events are communication between principals:

$$\begin{array}{ll} 1. & A \rightarrow B : \{A, T_A, K\}_{K_B} \\ 2. & B \rightarrow A : \{B, A\}_K \end{array}$$

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

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
- The attacker controls the network, but cannot break crypto.

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 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
- The attacker controls the network, but cannot break crypto.

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 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
- The attacker controls the network, but cannot break crypto.

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 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
- The attacker controls the network, but cannot break crypto.

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 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
- The attacker controls the network, but cannot break crypto.

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 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
- The attacker controls the network, but cannot break crypto.

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 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
- The attacker controls the network, but cannot break crypto.

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 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
- The attacker controls the network, but cannot break crypto.

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 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
- The attacker controls the network, but cannot break crypto.

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!

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!

Assumptions: Attacker

How do we model the attacker? Possibilities:

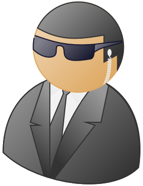
- He knows the protocol but cannot break crypto. (Standard)
- He is passive but overhears all communications.
- He is active and can intercept and generate messages.
- He might even be one of the principals running the protocol!

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

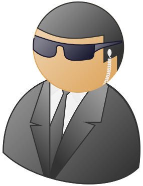


- The attacker is active. Namely:
  - He can intercept and read all messages.
  - He can decompose messages into their parts.
    But cryptography is secure: decryption requires inverse keys.
  - He can build new messages with the different constructors.
  - He can send messages at any time.
- Sometimes called the Dolev-Yao attacker model.
- Strongest possible assumptions about the attacker

⟹

correct protocols function in the largest range of environments.

Standard Attacker Model

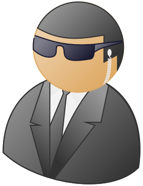


- The attacker is active. Namely:
  - He can intercept and read all messages.
  - He can decompose messages into their parts.
    But cryptography is secure: decryption requires inverse keys.
  - He can build new messages with the different constructors.
  - He can send messages at any time.
- Sometimes called the Dolev-Yao attacker model.
- Strongest possible assumptions about the attacker

⟹

correct protocols function in the largest range of environments.

Standard Attacker Model

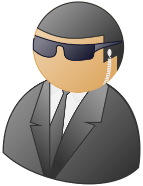


- The attacker is active. Namely:
  - He can intercept and read all messages.
  - He can decompose messages into their parts.
    But cryptography is secure: decryption requires inverse keys.
  - He can build new messages with the different constructors.
  - He can send messages at any time.
- Sometimes called the Dolev-Yao attacker model.
- Strongest possible assumptions about the attacker

⟹

correct protocols function in the largest range of environments.

Standard Attacker Model

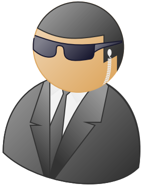


- The attacker is active. Namely:
  - He can intercept and read all messages.
  - He can decompose messages into their parts.
    But cryptography is secure: decryption requires inverse keys.
  - He can build new messages with the different constructors.
  - He can send messages at any time.
- Sometimes called the Dolev-Yao attacker model.
- Strongest possible assumptions about the attacker

⟹

correct protocols function in the largest range of environments.

Standard Attacker Model

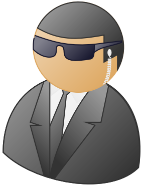


- The attacker is active. Namely:
  - He can intercept and read all messages.
  - He can decompose messages into their parts.
    But cryptography is secure: decryption requires inverse keys.
  - He can build new messages with the different constructors.
  - He can send messages at any time.
- Sometimes called the Dolev-Yao attacker model.
- Strongest possible assumptions about the attacker

⟹

correct protocols function in the largest range of environments.

Standard Attacker Model

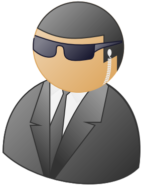


- The attacker is active. Namely:
  - He can intercept and read all messages.
  - He can decompose messages into their parts.
    But cryptography is secure: decryption requires inverse keys.
  - He can build new messages with the different constructors.
  - He can send messages at any time.
- Sometimes called the Dolev-Yao attacker model.
- Strongest possible assumptions about the attacker

⟹

correct protocols function in the largest range of environments.

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.

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.

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.

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

1 Motivation

2 Basic notions

3 Needham-Schroeder Public Key Authentication Protocol

4 Needham-Schroeder Shared-Key Protocol

5 Kerberos

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.

NSPK Protocol

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

- **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.

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.

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.

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.

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.

NSPK Protocol

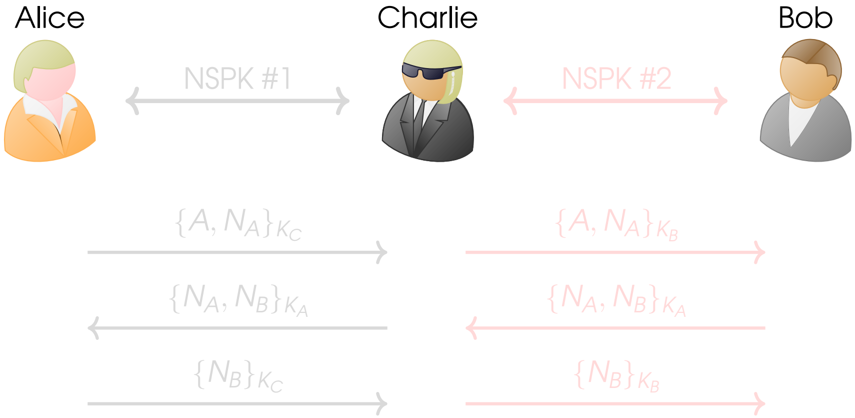
1. $A \rightarrow B : \{A, N_A\}_{K_B}$
2. $B \rightarrow A : \{N_A, N_B\}_{K_A}$
3. $A \rightarrow B : \{N_B\}_{K_B}$

- **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.

