

# Secure communications: Cryptographic protocols

**Markulf Kohlweiss & Myrto Arapinis**  
School of Informatics  
University of Edinburgh

January 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  
→ **the network is the attacker**
- ▶ control dishonest participants

# The attacker controls the network (1)

Network Utility

Info | Netstat | Ping | Lookup | Traceroute | Whois | Finger | Port Scan

Enter the network address to trace an internet route to.

www.facebook.com (ex. 10.0.2.1 or www.example.com)

Trace

Traceroute has started.

```
traceroute to star-mini.c10r.facebook.com (157.240.0.35), 64 hops max, 72 byte packets
 1 knussen (129.215.91.246) 0.435 ms 0.211 ms 0.185 ms
 2 vlan160.kb9-msfc.net.ed.ac.uk (129.215.160.254) 0.502 ms 0.435 ms 0.467 ms
 3 vlan688.s-pop.eastman.ja.net (194.81.57.211) 0.886 ms 0.824 ms 0.876 ms
 4 ae2.leedaq-sbr1.ja.net (146.97.41.33) 4.578 ms 4.550 ms 4.574 ms
 5 ae30.manchk-sbr1.ja.net (146.97.33.45) 7.165 ms 7.121 ms 7.133 ms
 6 port-channel205.car1.manchester1.level3.net (195.50.119.97) 21.449 ms 10.438 ms 205.486 ms
 7 4.15.154.86 (4.15.154.86) 111.234 ms 111.232 ms 111.284 ms
 8 po103.psw01c.mia1.tfbnw.net (157.240.32.223) 110.868 ms 110.802 ms 110.810 ms
 9 157.240.36.71 (157.240.36.71) 110.913 ms 110.802 ms 110.745 ms
10 edge-star-mini-shv-02-mia1.facebook.com (157.240.0.35) 112.594 ms 112.943 ms 112.861 ms
```

## The attacker controls the network (2)



A 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



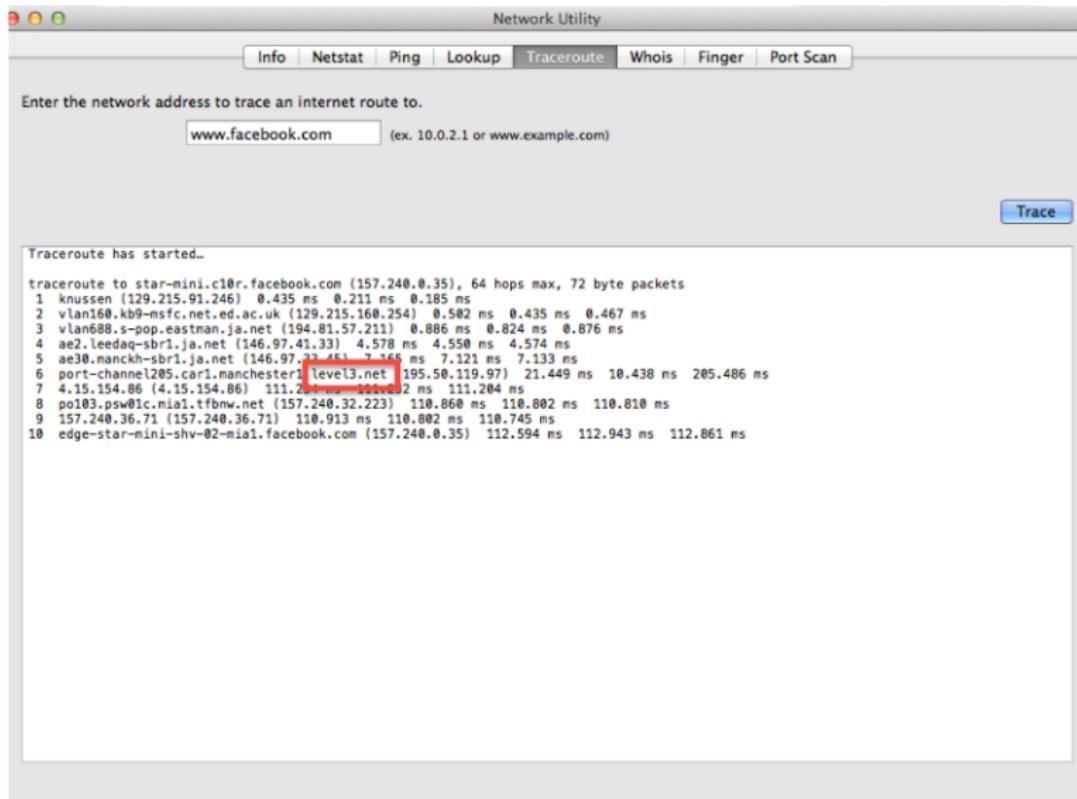
27 Nov 2013 at 02:19, Shaun Nichols



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)

