Secure communications: Cryptographic protocols

Markulf Kohlweiss & Myrto Arapinis
School of Informatics
University of Edinburgh

Jannuary 16, 2021

Context

Applications exchanging sensitive data over a public network:

- eBanking,
- eCommerce,
- eVoting,
- eHealth,
- Blockchains,
- Mobile phones,
- ...

Context

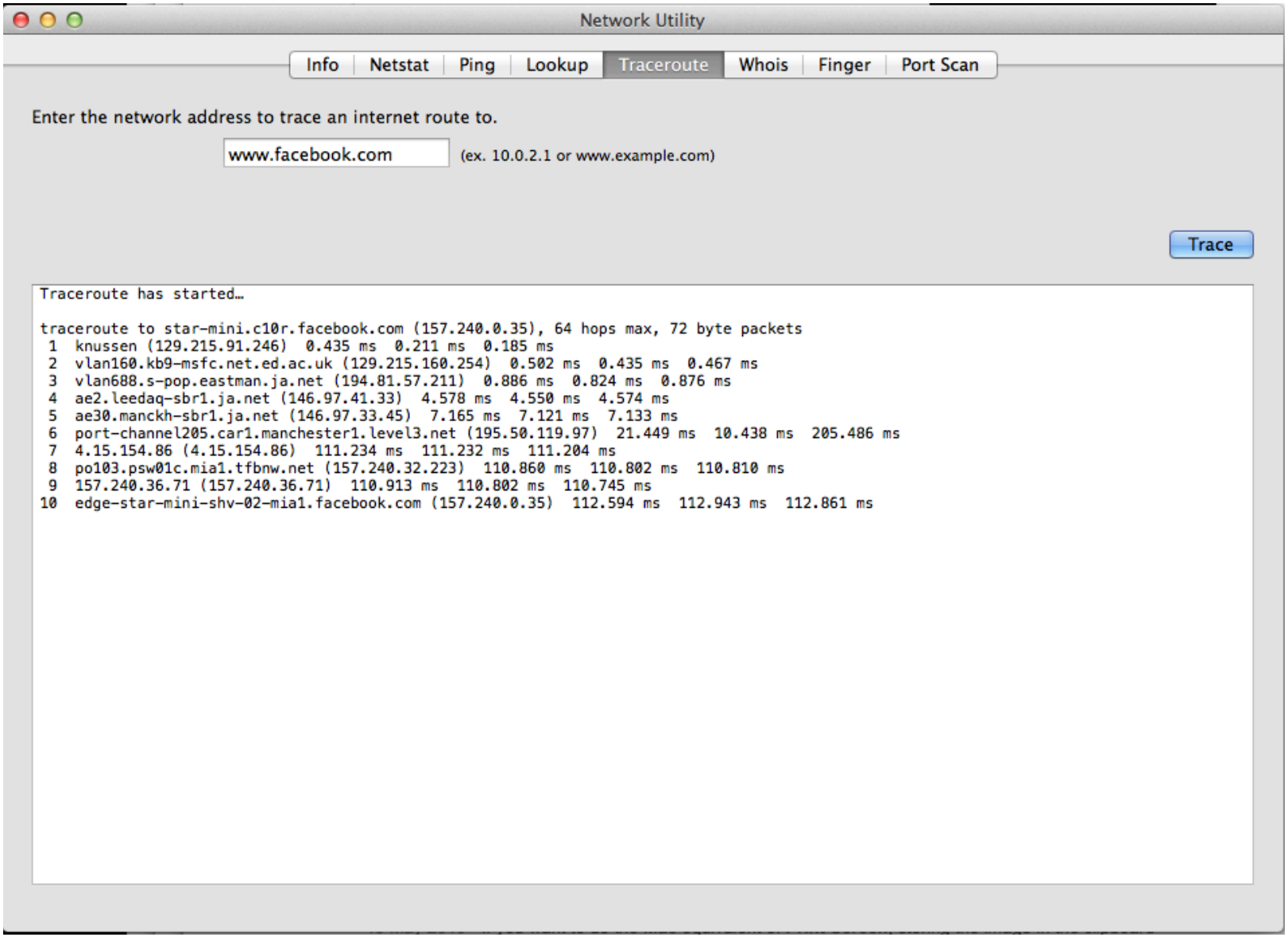
Applications exchanging sensitive data over a public network:

- eBanking,
- eCommerce,
- eVoting,
- eHealth,
- Blockchains,
- Mobile phones,
- ...

A malicious agent can:

- record, alter, delete, insert, redirect, reorder, and reuse past or current messages, and inject new messages
  $\longrightarrow$ the network is the attacker
- control dishonest participants

The attacker controls the network (1)



The attacker controls the network (2)

The Register®
Biting the hand that feeds IT

DATA CENTRE SOFTWARE NETWORKS SECURITY TRANSFORMATION DEVOPS BUSINESS HARDWARE

Networks

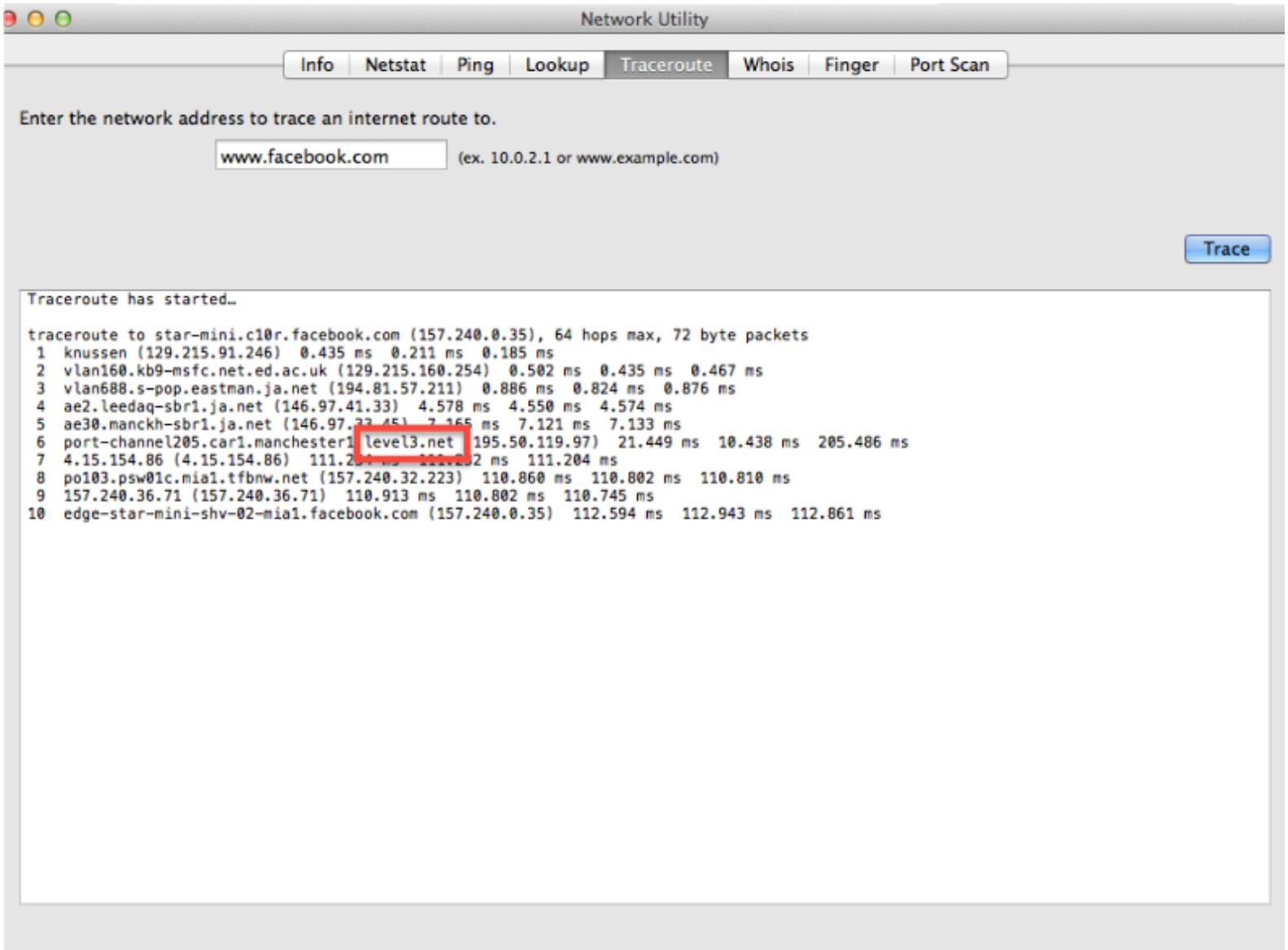
Verizon, BT, Vodafone, Level 3 'let NSA jack into Google, Yahoo! fiber'

Telcos cooperated with g-men in data slurp, claim sources

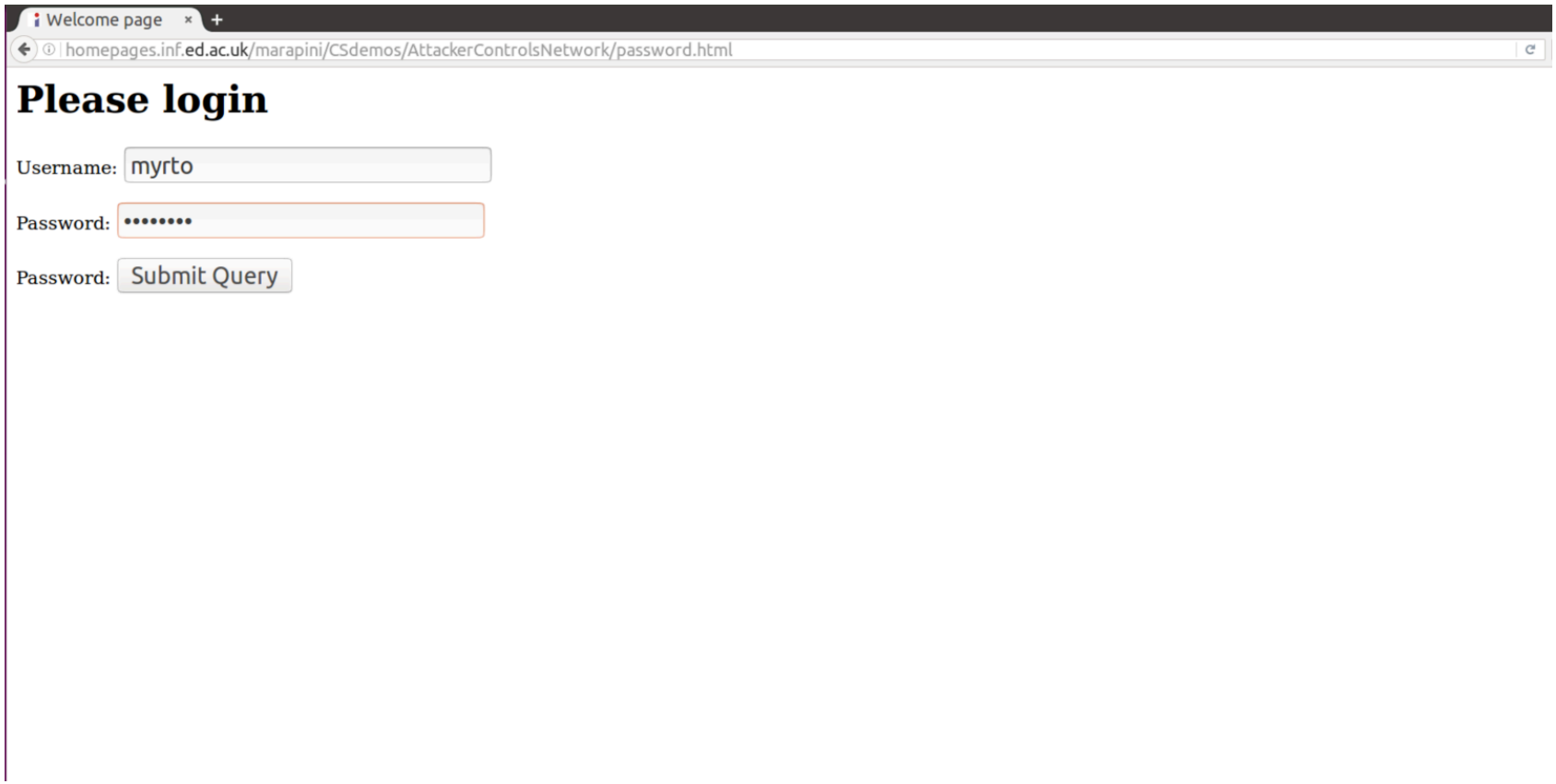
27 Nov 2013 at 02:19, Shaun Nichols 2

In October, NSA whistleblower Edward Snowden claimed Uncle Sam's spies tapped into the optic-fiber cables linking the data centers of Google and Yahoo!

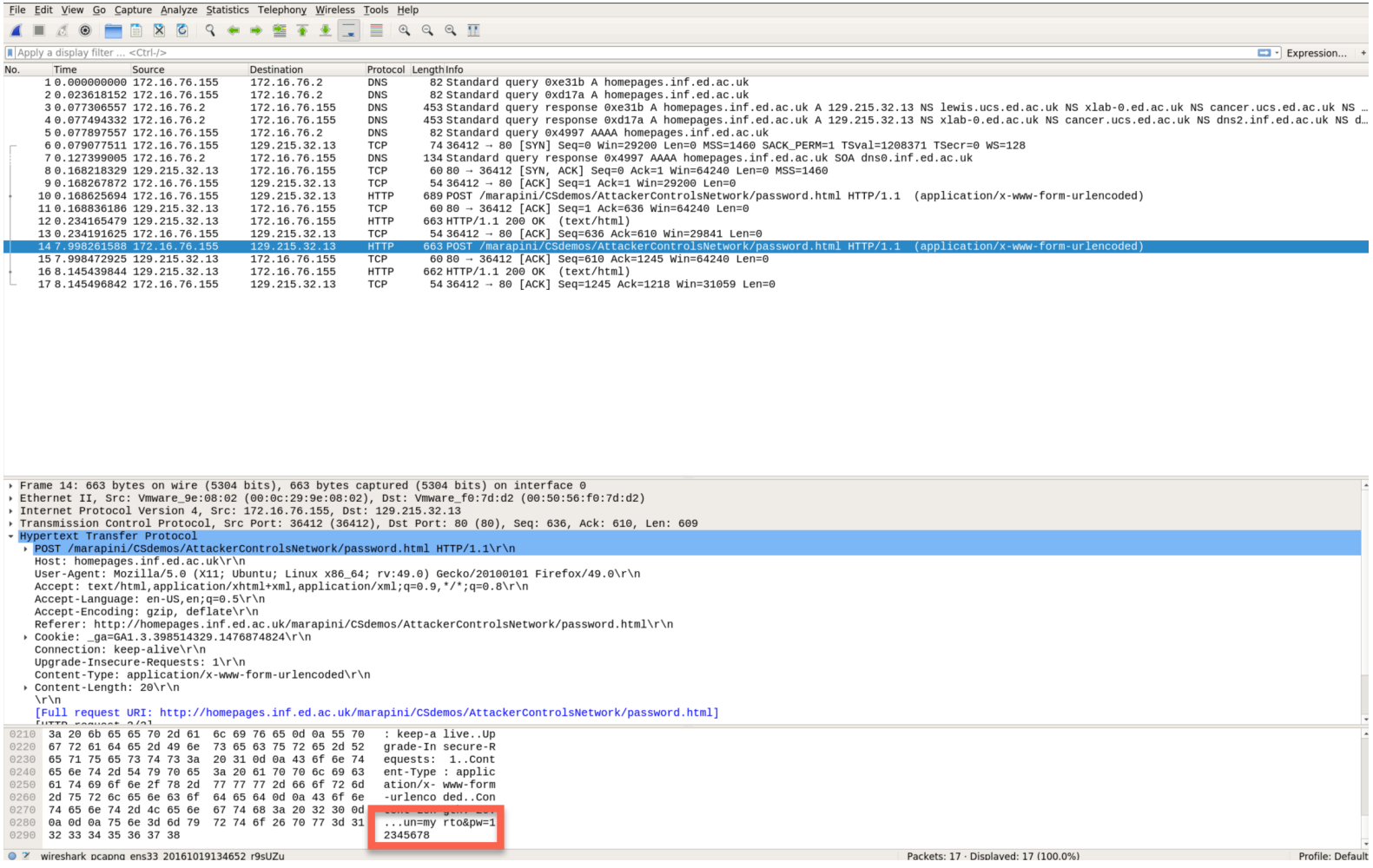
The attacker controls the network (3)