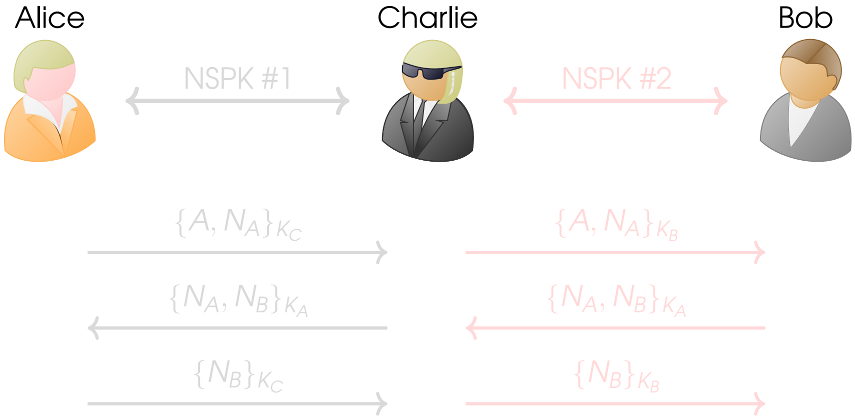
Attack on NSPK

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



Alice

Charlie

Bob

NSPK #1

NSPK #2

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

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

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

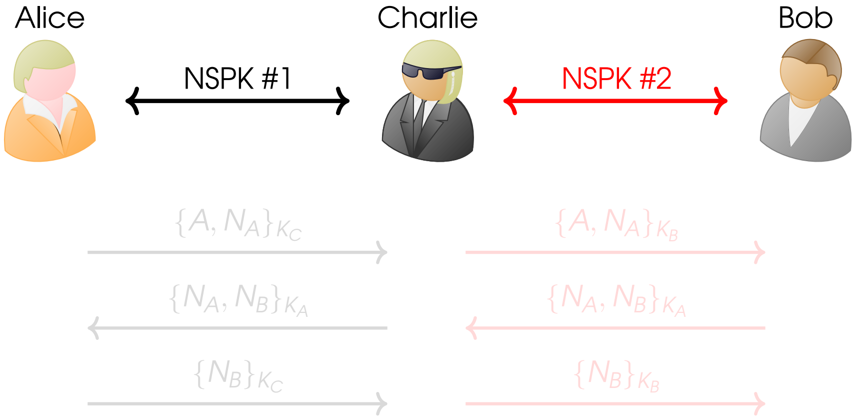
$\{N_A, N_B\}_{K_A}$

$\{N_B\}_{K_C}$

$\{N_B\}_{K_B}$

Attack on NSPK

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



Alice

Charlie

Bob

NSPK #1

NSPK #2

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

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

$\{N_B\}_{K_C}$

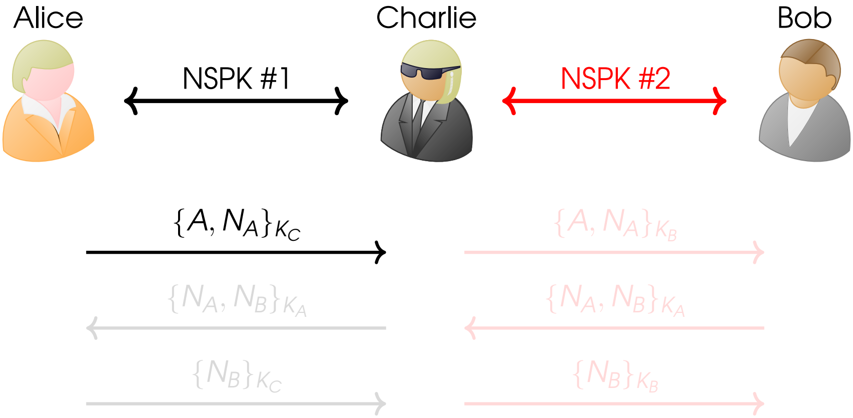
$\{A, N_A\}_{K_B}$

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

$\{N_B\}_{K_B}$

Attack on NSPK

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



Alice

Charlie

Bob

NSPK #1

NSPK #2

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