A screenshot of a web browser window titled "Welcome page". The address bar shows the URL: "homepages.inf.ed.ac.uk/marapini/CSdemos/AttackerControlsNetwork/password.html". The main content area displays a "Please login" form. It contains two text input fields: "Username:" with the value "myrto" and "Password:" with the value "\*\*\*\*\*". Below these fields is a button labeled "Submit Query".

Welcome page

homepages.inf.ed.ac.uk/marapini/CSdemos/AttackerControlsNetwork/password.html

Please login

Username: myrto

Password: \*\*\*\*\*

Submit Query

# All messages can be intercepted by an attacker (2)

No.	Time	Source	Destination	Protocol	Length
1	00:00:00.000	172.16.76.2	172.16.76.2	DNS	68 standard query 0x031b A homepage.inf.ed.ac.uk
2	0.023618152	172.16.76.155	172.16.76.155	DNS	68 standard query 0xd17a A homepage.inf.ed.ac.uk
3	0.077306592	172.16.76.2	172.16.76.155	DNS	453 standard query response 0x031b A homepage.inf.ed.ac.uk 129.215.32.13 NS lewis.ucs.ed.ac.uk NS xlab-0.ed.ac.uk NS cancer.ucs.ed.ac.uk NS -
4	0.077494322	172.16.76.2	172.16.76.155	DNS	453 standard query response 0xd17a A homepage.inf.ed.ac.uk 129.215.32.13 NS xlab-0.ed.ac.uk NS cancer.ucs.ed.ac.uk NS dns2.inf.ed.ac.uk NS -
5	0.090900000	172.16.76.155	172.16.76.155	TCP	74 36412 - 89 [SYN] Seq=9 Win=29200 Lm=0 MSS=1460 SACK_PERM=1 ISval=1208371 TSscr=0 MS=0.000000000
6	0.097007751	172.16.76.155	129.215.32.13	TCP	74 36412 - 89 [SYN] Seq=9 Win=29200 Lm=0 MSS=1460 SACK_PERM=1 ISval=1208371 TSscr=0 MS=0.000000000
7	0.127399060	172.16.76.2	172.16.76.155	DNS	134 standard query response 0x031b A homepage.inf.ed.ac.uk SQA dnse.inf.ed.ac.uk
8	0.130200000	172.16.76.155	172.16.76.155	TCP	68 36412 - 89 [ACK] Seq=10 Win=29200 Lm=0 MSS=1460
9	0.160826787	172.16.76.155	129.215.32.13	TCP	54 36412 - 89 [ACK] Seq=10 Win=29200 Lm=0 MSS=1460
10	0.160825694	172.16.76.155	129.215.32.13	HTTP	689 POST /merapini/CSdemos/AttackerControlsNetwork/password.html HTTP/1.1 (application/x-www-form-urlencoded)
11	0.160830316	129.215.32.13	172.16.76.155	TCP	68 00 - 36412 [ACK] Seq=1 Ack=80 Win=4240 Lm=0
12	0.160830316	129.215.32.13	172.16.76.155	TCP	68477 TCP FIN/SYN 129.215.32.13:4218 -> 172.16.76.155:4218 Seq=12458 Ack=1218 Win=31859 Lm=0
13	0.234191625	172.16.76.155	129.215.32.13	TCP	54 36412 - 89 [ACK] Seq=630 Ack=610 Win=29481 Lm=0
14	7.94957611508	172.16.76.155	129.215.32.13	HTTP	683 POST /merapini/CSdemos/AttackerControlsNetwork/password.html HTTP/1.1 (application/x-www-form-urlencoded)
15	7.9495761295	129.215.32.13	172.16.76.155	TCP	68477 TCP FIN/SYN 129.215.32.13:4218 -> 172.16.76.155:4240 Seq=12458 Ack=31859 Lm=0
16	8.145432000	172.16.76.155	129.215.32.13	HTTP	682 HTTP/1.1 200 OK [text/html]
17	8.145496642	172.16.76.155	129.215.32.13	TCP	54 36412 - 89 [ACK] Seq=12458 Ack=31859 Lm=0