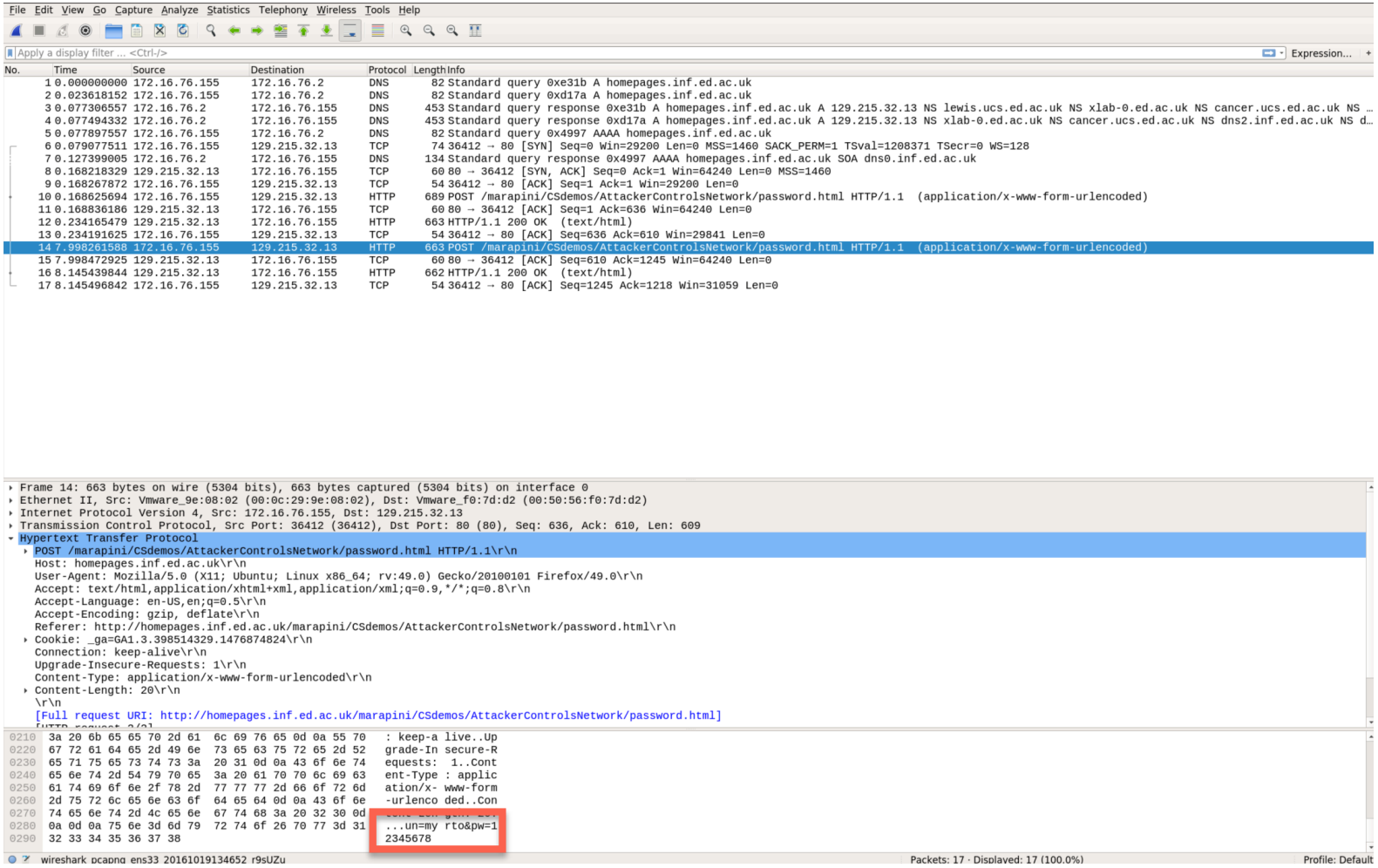
All messages can be intercepted by an attacker (1)



All messages can be intercepted by an attacker (2)



All messages can be intercepted by an attacker (2)



Man in the Middle (MitM) - An attacker can intercept packets, but also alter, forge new, and inject packets

More complex systems needed...

More complex systems needed...

$$\xrightarrow[m=MAC(K_M, E(K_E, \text{Transfer 100 € on Amazon's account}))]{e=E(K_E, \text{Transfer 100 € on Amazon's account})}$$

Royal Bank of Scotland

More complex systems needed...

$$\xrightarrow[m=MAC(K_M,E(K_E,\text{Transfer 100 € on Amazon's account}))]{e=E(K_E,\text{Transfer 100 € on Amazon's account})}$$

Royal Bank of Scotland

Replay attack

$\xrightarrow{(e,m)}$

$\xrightarrow{(e,m)}$

$\vdots$

$\xrightarrow{(e,m)}$

Royal Bank of Scotland

... to achieve more complex properties

- Confidentiality: Some information should never be revealed to unauthorised entities.
- Integrity: Data should not be altered in an unauthorised manner since the time it was created, transmitted or stored by an authorised source.
- Authentication: Ability to know with certainty the identity of an communicating entity.
- Anonymity: The identity of the author of an action (*e.g.* sending a message) should not be revealed.
- Unlinkability: An attacker should not be able to deduce whether different services are delivered to the same user
- Non-repudiation: The author of an action should not be able to deny having triggered this action.
- ...

Cryptographic protocols

Cryptographic protocols

Distributed programs relying on cryptographic primitives and whose goal is the establishment of "secure" communications.

Cryptographic protocols

Cryptographic protocols

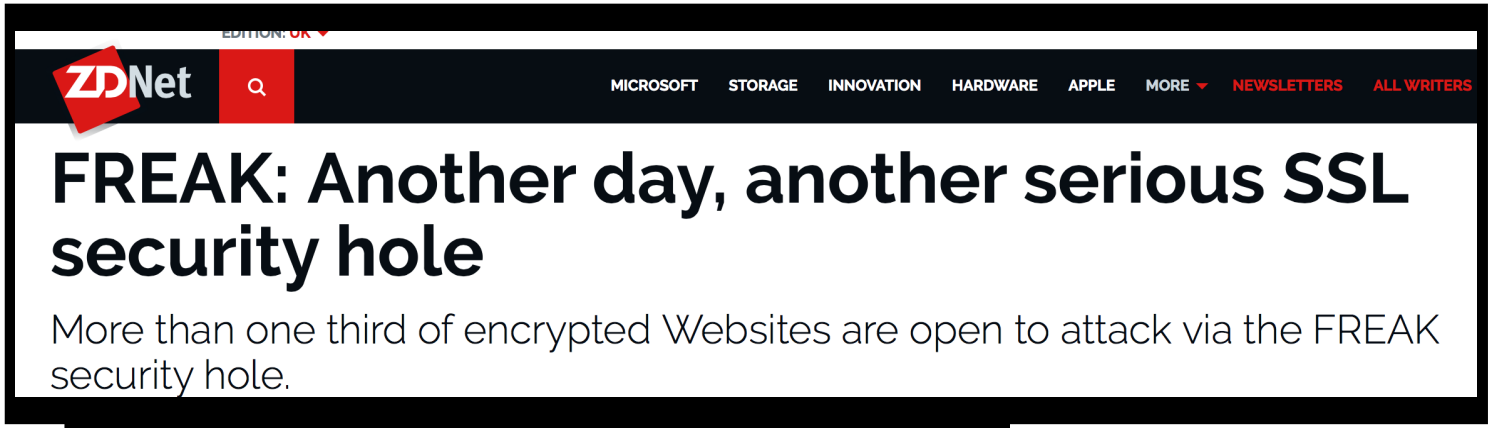
Distributed programs relying on cryptographic primitives and whose goal is the establishment of "secure" communications.

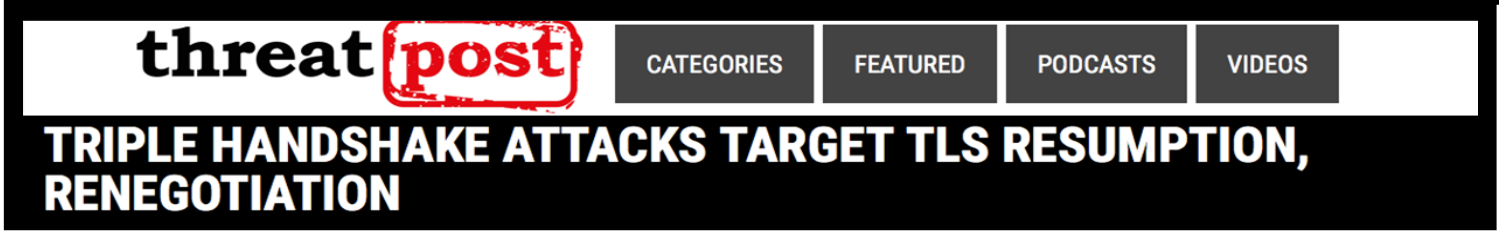
But!

Many exploitable errors are due not to design errors in the primitives, but to the way they are used, *i.e.* bad protocol design and buggy or not careful enough implementation

Numerous deployed protocols are flawed...

... and end up in the news :(





Logical attacks

Many of these attacks do not even break the crypto primitives!!

Example of a logical attack

Assume a commutative symmetric encryption scheme

$$\{\{m\}_{k_1}\}_{k_2} = \{\{m\}_{k_2}\}_{k_1}$$

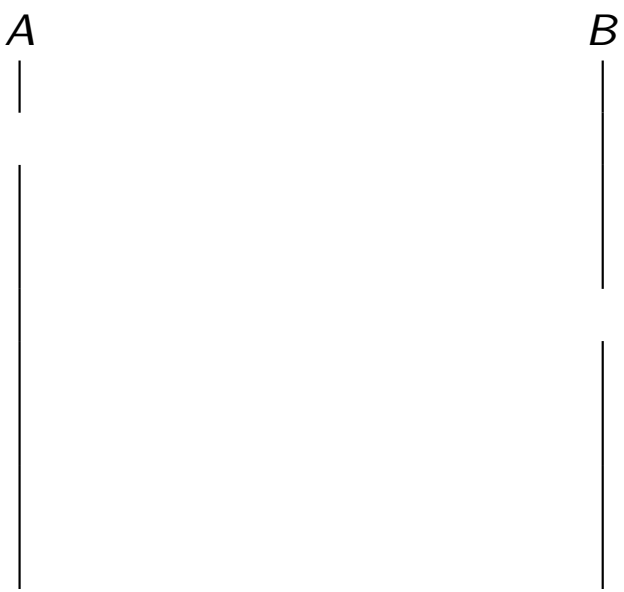
where $\{m\}_k$ denotes the encryption of message m under the key k
Example: stream ciphers

Example of a logical attack

Assume a commutative symmetric encryption scheme

$$\{\{m\}_{k_1}\}_{k_2} = \{\{m\}_{k_2}\}_{k_1}$$

where $\{m\}_k$ denotes the encryption of message m under the key k
Example: stream ciphers

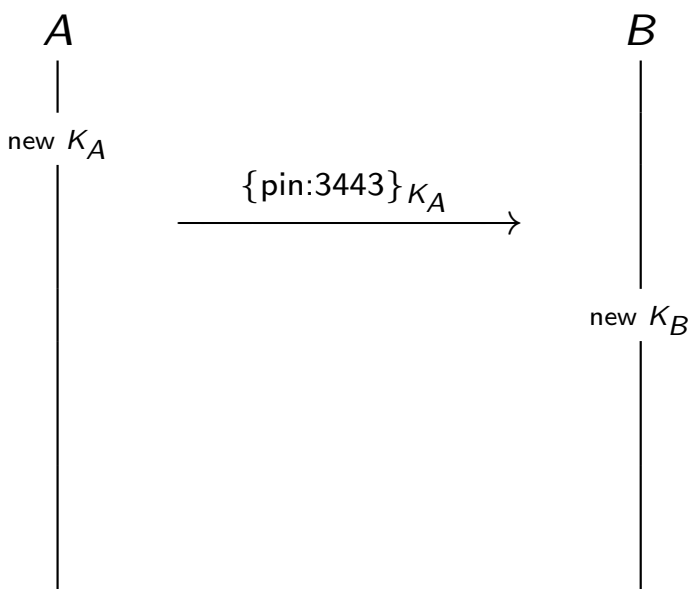


Example of a logical attack

Assume a commutative symmetric encryption scheme

$$\{\{m\}_{k_1}\}_{k_2} = \{\{m\}_{k_2}\}_{k_1}$$

where $\{m\}_k$ denotes the encryption of message m under the key k
Example: stream ciphers

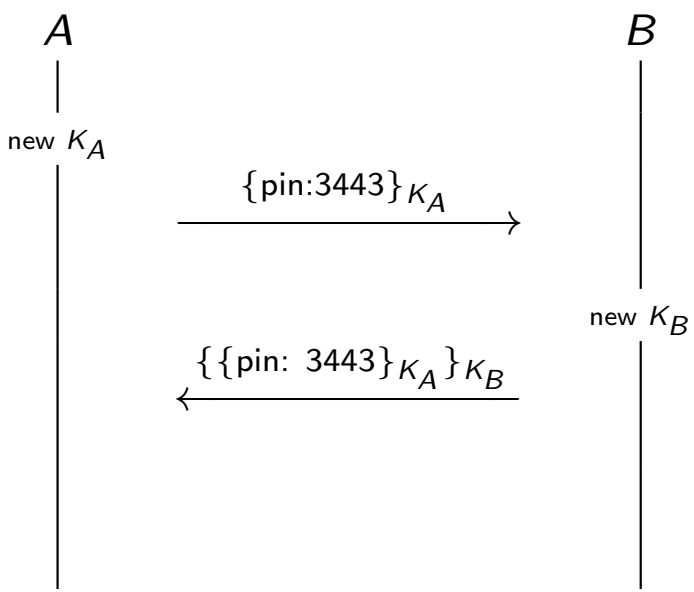


Example of a logical attack

Assume a commutative symmetric encryption scheme

$$\{\{m\}_{k_1}\}_{k_2} = \{\{m\}_{k_2}\}_{k_1}$$

where $\{m\}_k$ denotes the encryption of message m under the key k
Example: stream ciphers