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

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

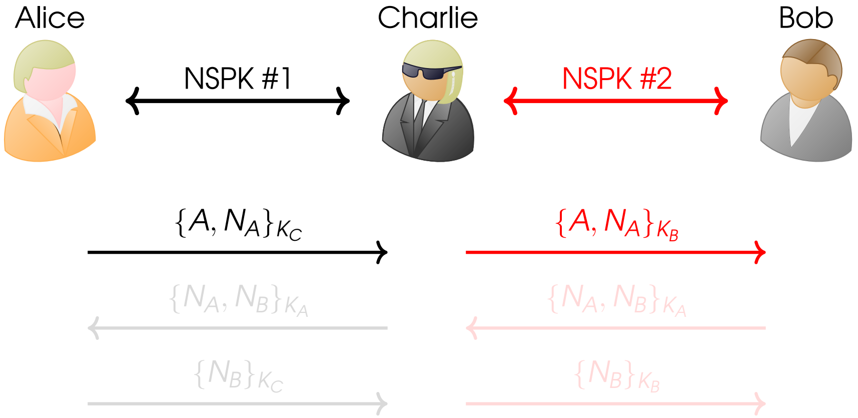
$\{N_A, N_B\}_{K_A}$

$\{N_B\}_{K_C}$

$\{N_B\}_{K_B}$

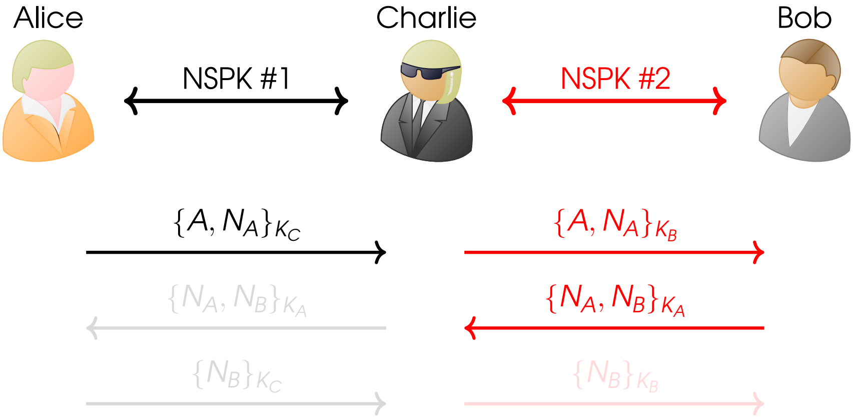
Attack on NSPK

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



Alice

Charlie

Bob

NSPK #1

NSPK #2

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

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

$\{N_B\}_{K_C}$

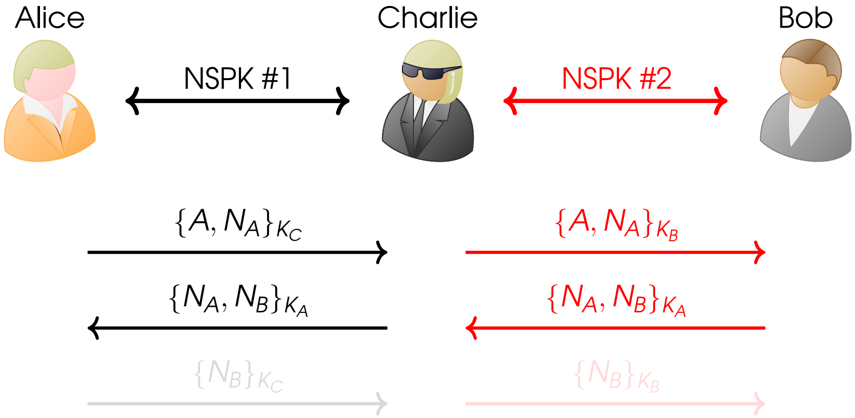
$\{A, N_A\}_{K_B}$

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

$\{N_B\}_{K_B}$

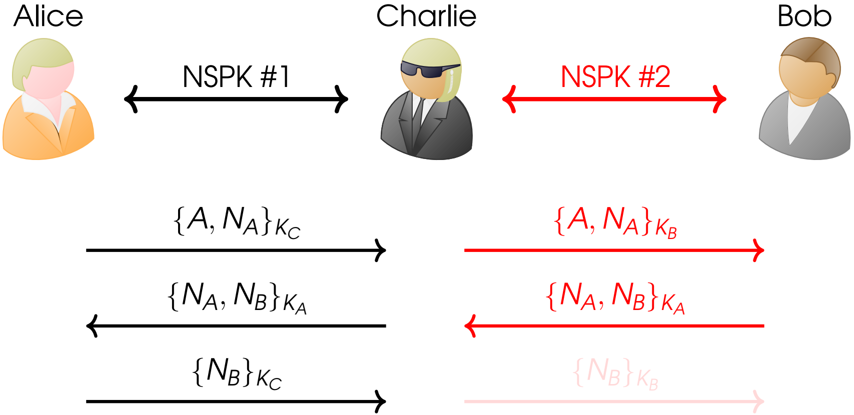
Attack on NSPK

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



Attack on NSPK

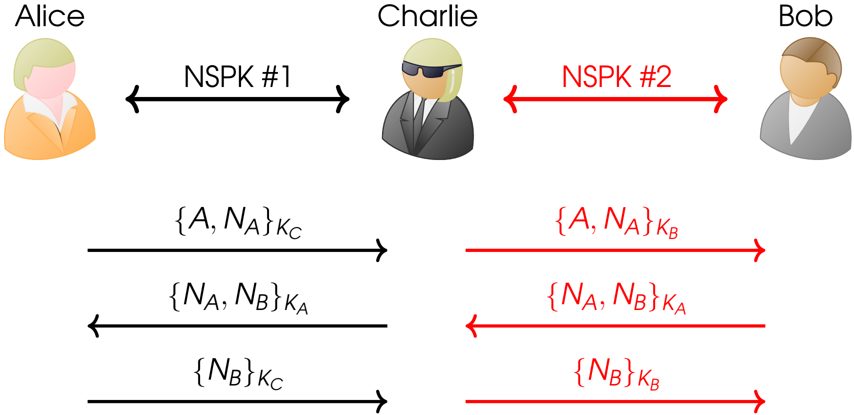
1. $A \rightarrow B : \{A, N_A\}_{K_B}$
2. $B \rightarrow A : \{N_A, N_B\}_{K_A}$
3. $A \rightarrow B : \{N_B\}_{K_B}$



B believes he is speaking with *A*!