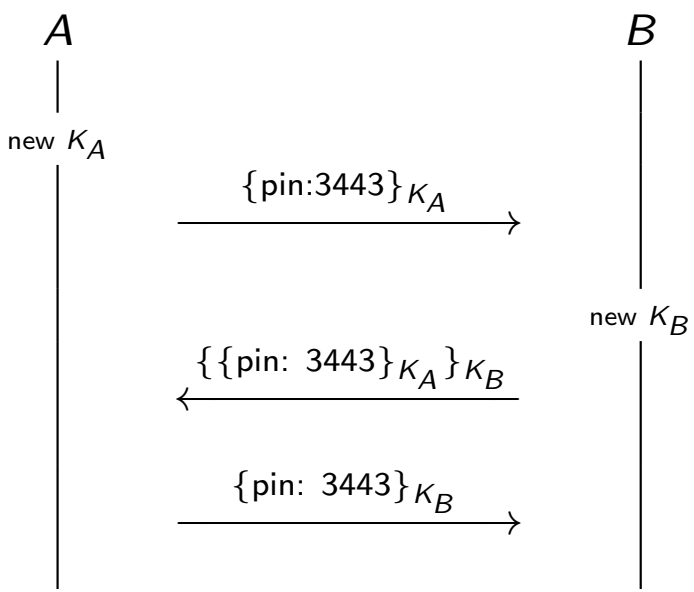


Example of a logical attack

Assume a commutative symmetric encryption scheme

$$\{\{m\}_{k_1}\}_{k_2} = \{\{m\}_{k_2}\}_{k_1}$$

where $\{m\}_k$ denotes the encryption of message m under the key k
Example: stream ciphers



since $\{\{\text{pin: 3443}\}_{K_A}\}_{K_B} = \{\{\text{pin: 3443}\}_{K_B}\}_{K_A}$ by commutativity

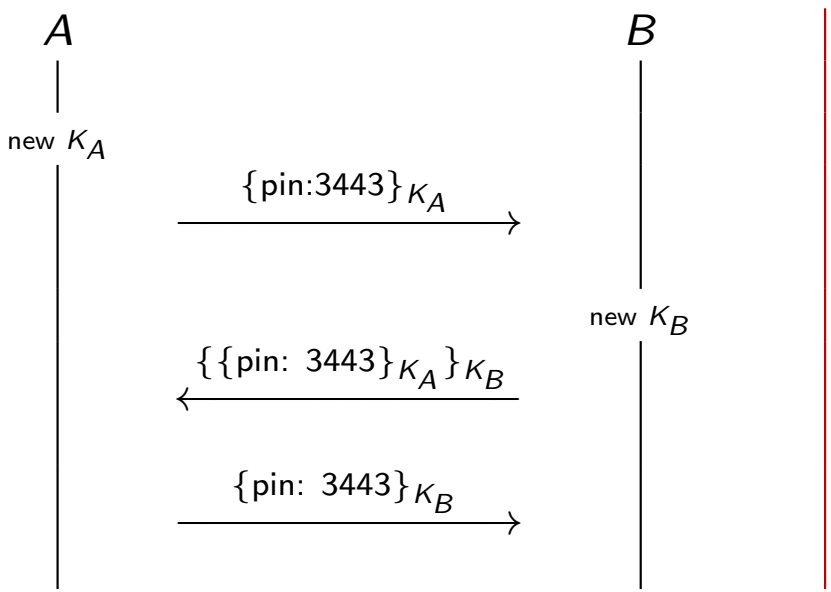
Example of a logical attack

Assume a commutative symmetric encryption scheme

$$\{\{m\}_{k_1}\}_{k_2} = \{\{m\}_{k_2}\}_{k_1}$$

where $\{m\}_k$ denotes the encryption of message m under the key k
Example: stream ciphers

No authentication!



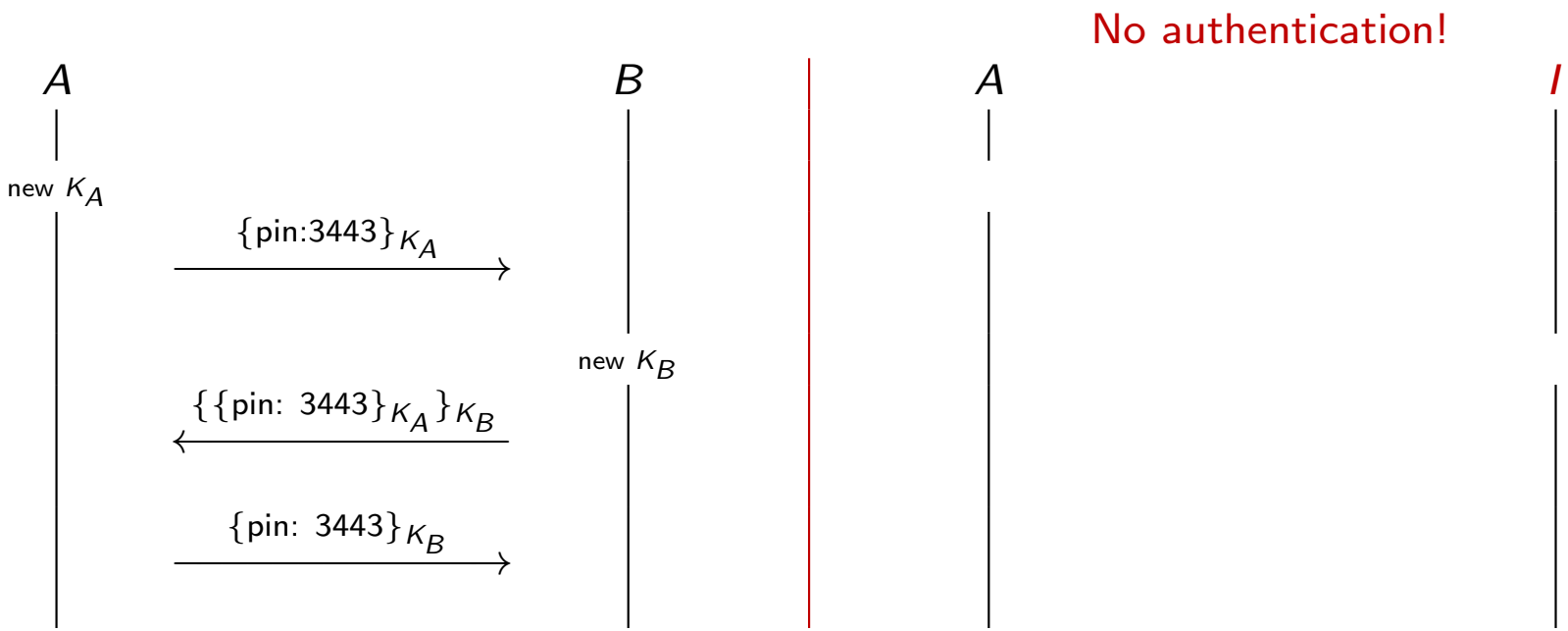
since $\{\{\text{pin: 3443}\}_{K_A}\}_{K_B} = \{\{\text{pin: 3443}\}_{K_B}\}_{K_A}$ by commutativity

Example of a logical attack

Assume a commutative symmetric encryption scheme

$$\{\{m\}_{k_1}\}_{k_2} = \{\{m\}_{k_2}\}_{k_1}$$

where $\{m\}_k$ denotes the encryption of message m under the key k
Example: stream ciphers



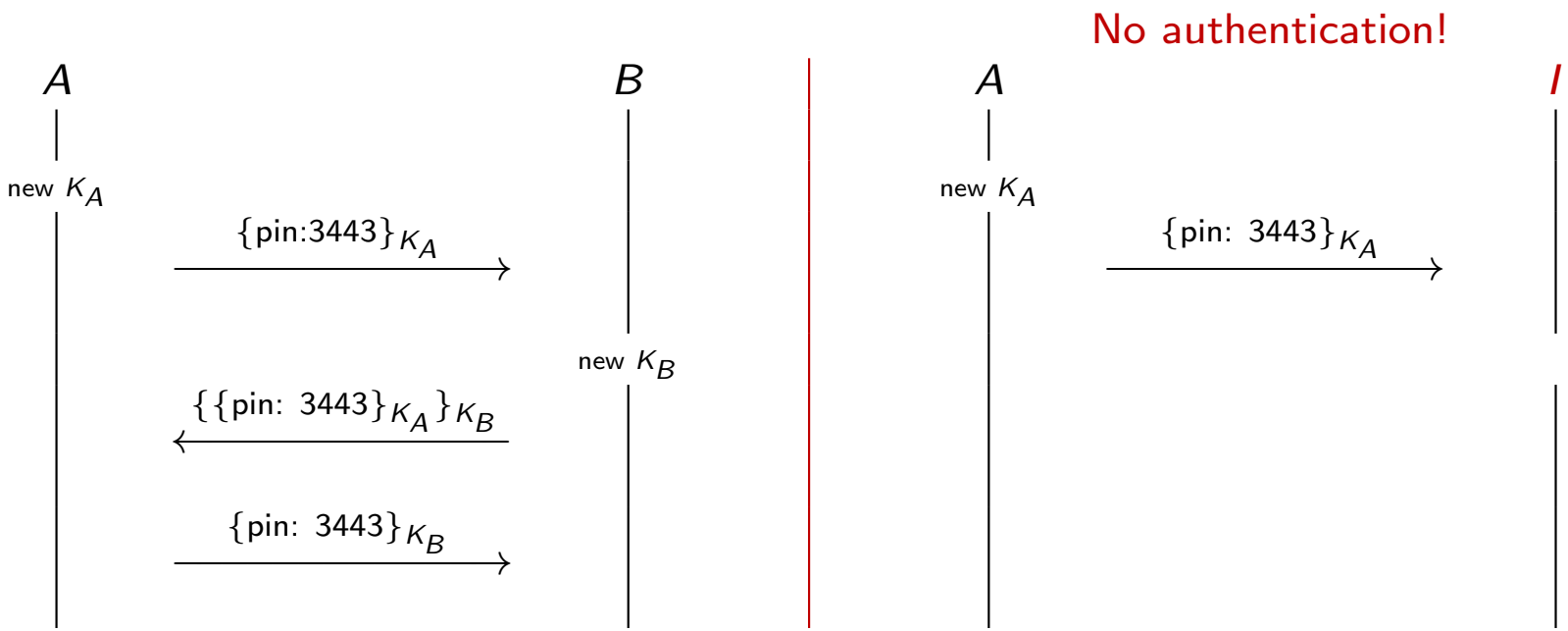
since $\{\{\text{pin: 3443}\}_{K_A}\}_{K_B} = \{\{\text{pin: 3443}\}_{K_B}\}_{K_A}$ by commutativity

Example of a logical attack

Assume a commutative symmetric encryption scheme

$$\{\{m\}_{k_1}\}_{k_2} = \{\{m\}_{k_2}\}_{k_1}$$

where $\{m\}_k$ denotes the encryption of message m under the key k
Example: stream ciphers



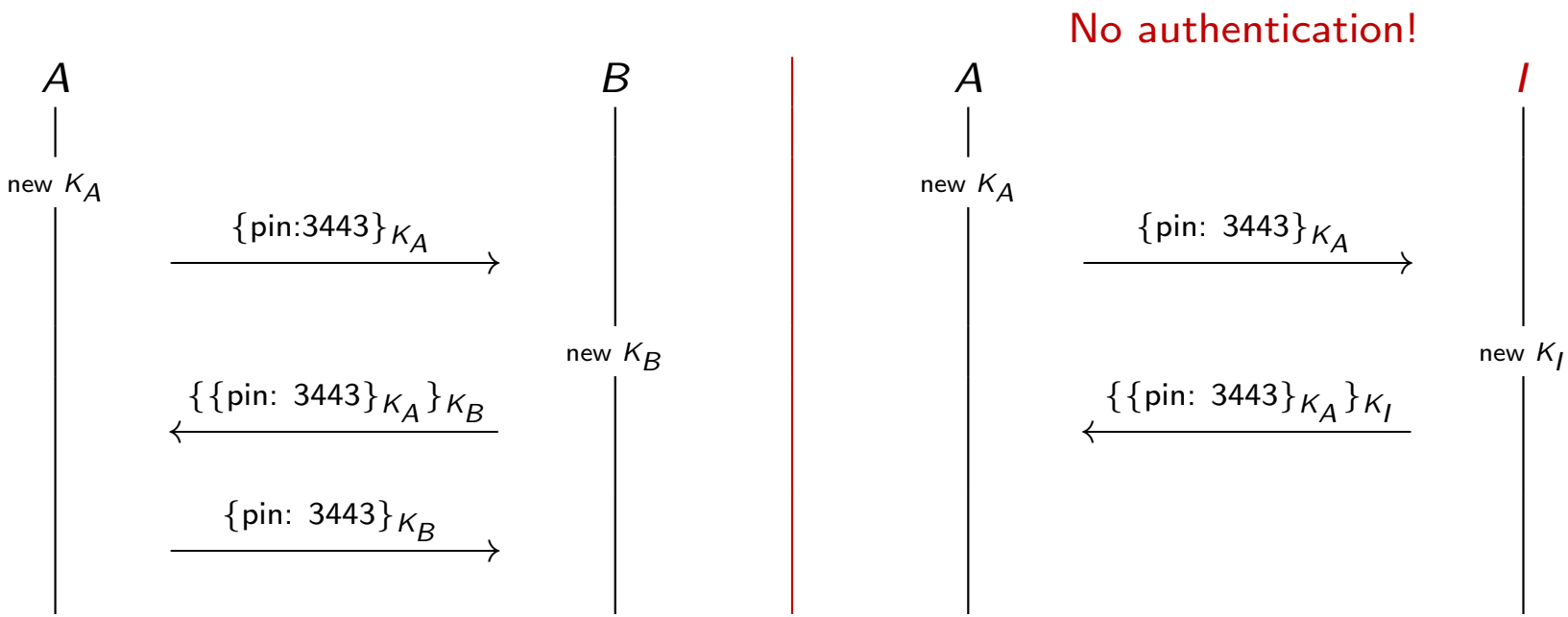
since $\{\{\text{pin: 3443}\}_{K_A}\}_{K_B} = \{\{\text{pin: 3443}\}_{K_B}\}_{K_A}$ by commutativity

Example of a logical attack

Assume a commutative symmetric encryption scheme

$$\{\{m\}_{k_1}\}_{k_2} = \{\{m\}_{k_2}\}_{k_1}$$

where $\{m\}_k$ denotes the encryption of message m under the key k
Example: stream ciphers



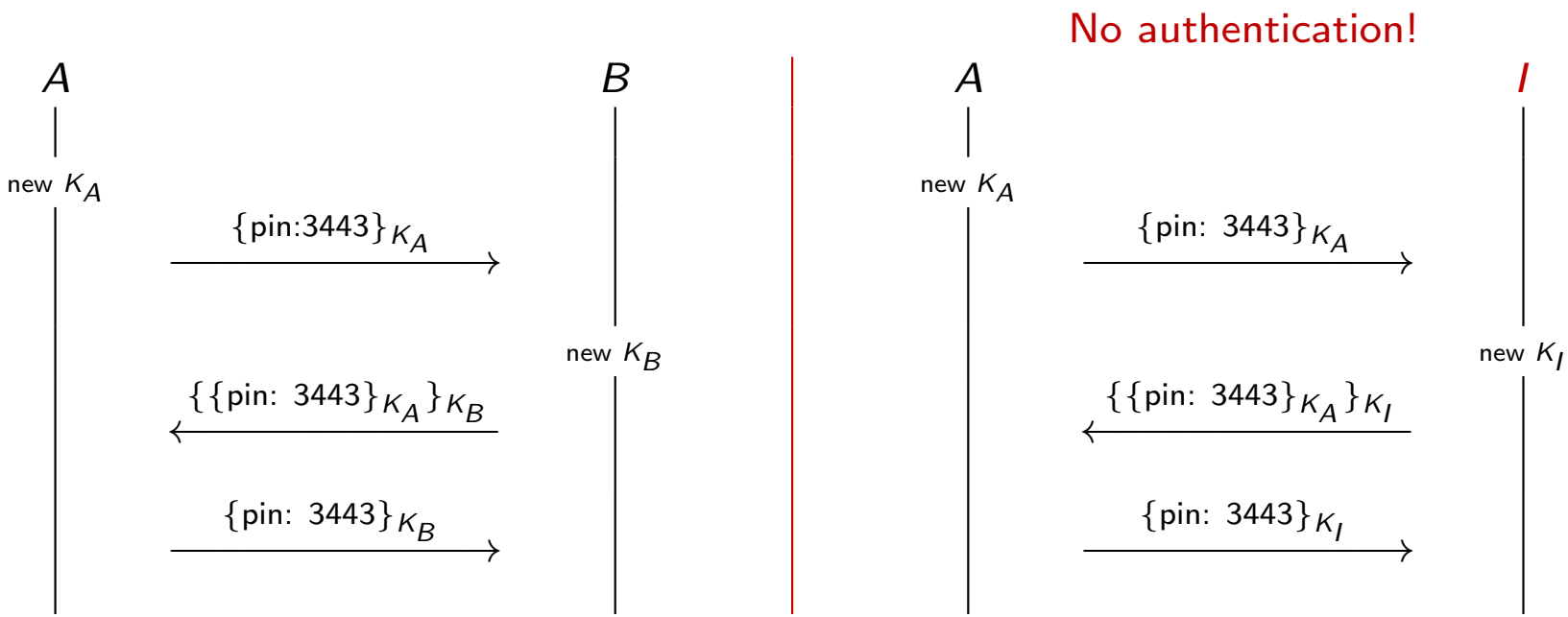
since $\{\{\text{pin: 3443}\}_{K_A}\}_{K_B} = \{\{\text{pin: 3443}\}_{K_B}\}_{K_A}$ by commutativity