Attack on NSPK

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



Alice

Charlie

Bob

NSPK #1

NSPK #2

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

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

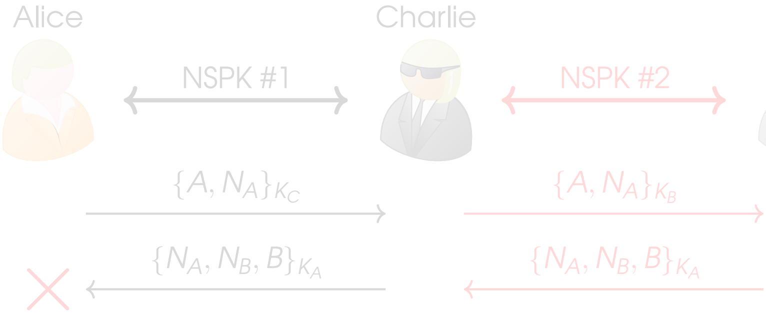
$\{N_B\}_{K_C}$ → $\{N_B\}_{K_B}$ →

B believes he is speaking with *A*!

NSPK Protocol with Lowe's Fix

How can we protect against the previous attack?

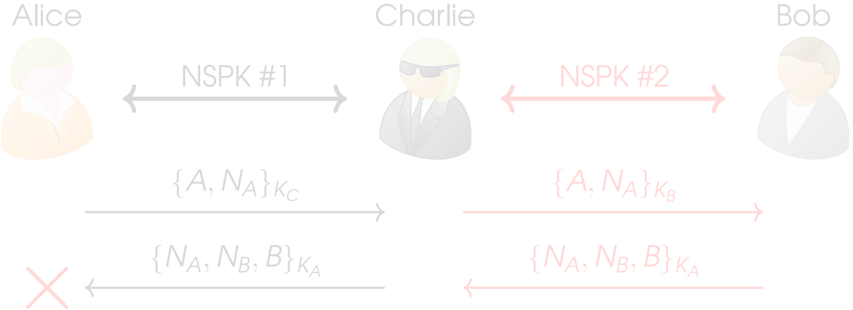
1. $A \rightarrow B : \{A, N_A\}_{K_B}$
2. $B \rightarrow A : \{N_A, N_B, B\}_{K_A}$
3. $A \rightarrow B : \{N_B\}_{K_B}$



NSPK Protocol with Lowe´s Fix

How can we protect against the previous attack?

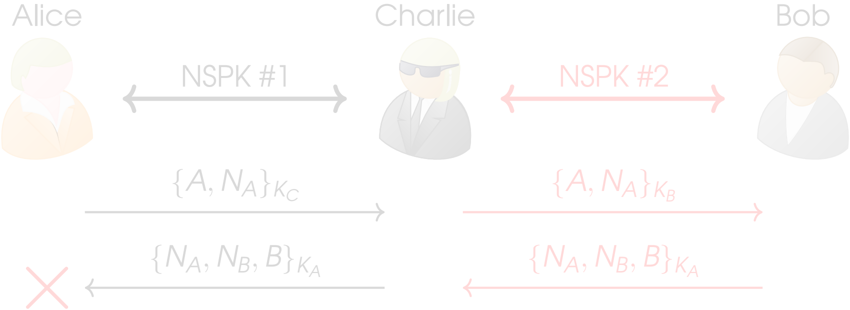
1. $A \rightarrow B : \{A, N_A\}_{K_B}$
2. $B \rightarrow A : \{N_A, N_B, B\}_{K_A}$
3. $A \rightarrow B : \{N_B\}_{K_B}$



NSPK Protocol with Lowe´s Fix

How can we protect against the previous attack?

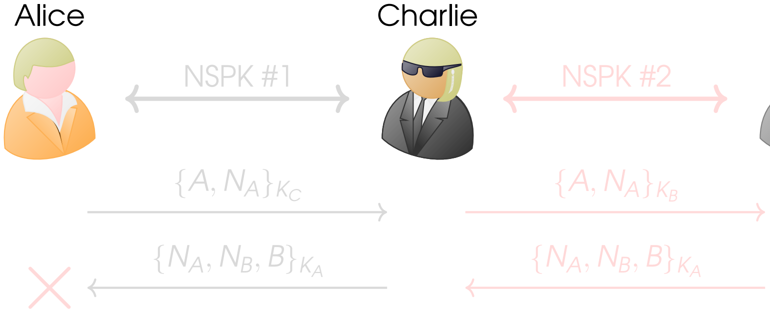
1. $A \rightarrow B : \{A, N_A\}_{K_B}$
2. $B \rightarrow A : \{N_A, N_B, B\}_{K_A}$
3. $A \rightarrow B : \{N_B\}_{K_B}$



NSPK Protocol with Lowe's Fix

How can we protect against the previous attack?

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



Alice

Charlie

Bob

NSPK #1

NSPK #2

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

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

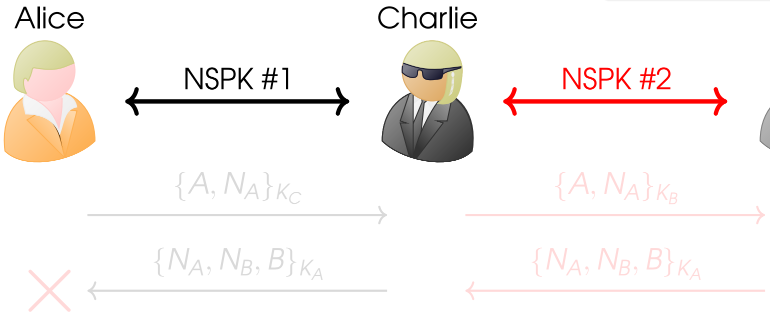
$\{N_A, N_B, B\}_{K_A}$

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

NSPK Protocol with Lowe´s Fix

How can we protect against the previous attack?

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



Alice

Charlie

Bob

NSPK #1

NSPK #2

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

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

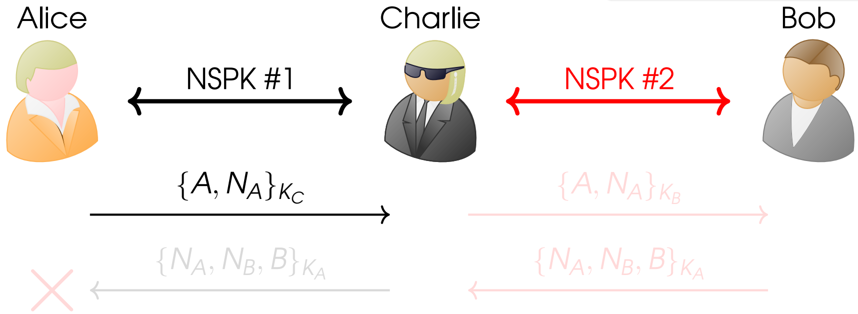
$\{N_A, N_B, B\}_{K_A}$

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

NSPK Protocol with Lowe´s Fix

How can we protect against the previous attack?

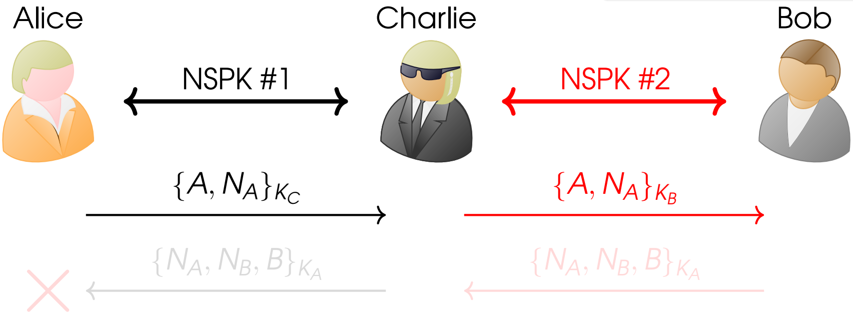
1. $A \rightarrow B : \{A, N_A\}_{K_B}$
2. $B \rightarrow A : \{N_A, N_B, B\}_{K_A}$
3. $A \rightarrow B : \{N_B\}_{K_B}$



NSPK Protocol with Lowe's Fix

How can we protect against the previous attack?

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



Alice

Charlie

Bob

NSPK #1

NSPK #2

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

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

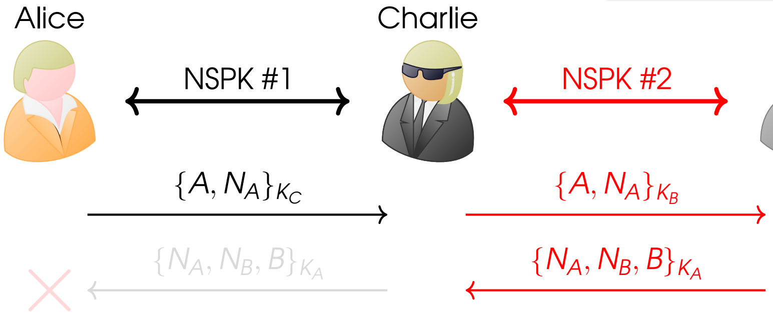
$\{N_A, N_B, B\}_{K_A}$

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

NSPK Protocol with Lowe´s Fix

How can we protect against the previous attack?

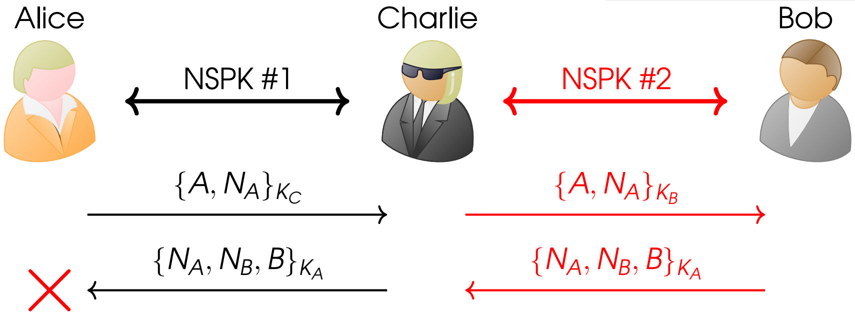
1. $A \rightarrow B : \{A, N_A\}_{K_B}$
2. $B \rightarrow A : \{N_A, N_B, B\}_{K_A}$
3. $A \rightarrow B : \{N_B\}_{K_B}$



NSPK Protocol with Lowe´s Fix

How can we protect against the previous attack?