Example of a logical attack

Assume a commutative symmetric encryption scheme

$$\{\{m\}_{k_1}\}_{k_2} = \{\{m\}_{k_2}\}_{k_1}$$

where $\{m\}_k$ denotes the encryption of message m under the key k
Example: stream ciphers



since $\{\{\text{pin: 3443}\}_{K_A}\}_{K_B} = \{\{\text{pin: 3443}\}_{K_B}\}_{K_A}$ by commutativity

Authentication and key agreement protocols

Authentication and key agreement

- Long-term keys should be used as little as possible to to reduce "attack-surface"
- The use of a key should be restricted to a specific purpose
  e.g. you shouldn't use the same RSA key both for encryption and signing
- Public key algorithms tend to be computationally more expensive than symmetric key algorithms

$\rightsquigarrow$ Long-term keys are used to establish short-term session keys
e.g. TLS over HTTP, AKA for 3G, BAC for epassports, *etc.*

Needham-Schroeder Public Key (NSPK)

NSPK: authentication and key agreement protocol

[Roger Needham, Michael Schroeder, "Using encryption for authentication in large networks of computers". Communications of the ACM (December 1978)]

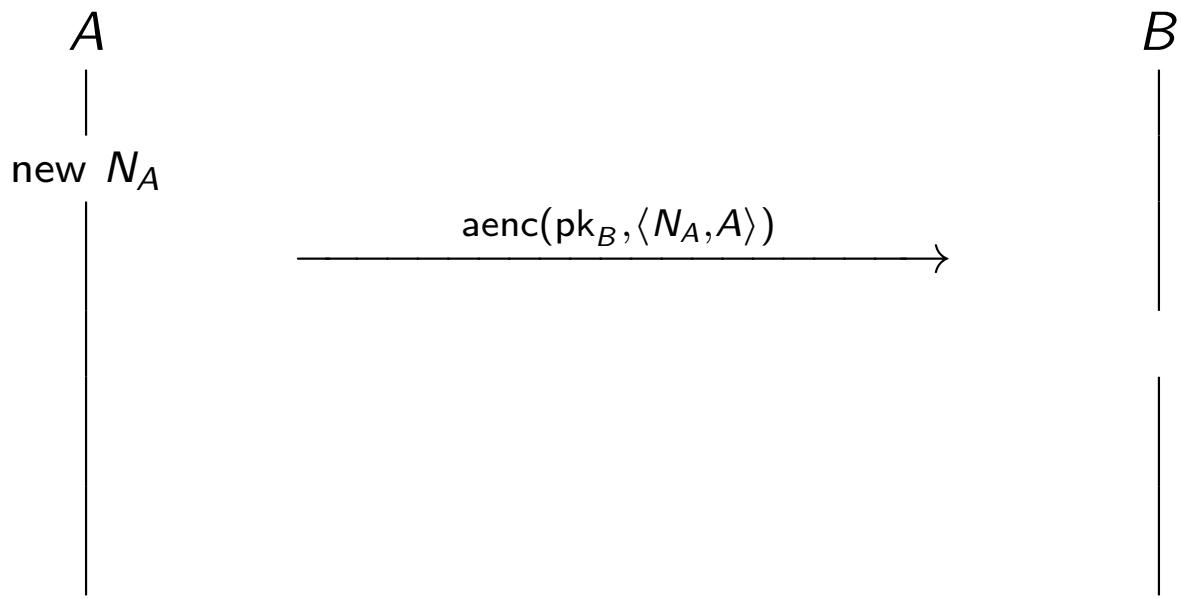
Needham-Schroeder Public Key (NSPK)

NSPK: authentication and key agreement protocol

[Roger Needham, Michael Schroeder, "Using encryption for authentication in large networks of computers". Communications of the ACM (December 1978)]

Needham-Schroeder Public Key (NSPK)

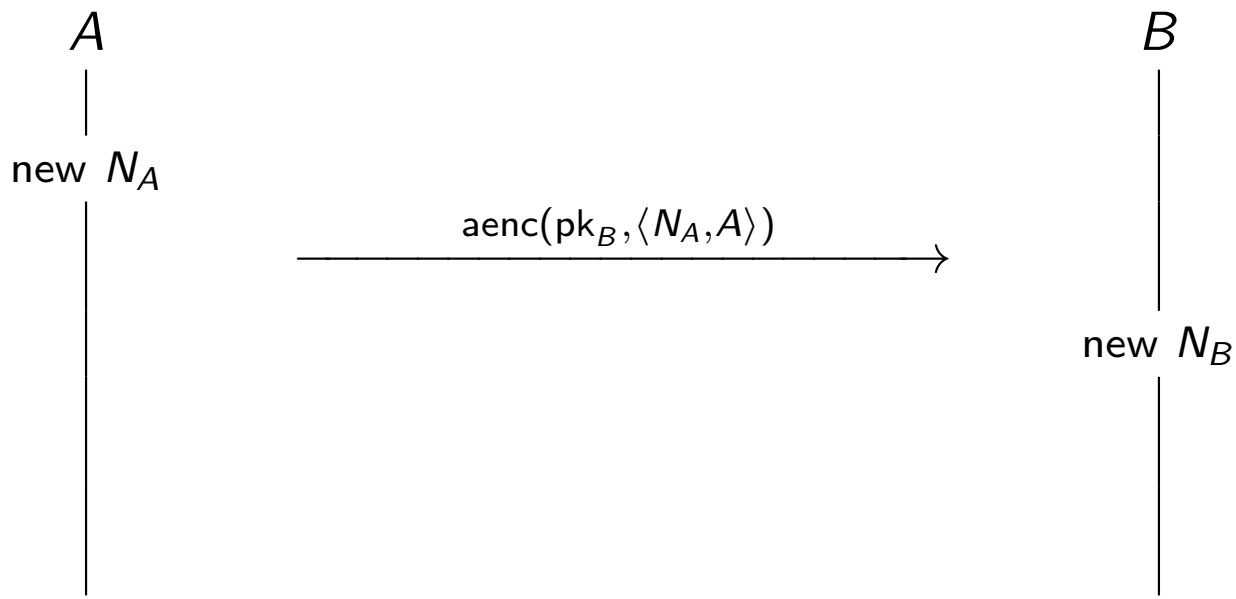
NSPK: authentication and key agreement protocol



[Roger Needham, Michael Schroeder, "Using encryption for authentication in large networks of computers". Communications of the ACM (December 1978)]

Needham-Schroeder Public Key (NSPK)

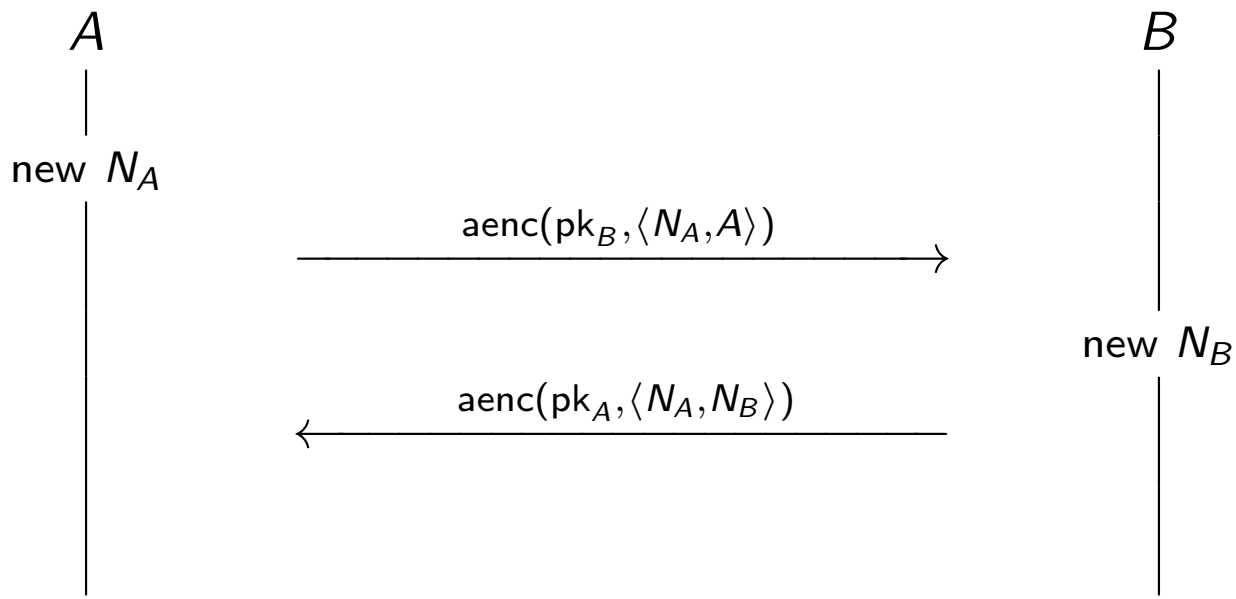
NSPK: authentication and key agreement protocol



[Roger Needham, Michael Schroeder, "Using encryption for authentication in large networks of computers". Communications of the ACM (December 1978)]

Needham-Schroeder Public Key (NSPK)

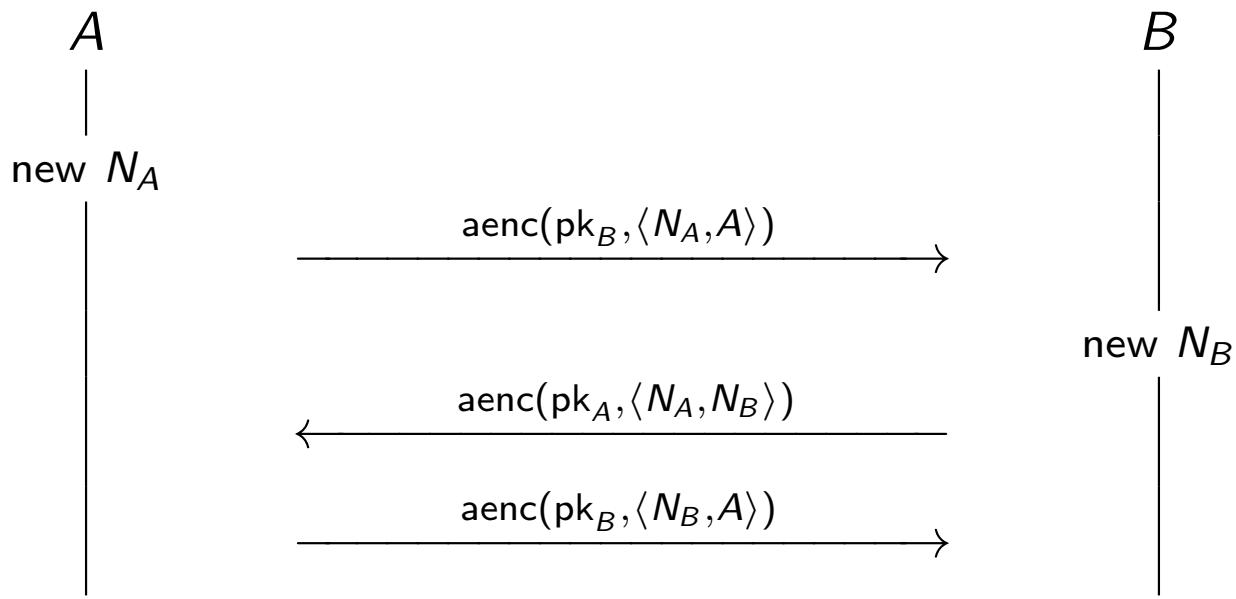
NSPK: authentication and key agreement protocol



[Roger Needham, Michael Schroeder, "Using encryption for authentication in large networks of computers". Communications of the ACM (December 1978)]

Needham-Schroeder Public Key (NSPK)

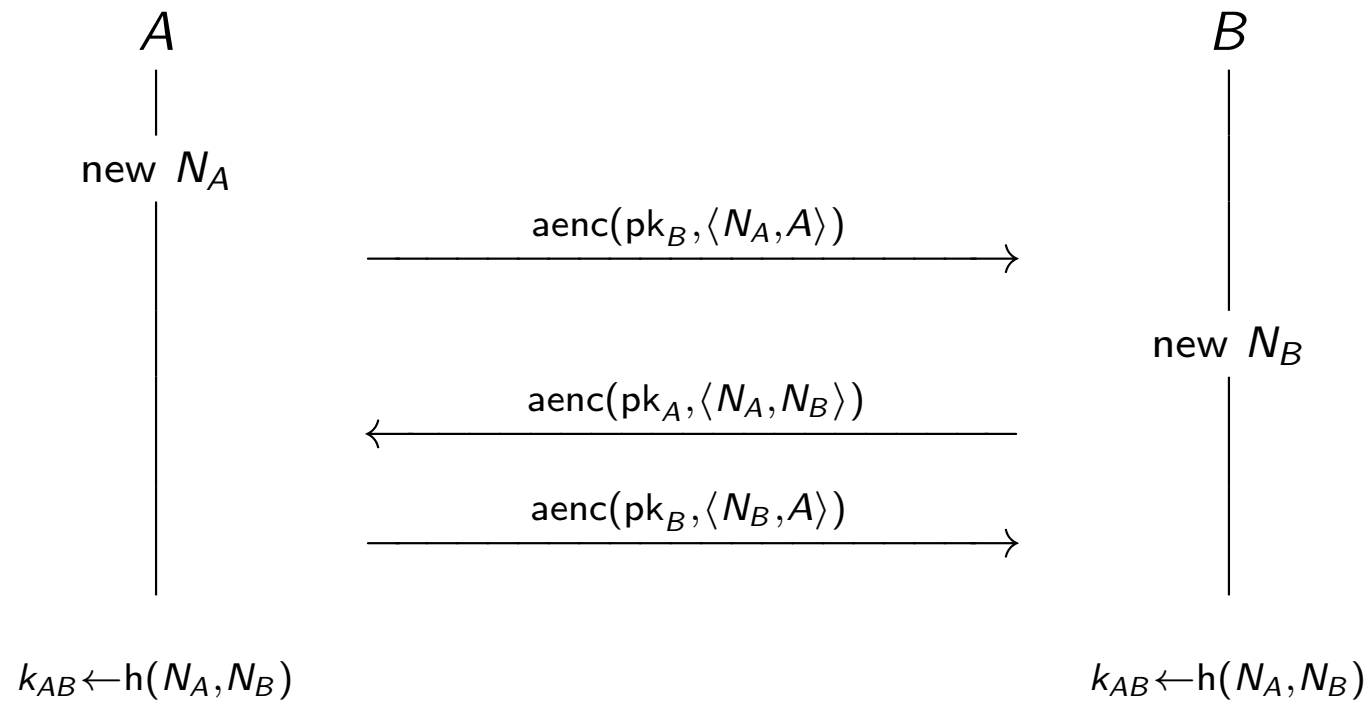
NSPK: authentication and key agreement protocol