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



Outline

1 Motivation

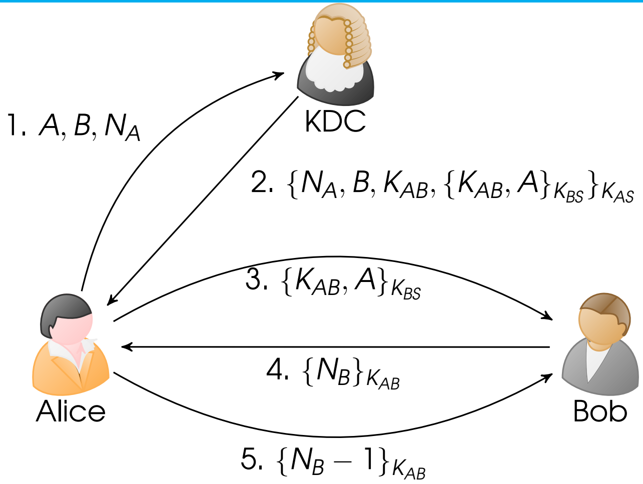
2 Basic notions

3 Needham-Schroeder Public Key Authentication Protocol

4 **Needham-Schroeder Shared-Key Protocol**

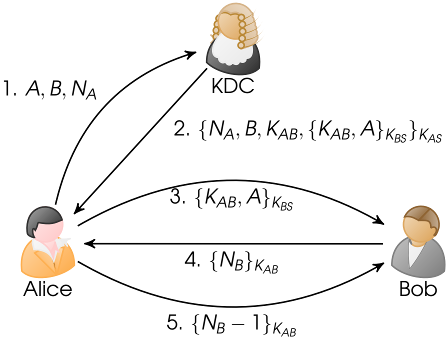
5 Kerberos

Needham-Schroeder Shared-Key Protocol


KDC
1. A, B, N_A
2. $\{N_A, B, K_{AB}, \{K_{AB}, A\}_{K_{BS}}\}_{K_{AS}}$
3. $\{K_{AB}, A\}_{K_{BS}}$
4. $\{N_B\}_{K_{AB}}$
5. $\{N_B - 1\}_{K_{AB}}$
Alice
Bob

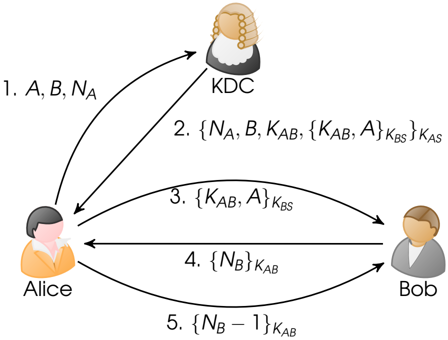
Security Goal: Authenticated key exchange.

Weakness of NSSK


KDC
Alice
Bob
1. A, B, N_A
2. $\{N_A, B, K_{AB}, \{K_{AB}, A\}_{K_{BS}}\}_{K_{AS}}$
3. $\{K_{AB}, A\}_{K_{BS}}$
4. $\{N_B\}_{K_{AB}}$
5. $\{N_B - 1\}_{K_{AB}}$

- If previous session key 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...

Weakness of NSSK


KDC
Alice
Bob
1. A, B, N_A
2. $\{N_A, B, K_{AB}, \{K_{AB}, A\}_{K_{BS}}\}_{K_{AS}}$
3. $\{K_{AB}, A\}_{K_{BS}}$
4. $\{N_B\}_{K_{AB}}$
5. $\{N_B - 1\}_{K_{AB}}$

- If previous session key 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

1 Motivation

2 Basic notions

3 Needham-Schroeder Public Key Authentication Protocol

4 Needham-Schroeder Shared-Key Protocol

5 Kerberos

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

- 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

- 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

- 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

- 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)

(Taken 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 (1/2)

(Taken 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 (1/2)

(Taken 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 (1/2)

(Taken 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 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 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 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 Authentication Protocol: Aims (2/2)

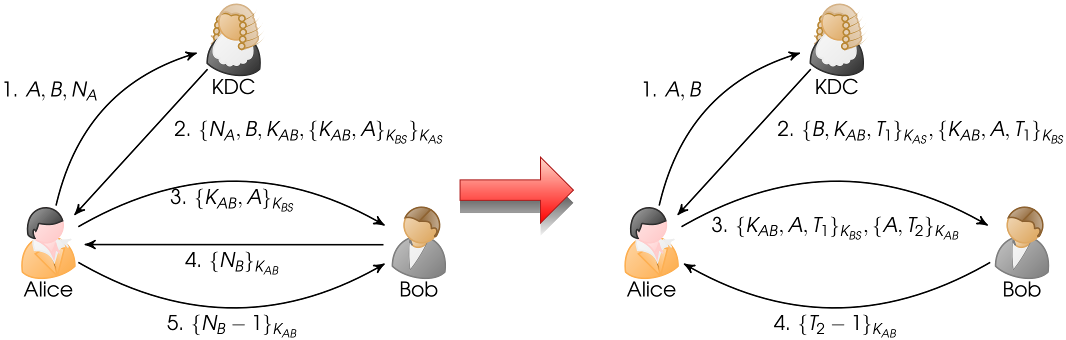
- Authentication information resides on the authentication server only. The application servers must not contain the authentication information for their users. This is essential for obtaining the following results:
  - The administrator can disable the account of any user by acting in a single location 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 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.

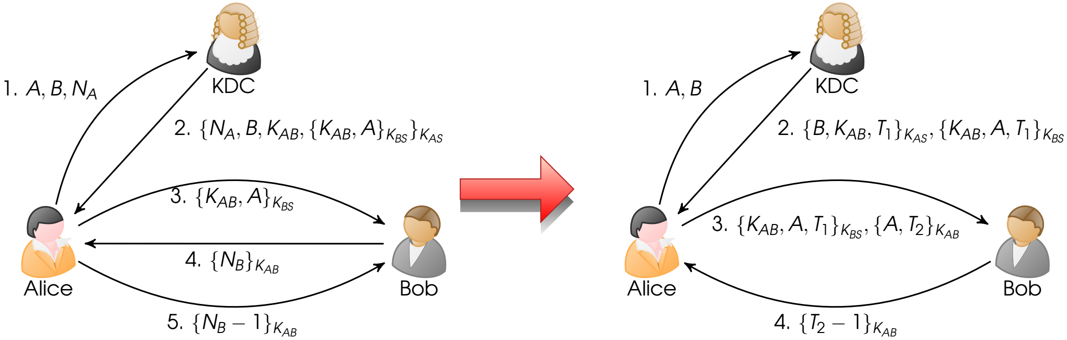
Kerberos Authentication Protocol

- Loosely based on the Needham-Schroeder Shared-Key Protocol:
  - Timestamps instead of nonces to assure freshness of session keys.
  - Removal of nested encryption.