[Roger Needham, Michael Schroeder, "Using encryption for authentication in large networks of computers". Communications of the ACM (December 1978)]

Needham-Schroeder Public Key (NSPK)

NSPK: authentication and key agreement protocol



[Roger Needham, Michael Schroeder, "Using encryption for authentication in large networks of computers". Communications of the ACM (December 1978)]

NSPK: security requirements

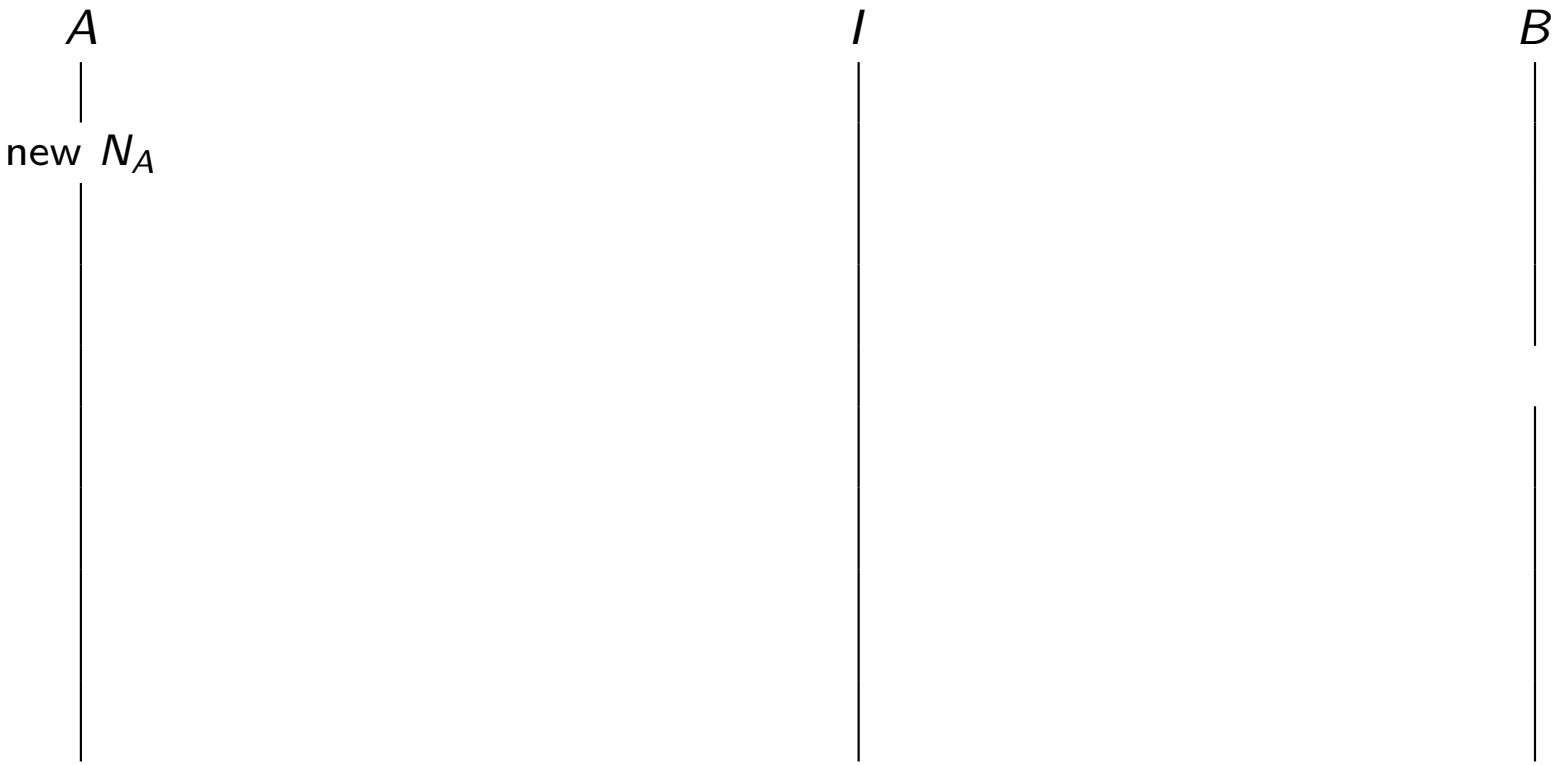
- Authentication: if Alice has completed the protocol, apparently with Bob, then Bob must also have completed the protocol with Alice.
- Authentication: If Bob has completed the protocol, apparently with Alice, then Alice must have completed the protocol with Bob.
- Confidentiality: Messages sent encrypted with the agreed key ($k \leftarrow \mathsf{h}(N_A, N_B)$) remain secret.

NSPK: Lowe's attack on authentication

Attack found 17 years after the publication of the NS protocol!!

NSPK: Lowe's attack on authentication

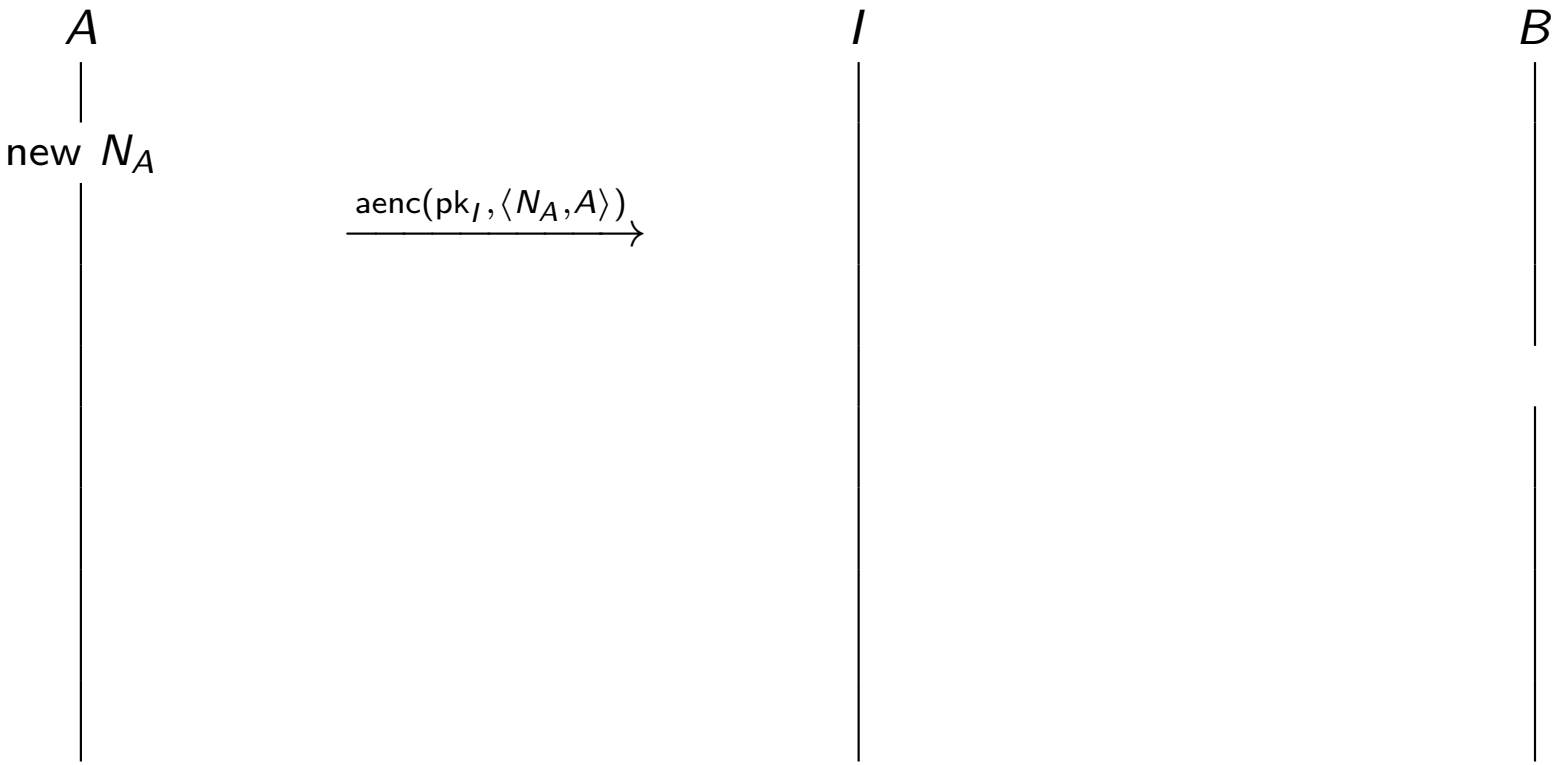
Attack found 17 years after the publication of the NS protocol!!



[G. Lowe. "An attack on the Needham-Schroeder public key authentication protocol". Information Processing Letters (November 1995)]

NSPK: Lowe's attack on authentication

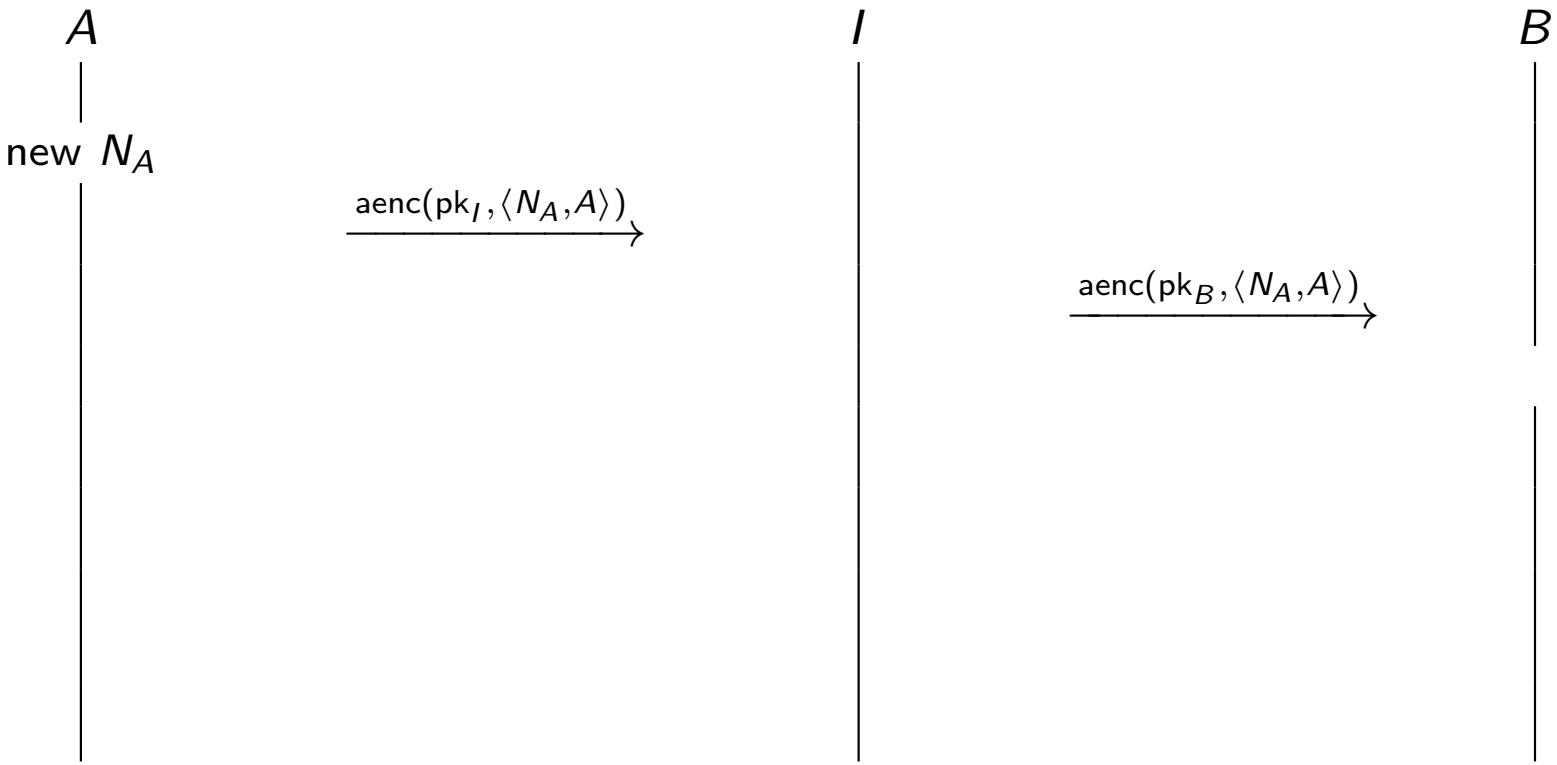
Attack found 17 years after the publication of the NS protocol!!



[G. Lowe. "An attack on the Needham-Schroeder public key authentication protocol". Information Processing Letters (November 1995)]

NSPK: Lowe's attack on authentication

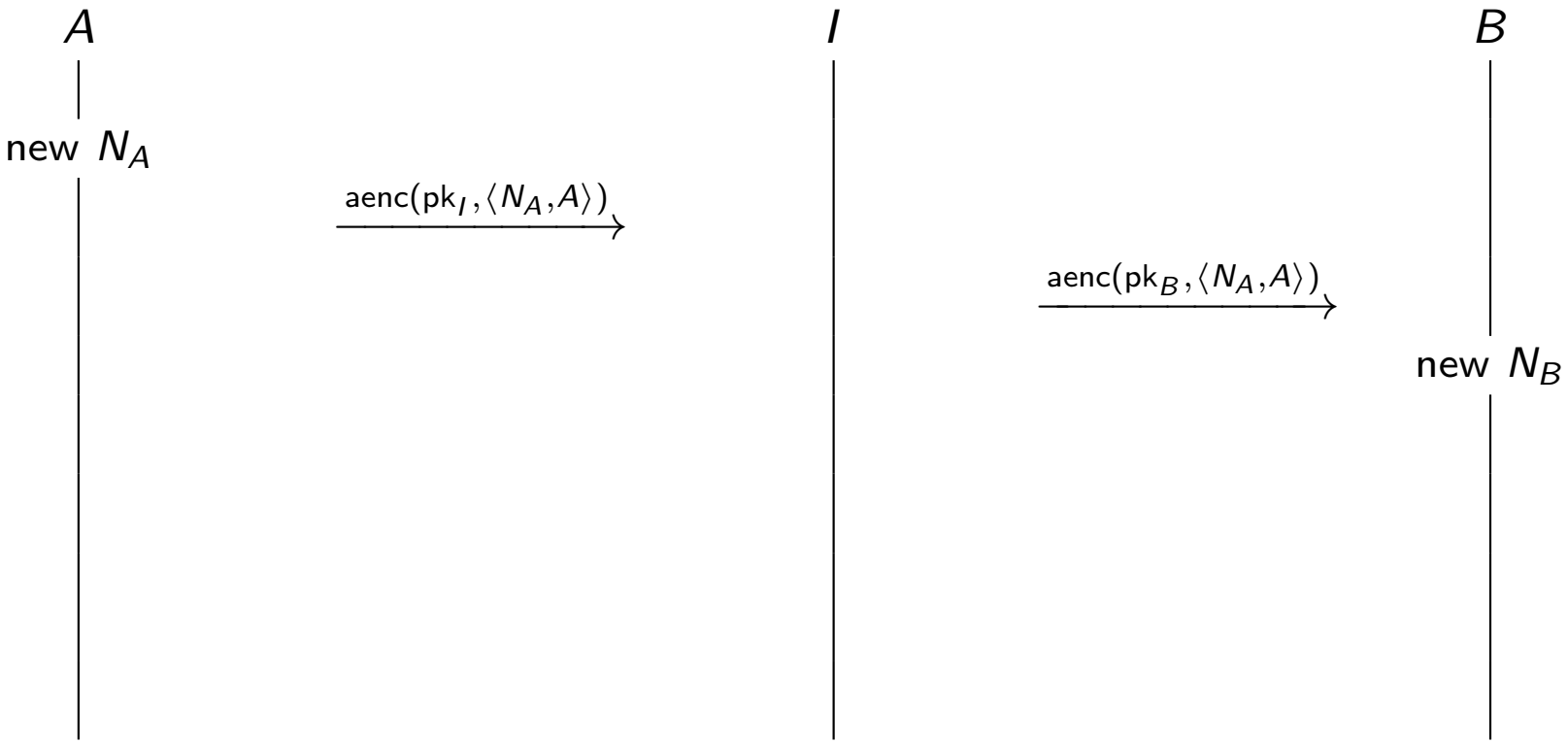
Attack found 17 years after the publication of the NS protocol!!



[G. Lowe. "An attack on the Needham-Schroeder public key authentication protocol". Information Processing Letters (November 1995)]

NSPK: Lowe's attack on authentication

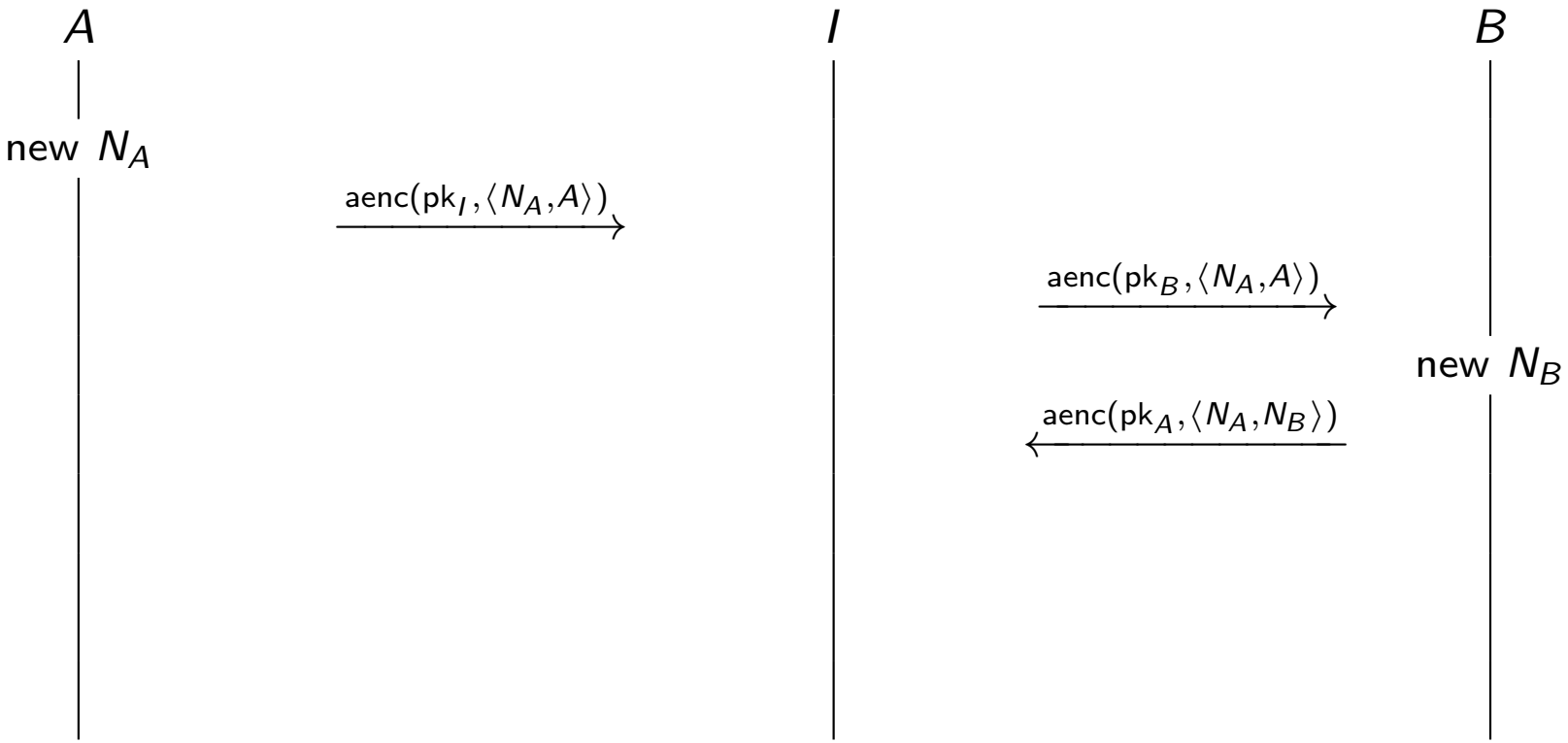
Attack found 17 years after the publication of the NS protocol!!



[G. Lowe. "An attack on the Needham-Schroeder public key authentication protocol". Information Processing Letters (November 1995)]

NSPK: Lowe's attack on authentication

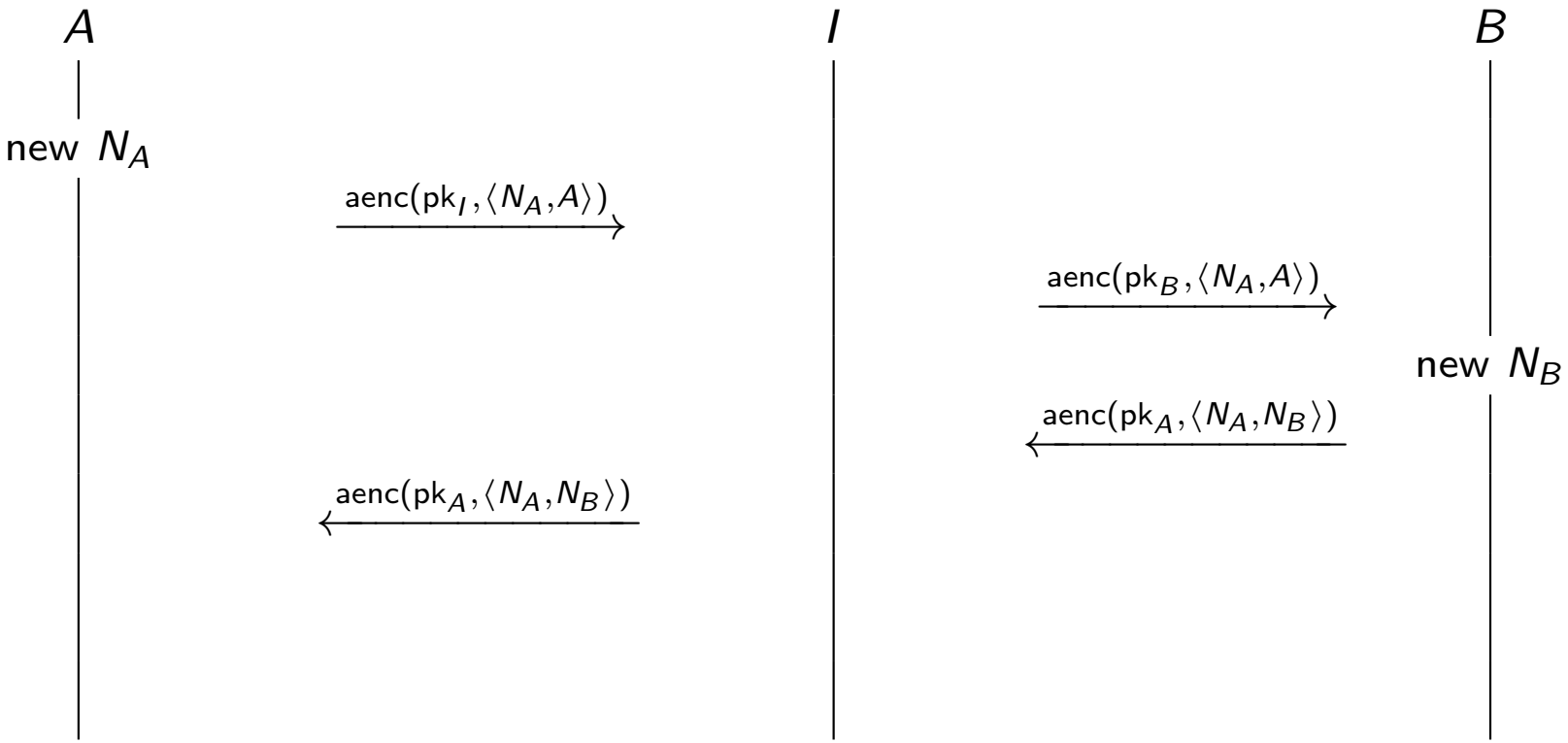
Attack found 17 years after the publication of the NS protocol!!



[G. Lowe. "An attack on the Needham-Schroeder public key authentication protocol". Information Processing Letters (November 1995)]

NSPK: Lowe's attack on authentication

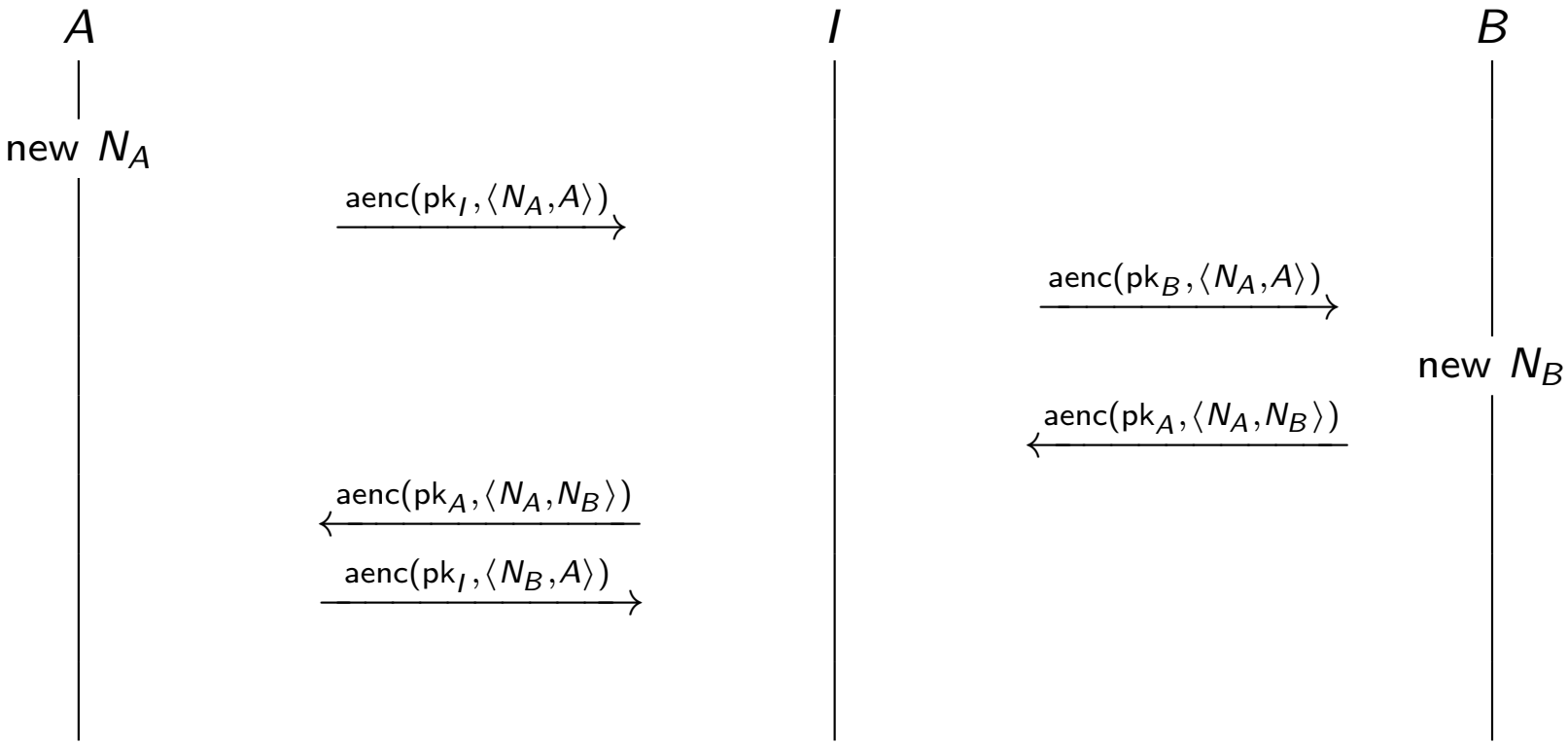
Attack found 17 years after the publication of the NS protocol!!



[G. Lowe. "An attack on the Needham-Schroeder public key authentication protocol". Information Processing Letters (November 1995)]

NSPK: Lowe's attack on authentication

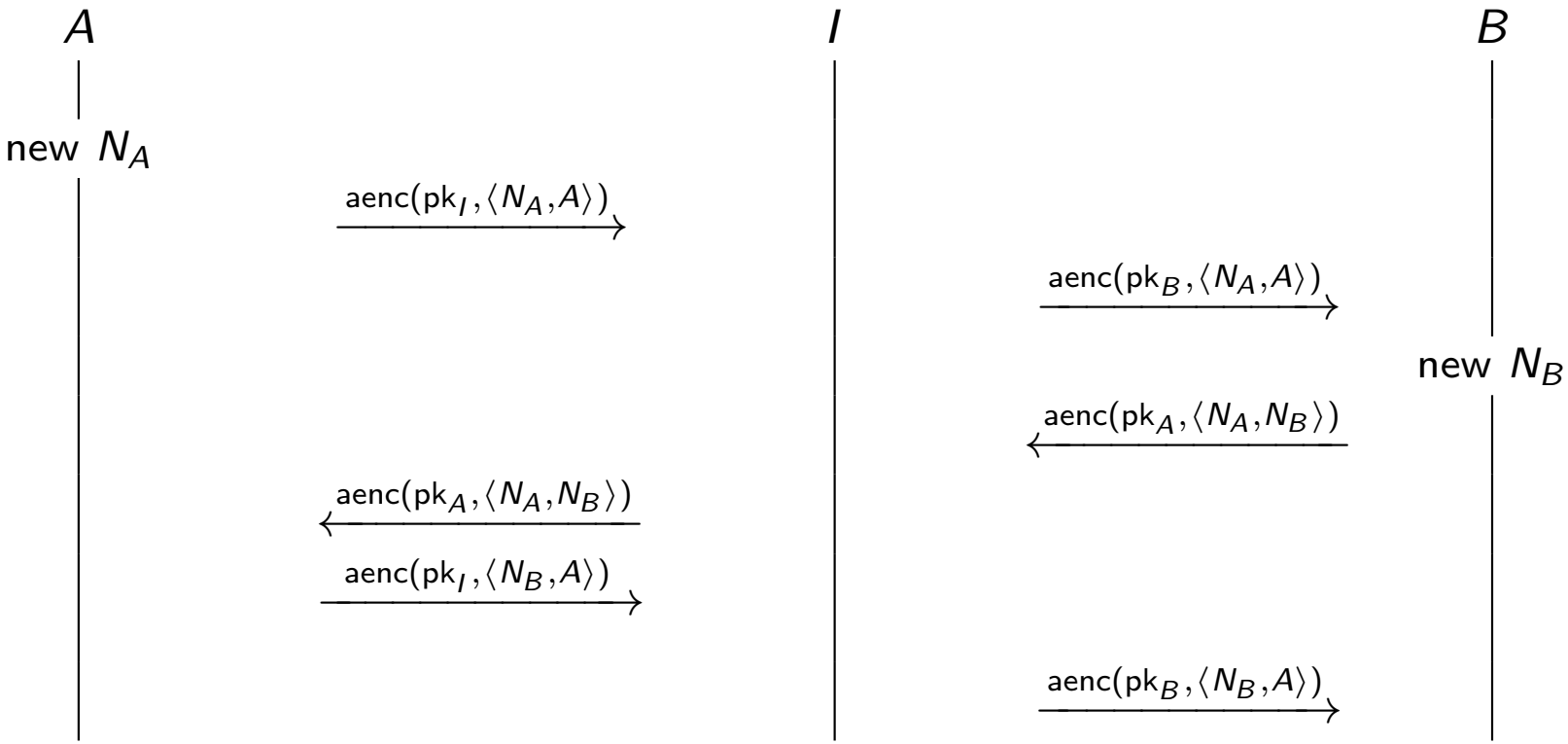
Attack found 17 years after the publication of the NS protocol!!