1. A, B, N_A
KDC
2. $\{N_A, B, K_{AB}, \{K_{AB}, A\}_{K_{BS}}\}_{K_{AS}}$
3. $\{K_{AB}, A\}_{K_{BS}}$
Alice
4. $\{N_B\}_{K_{AB}}$
Bob
5. $\{N_B - 1\}_{K_{AB}}$
1. A, B
KDC
2. $\{B, K_{AB}, T_1\}_{K_{AS}}, \{K_{AB}, A, T_1\}_{K_{BS}}$
3. $\{K_{AB}, A, T_1\}_{K_{BS}}, \{A, T_2\}_{K_{AB}}$
Alice
Bob
4. $\{T_2 - 1\}_{K_{AB}}$

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.



KDC

1. A, B, N_A

2. $\{N_A, B, K_{AB}, \{K_{AB}, A\}_{K_{BS}}\}_{K_{AS}}$

3. $\{K_{AB}, A\}_{K_{BS}}$

4. $\{N_B\}_{K_{AB}}$

5. $\{N_B - 1\}_{K_{AB}}$

Alice

Bob

KDC

1. A, B

2. $\{B, K_{AB}, T_1\}_{K_{AS}}, \{K_{AB}, A, T_1\}_{K_{BS}}$

3. $\{K_{AB}, A, T_1\}_{K_{BS}}, \{A, T_2\}_{K_{AB}}$

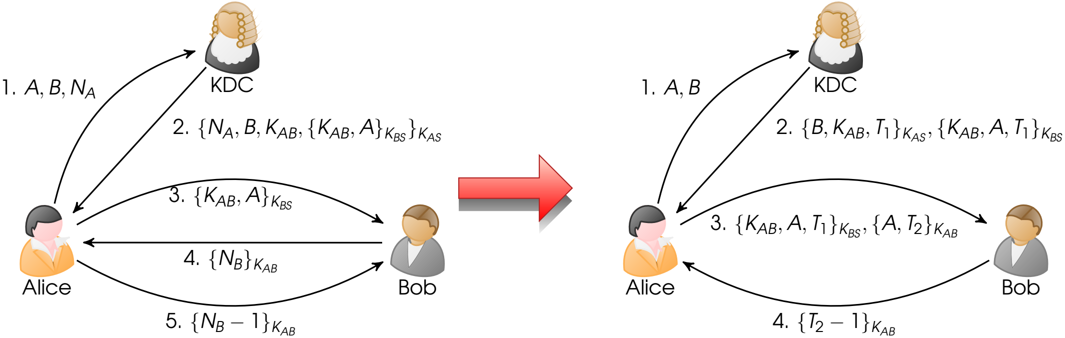
4. $\{T_2 - 1\}_{K_{AB}}$

Alice

Bob

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.



KDC

Alice

Bob

1. A, B, N_A

2. $\{N_A, B, K_{AB}, \{K_{AB}, A\}_{K_{BS}}\}_{K_{AS}}$

3. $\{K_{AB}, A\}_{K_{BS}}$

4. $\{N_B\}_{K_{AB}}$

5. $\{N_B - 1\}_{K_{AB}}$

KDC

Alice

Bob

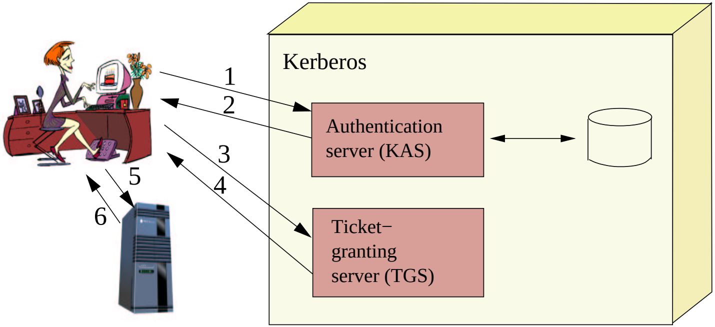
1. A, B

2. $\{B, K_{AB}, T_1\}_{K_{AS}}, \{K_{AB}, A, T_1\}_{K_{BS}}$

3. $\{K_{AB}, A, T_1\}_{K_{BS}}, \{A, T_2\}_{K_{AB}}$

4. $\{T_2 - 1\}_{K_{AB}}$

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. $KAS \rightarrow A \quad : \quad \{K_{A,TGS}, TGS, T_1\}_{K_{AS}}, \underbrace{\{A, TGS, K_{A,TGS}, T_1\}_{K_{KAS,TGS}}}_{AuthTicket}$

- *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.

Authentication phase

$$
\begin{array}{lrcl}
1. & A \rightarrow KAS & : & A, TGS \\
2. & KAS \rightarrow A & : & \{K_{A,TGS}, TGS, T_1\}_{K_{AS}}, \underbrace{\{A, TGS, K_{A,TGS}, T_1\}_{K_{KAS,TGS}}}_{AuthTicket}
\end{array}
$$

- *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.

Authentication phase

$$1. \quad A \rightarrow KAS \quad : \quad A, TGS$$

$$2. \quad KAS \rightarrow A \quad : \quad \{K_{A,TGS}, TGS, T_1\}_{K_{AS}}, \underbrace{\{A, TGS, K_{A,TGS}, T_1\}_{K_{KAS,TGS}}}_{AuthTicket}$$

- *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.