[G. Lowe. "An attack on the Needham-Schroeder public key authentication protocol". Information Processing Letters (November 1995)]

NSPK: Lowe's attack on authentication

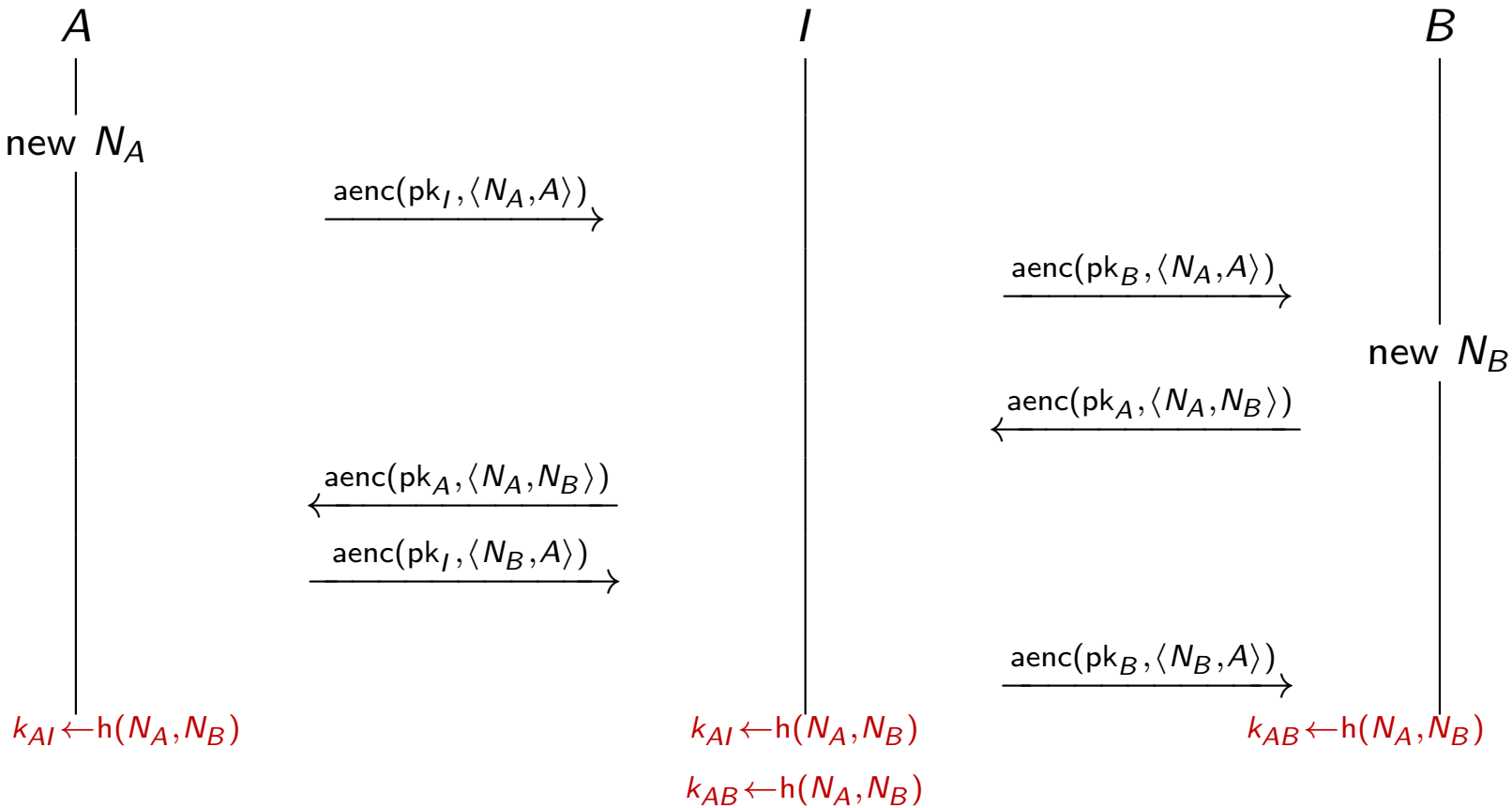
Attack found 17 years after the publication of the NS protocol!!



[G. Lowe. "An attack on the Needham-Schroeder public key authentication protocol". Information Processing Letters (November 1995)]

NSPK: Lowe's attack on authentication

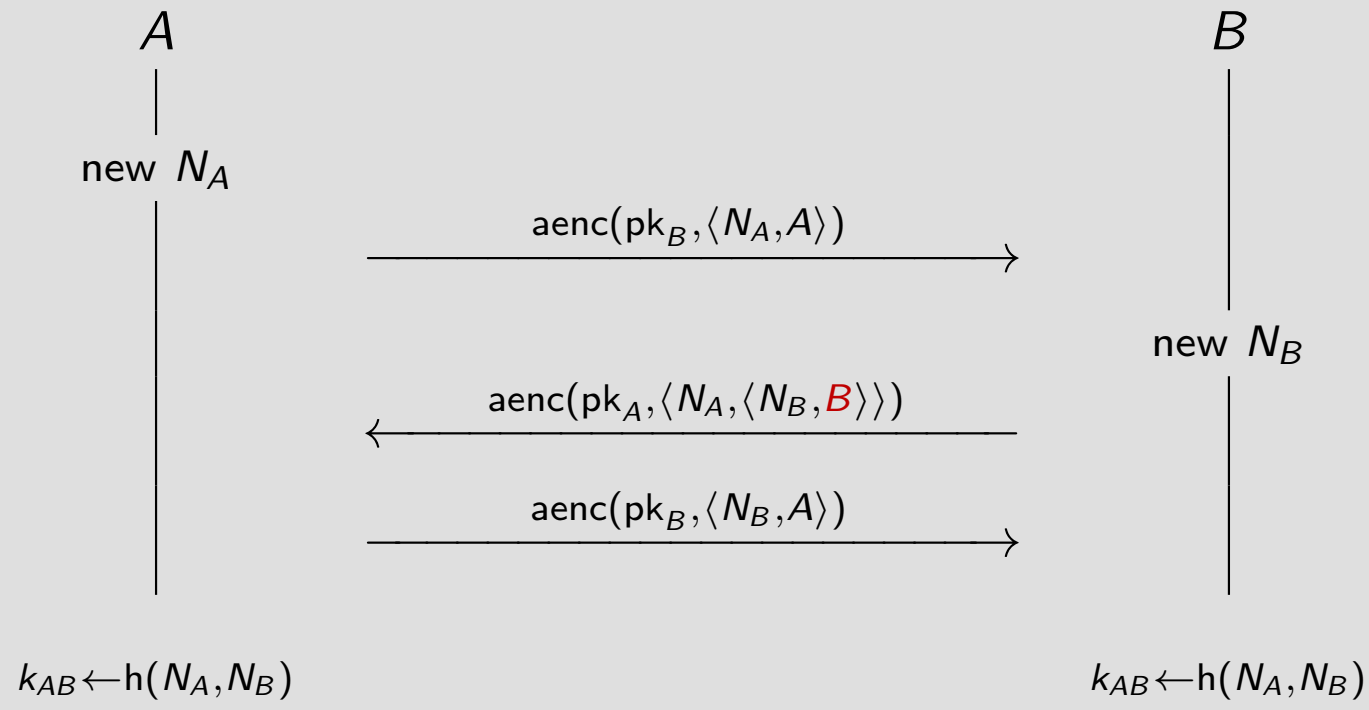
Attack found 17 years after the publication of the NS protocol!!



[G. Lowe. "An attack on the Needham-Schroeder public key authentication protocol". Information Processing Letters (November 1995)]

NSPK: Lowe's fix

The Needham-Schroeder-Lowe (NSL) protocol



Forward secrecy

- The NSL protocol is secure against an attacker that controls the network.
- What if the Alice's and Bob's private keys get compromised?
- What if the government forces Alice and Bob to reveal their private keys?
- Can we still protect confidentiality?

Forward secrecy

A protocol ensures forward secrecy, if even if long-term keys are compromised, past sessions of the protocol are still kept confidential, and this even if an attacker actively interferred.

The Station-to-Station (StS) protocol

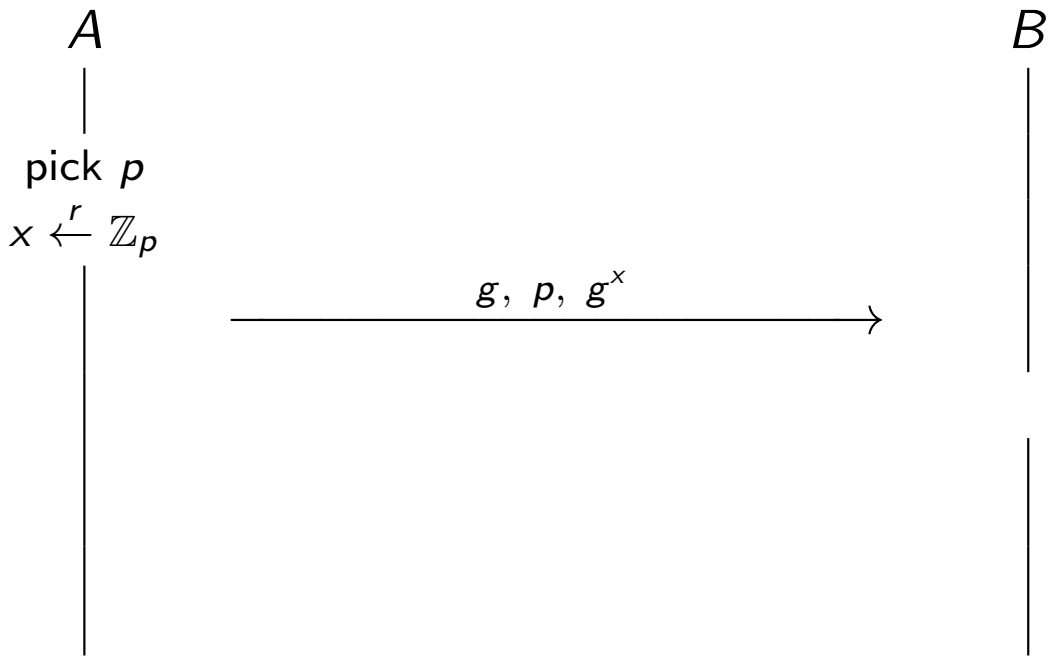
The Station-to-Station (StS) protocol

- where p is a large prime
- and g a generator of $\mathbb{Z}_p^*$

The Station-to-Station (StS) protocol

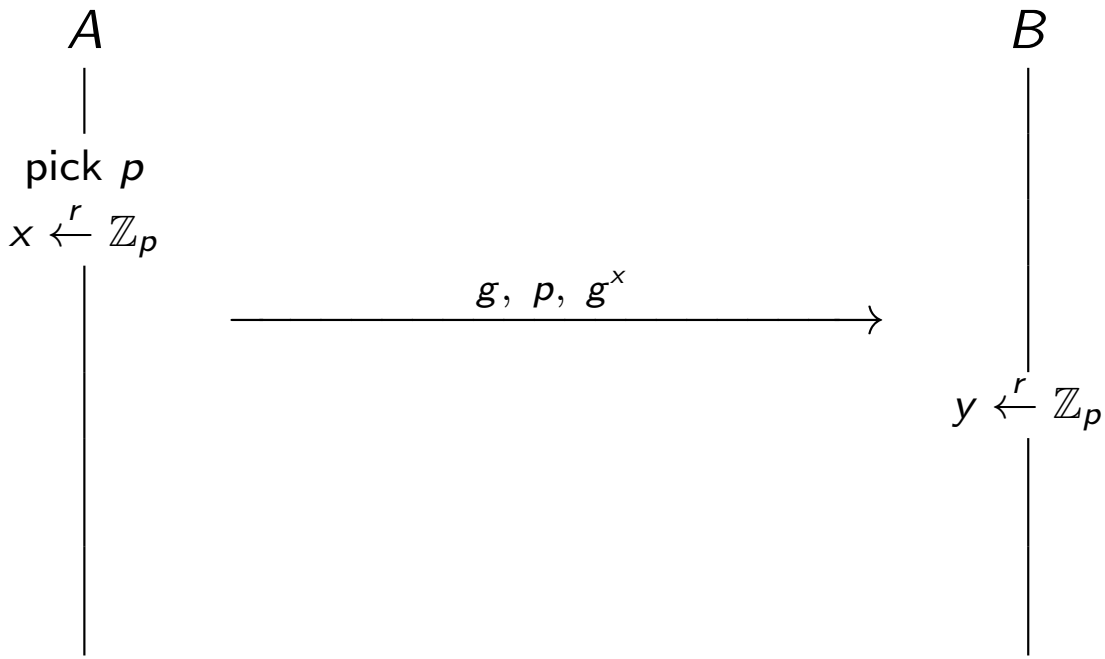
- where p is a large prime
- and g a generator of $\mathbb{Z}_p^*$

The Station-to-Station (StS) protocol



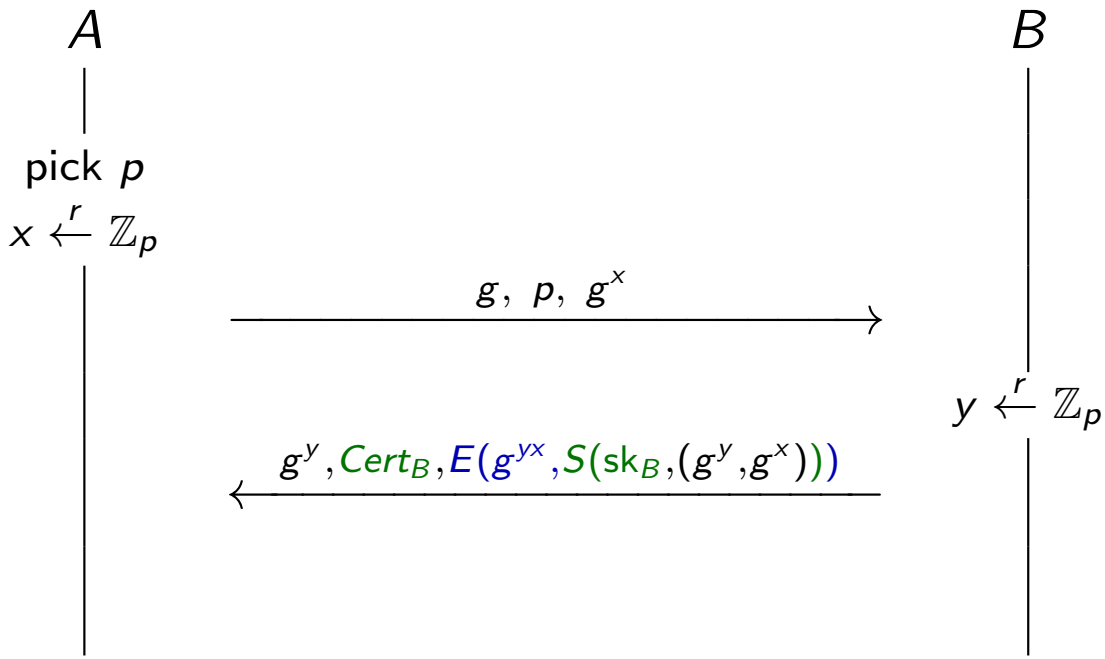
- where p is a large prime
- and g a generator of $\mathbb{Z}_p^*$