Authentication phase

$$
\begin{array}{lrcl}
1. & A \rightarrow KAS & : & A, TGS \\
2. & KAS \rightarrow A & : & \{K_{A,TGS}, TGS, T_1\}_{K_{AS}}, \underbrace{\{A, TGS, K_{A,TGS}, T_1\}_{K_{KAS,TGS}}}_{AuthTicket}
\end{array}
$$

- *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.

Authentication phase

$$\begin{array}{lll} 1. & A \to KAS & : \quad A, TGS \\ 2. & KAS \to A & : \quad \{K_{A,TGS}, TGS, T_1\}_{K_{AS}}, \underbrace{\{A, TGS, K_{A,TGS}, T_1\}_{K_{KAS,TGS}}}_{AuthTicket} \end{array}$$

- *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.

Authentication phase

$$
\begin{array}{lrcl}
1. & A \rightarrow KAS & : & A, TGS \\
2. & KAS \rightarrow A & : & \{K_{A,TGS}, TGS, T_1\}_{K_{AS}}, \underbrace{\{A, TGS, K_{A,TGS}, T_1\}_{K_{KAS,TGS}}}_{AuthTicket}
\end{array}
$$

- *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

$$3.\ \mathrm{A} \rightarrow \mathrm{TGS}: \quad \underbrace{\{A,\ TGS,\ K_{A,TGS},\ T_1\}_{K_{KAS,TGS}}}_{\color{red}{AuthTicket}},\ \underbrace{\{A,\ T_2\}_{K_{A,TGS}}}_{\color{red}{authenticator}},\ \mathrm{B}$$

$$4.\ \mathrm{TGS} \rightarrow \mathrm{A}: \quad \{K_{AB}, B, T_3\}_{K_{A,TGS}},\ \underbrace{\{A,\ B,\ K_{AB},\ T_3\}_{K_{BS}}}_{\color{red}{ServTicket}}$$

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. $A \to \mathrm{TGS} : \quad \underbrace{\{A,\ TGS,\ K_{A,TGS},\ T_1\}_{K_{KAS,TGS}}}_{AuthTicket},\ \underbrace{\{A,\ T_2\}_{K_{A,TGS}}}_{authenticator},\ \mathrm{B}$

4. $\mathrm{TGS} \to A : \quad \{K_{AB}, B, T_3\}_{K_{A,TGS}},\ \underbrace{\{A,\ B,\ K_{AB},\ T_3\}_{K_{BS}}}_{ServTicket}$

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

$$3.\ \mathrm{A} \rightarrow \mathrm{TGS}:\quad \underbrace{\{A,\ TGS,\ K_{A,TGS},\ T_1\}_{K_{KAS,TGS}}}_{\color{red}{AuthTicket}},\ \underbrace{\{A,\ T_2\}_{K_{A,TGS}}}_{\color{red}{authenticator}},\ \mathrm{B}$$

$$4.\ \mathrm{TGS} \rightarrow \mathrm{A}:\quad \{K_{AB}, B, T_3\}_{K_{A,TGS}},\ \underbrace{\{A,\ B,\ K_{AB},\ T_3\}_{K_{BS}}}_{\color{red}{ServTicket}}$$

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

$$3.\ A \rightarrow TGS:\quad \underbrace{\{A,\ TGS,\ K_{A,TGS},\ T_1\}_{K_{KAS,TGS}}}_{\textit{AuthTicket}},\ \underbrace{\{A,\ T_2\}_{K_{A,TGS}}}_{\textit{authenticator}},\ B$$

$$4.\ TGS \rightarrow A:\quad \{K_{AB}, B, T_3\}_{K_{A,TGS}},\ \underbrace{\{A,\ B,\ K_{AB},\ T_3\}_{K_{BS}}}_{\textit{ServTicket}}$$

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

$$3.\ \mathrm{A} \rightarrow \mathrm{TGS}:\quad \underbrace{\{A,\ TGS,\ K_{A,TGS},\ T_1\}_{K_{KAS,TGS}}}_{\color{red}{AuthTicket}},\ \underbrace{\{A,\ T_2\}_{K_{A,TGS}}}_{\color{red}{authenticator}},\ \mathrm{B}$$

$$4.\ \mathrm{TGS} \rightarrow \mathrm{A}:\quad \{K_{AB}, B, T_3\}_{K_{A,TGS}},\ \underbrace{\{A,\ B,\ K_{AB},\ T_3\}_{K_{BS}}}_{\color{red}{ServTicket}}$$

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

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

$$6.\ B \rightarrow A: \quad \{\mathcal{T}_4 + 1\}_{K_{AB}}$$

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.

Service phase

$$5.\ A \to B: \quad \underbrace{\{A,\ B,\ K_{AB},\ T_3\}_{K_{BS}}}_{\text{ServTicket}},\ \underbrace{\{A, T_4\}_{K_{AB}}}_{\text{authenticator}}$$

$$6.\ B \to A: \quad \{T_4 + 1\}_{K_{AB}}$$

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.

CYBER CHALLENGE
CyberChallenge.IT

cini
Cybersecurity National Lab

SPONSOR PLATINUM

accenture security

aizoon AUSTRALIA EUROPE USA
TECHNOLOGY CONSULTING

B5

EY Building a better working world

eni

exprivia | ITALTEL

IBM

KPMG

LEONARDO

NTT Data
Trusted Global Innovator

NUMERA SISTEMI E INFORMATICA S.p.A.

Telsy

SPONSOR GOLD

bip.

cisco

MONTE DEI PASCHI DI SIENA
BANCA DAL 1472

negg

NOVANEXT connecting the future

pwc

SPONSOR SILVER

DiGi ONE the leading digital company

ICT CYBER CONSULTING