The Station-to-Station (StS) protocol



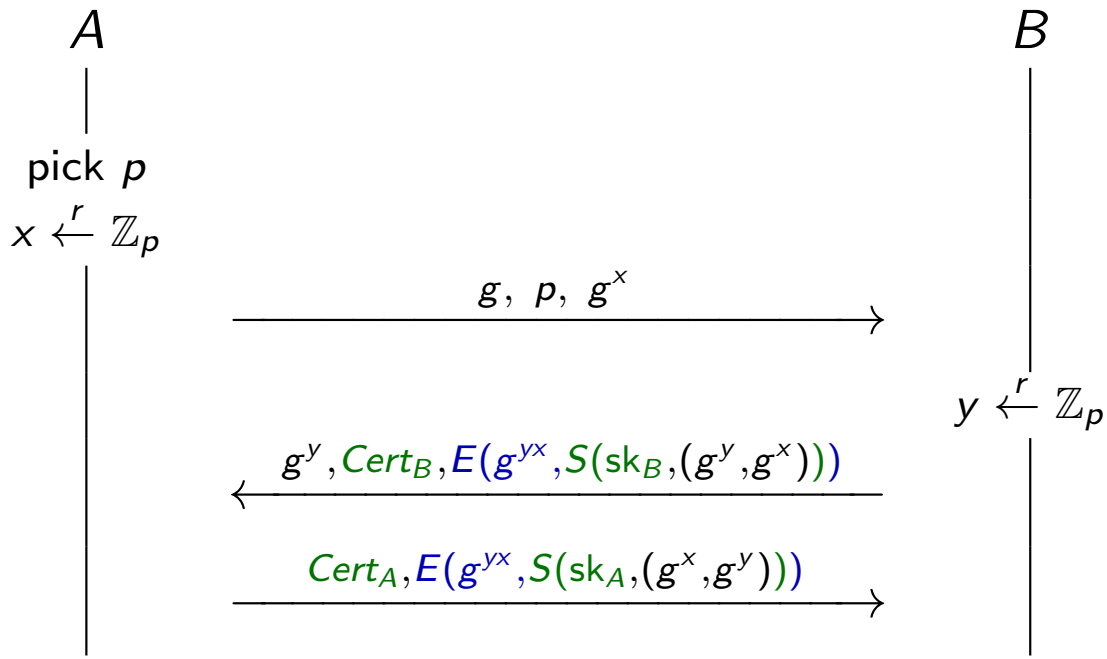
- where p is a large prime
- and g a generator of $\mathbb{Z}_p^*$

The Station-to-Station (StS) protocol



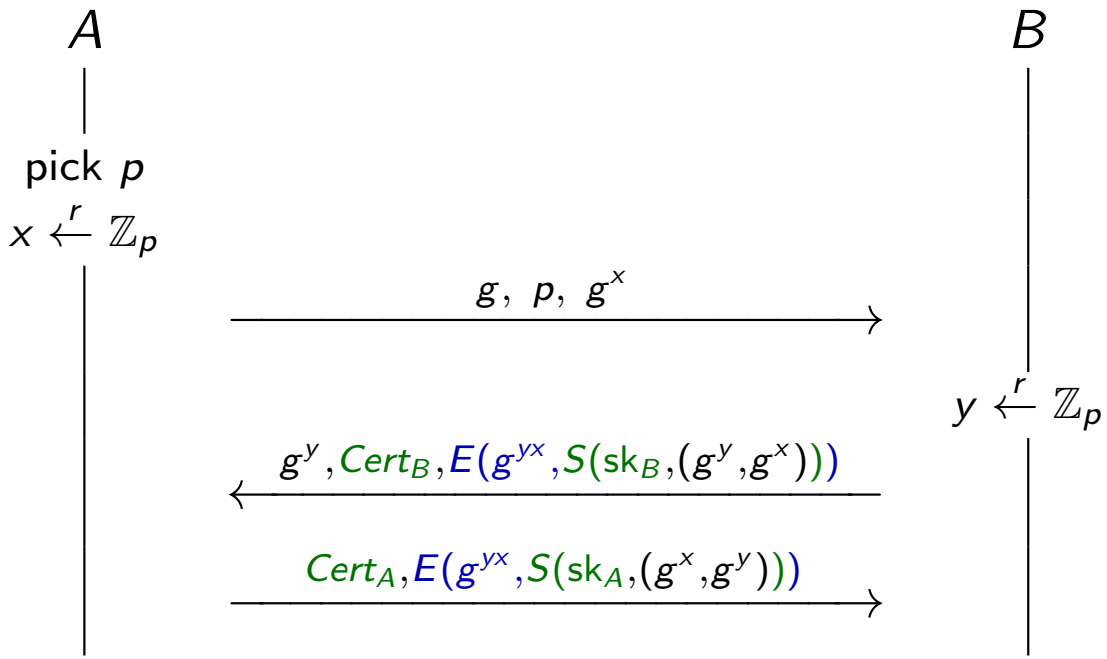
- where p is a large prime
- and g a generator of $\mathbb{Z}_p^*$

The Station-to-Station (StS) protocol



- where p is a large prime
- and g a generator of $\mathbb{Z}_p^*$

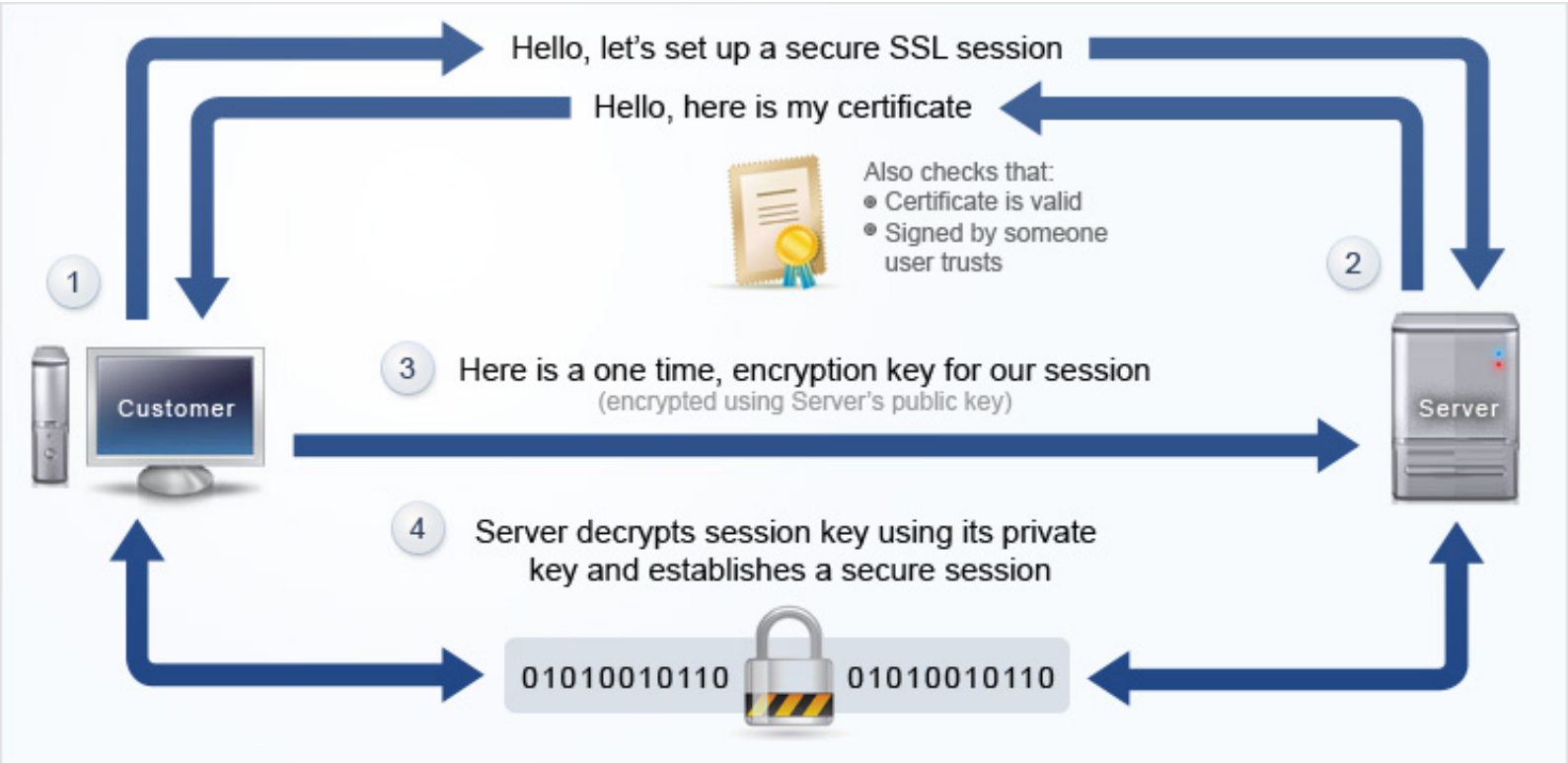
The Station-to-Station (StS) protocol



- where p is a large prime
- and g a generator of $\mathbb{Z}_p^*$

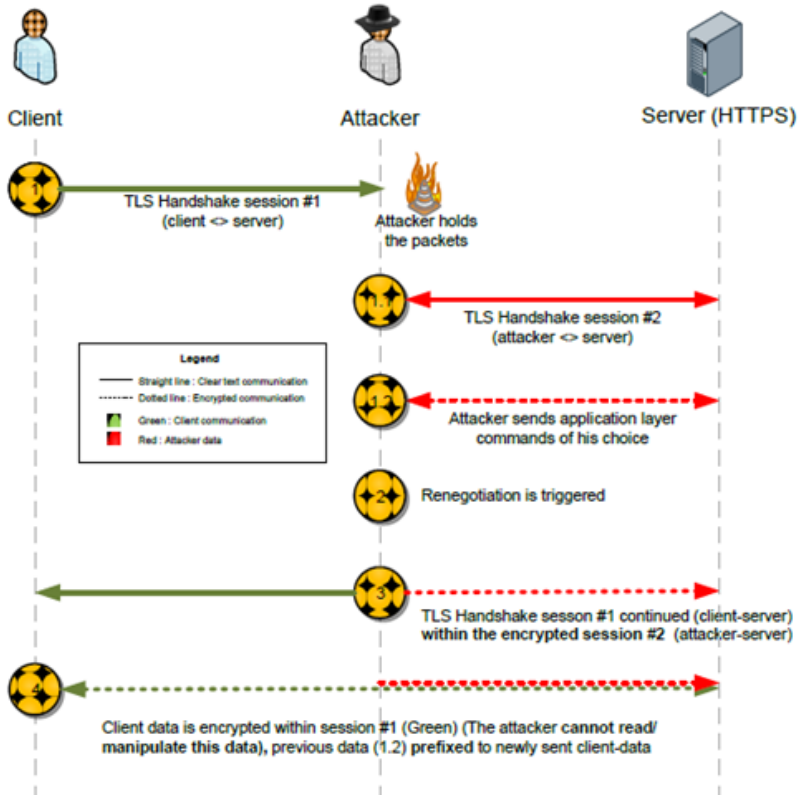
The StS ensures mutual authentication, key agreement, and forward secrecy

Old SSL/TLS handshake protocol



SSL/TLS renegotiation weaknesses

- Renegotiation has priority over application data!
- Renegotiation can take place in the middle of an application layer transaction!



Incorrect implicit assumtion: the client doesn't change through renegotiation

Marsh Ray's plaintext injection attack on HTTPS

Marsh Ray's plaintext injection attack on HTTPS

Attacker:

```
GET /pizza?toppings=pepperoni;address=attacker_str HTTP/1.1
X-Ignore-This:(no carriage return)
```

Marsh Ray's plaintext injection attack on HTTPS

Attacker:

```
GET /pizza?toppings=pepperoni;address=attacker_str HTTP/1.1
X-Ignore-This:(no carriage return)
```

Victim:

```
GET /pizza?toppings=sausage;address=victim_str HTTP/1.1
Cookie:victim_cookie
```

Marsh Ray's plaintext injection attack on HTTPS

Attacker:

```
GET /pizza?toppings=pepperoni;address=attacker_str HTTP/1.1
X-Ignore-This:(no carriage return)
```

Victim:

```
GET /pizza?toppings=sausage;address=victim_str HTTP/1.1
Cookie:victim_cookie
```

Result:

```
GET /pizza?toppings=pepperoni;address=attacker_str HTTP/1.1
X-Ignore-This:GET /pizza?toppings=sausage;address=victim_str HTTP/1.1
Cookie:victim_cookie
```

Marsh Ray's plaintext injection attack on HTTPS

Attacker:

```
GET /pizza?toppings=pepperoni;address=attacker_str HTTP/1.1
X-Ignore-This:(no carriage return)
```

Victim:

```
GET /pizza?toppings=sausage;address=victim_str HTTP/1.1
Cookie:victim_cookie
```

Result:

```
GET /pizza?toppings=pepperoni;address=attacker_str HTTP/1.1
X-Ignore-This:GET /pizza?toppings=sausage;address=victim_str HTTP/1.1
Cookie:victim_cookie
```

$\Rightarrow$ **Server uses victim's account to send a pizza to attacker!**

Anil Kurmus' plaintext injection attack on HTTPS

Anil Kurmus' plaintext injection attack on HTTPS

Twitter status updates using its API by posting the new status to http://twitter.com/statuses/update.xml, as well as the user name and password

Anil Kurmus' plaintext injection attack on HTTPS

Twitter status updates using its API by posting the new status to http://twitter.com/statuses/update.xml, as well as the user name and password

Attacker:

```
POST /statuses/update.xml HTTP/1.1
Authorization:  Basic username:password
User-Agent:  curl/7.19.5
Host:  twitter.com
Accept:*/*
Content-Length:  140
Content-Type:  application/x-www-form-urlencoded
status=
```

Anil Kurmus' plaintext injection attack on HTTPS

Twitter status updates using its API by posting the new status to http://twitter.com/statuses/update.xml, as well as the user name and password

Attacker:

```
POST /statuses/update.xml HTTP/1.1
Authorization:  Basic username:password
User-Agent:  curl/7.19.5
Host:  twitter.com
Accept:*/*
Content-Length:  140
Content-Type:  application/x-www-form-urlencoded
status=
```

Victim:

```
POST /statuses/update.xml HTTP/1.1
Authorization:  Basic username:password...
```

Anil Kurmus' plaintext injection attack on HTTPS

Twitter status updates using its API by posting the new status to http://twitter.com/statuses/update.xml, as well as the user name and password

Attacker:

```
POST /statuses/update.xml HTTP/1.1
Authorization:  Basic username:password
User-Agent:  curl/7.19.5
Host:  twitter.com
Accept:*/*
Content-Length:  140
Content-Type:  application/x-www-form-urlencoded
status=
```

Victim:

```
POST /statuses/update.xml HTTP/1.1
Authorization:  Basic username:password...
```

$\Rightarrow$ **the attacker gets the user name and password of the victim!**