Frame 14: 680 bytes on wire (5384 bits), 680 bytes captured (5384 bits) on interface 0  
Ethernet II, Src: Vmware\_0e:08:02 (00:0c:29:9e:08:02), Dst: Vmware\_f0:7d:d2 (00:50:56:f0:7d:d2)  
Internet Protocol Version 4, Src: 172.16.76.32, Dst: 129.215.32.13  
Transmission Control Protocol, Src Port: 36412 (36412), Dst Port: 80 (80), Seq: 630, Ack: 610, Len: 680  
HTTP/1.1 200 OK [text/html]  
Host: homepage.inf.ed.ac.uk\nUser-Agent: Mozilla/5.0 (Windows NT 6.1; rv:49.0) Gecko/20100101 Firefox/49.0\nAccept: text/html,application/xhtml+xml,application/xml;q=0.9,\*/\*;q=0.8\nAccept-Language: en-US,en;q=0.5\nAccept-Encoding: gzip, deflate\nReferer: http://homepage.inf.ed.ac.uk/merapini/CSdemos/AttackerControlsNetwork/password.html\nCookie: \_ga=GA1.3.398514292.1476874824\nConnection: keep-alive\nUpgrade-Insecure-Requests: 1\nContent-Type: application/x-www-form-urlencoded\n\n[null request URI: http://homepage.inf.ed.ac.uk/merapini/CSdemos/AttackerControlsNetwork/password.html]\n[null response body]

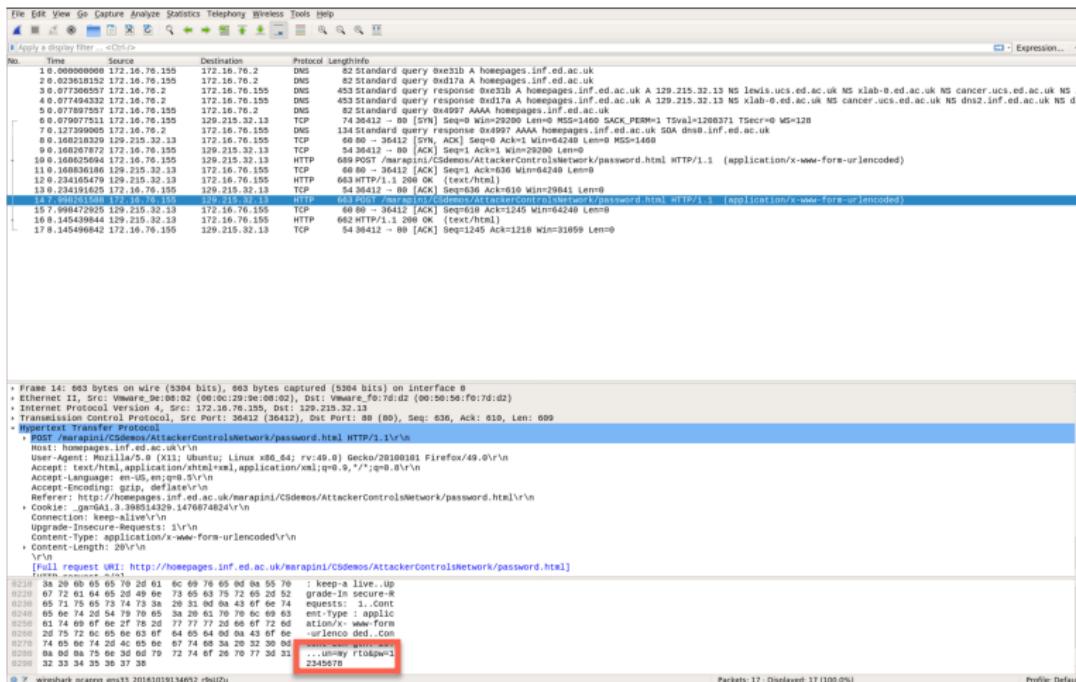
02320 38 29 05 05 05 05 29 2d 61 6c 69 65 6d 63 05 70 ... keep-alive\n02320 38 29 05 05 05 05 29 2d 61 6c 69 65 6d 63 05 70 ... keep-alive\n02320 65 71 75 65 73 74 73 3a 26 33 04 04 43 6f 6e 74 ... request: 1. Conn\n02420 05 66 74 2d 54 73 79 65 3a 26 61 70 78 66 69 63 ... ent-Type : applic\n02420 05 66 74 2d 54 73 79 65 3a 26 61 70 78 66 69 63 ... ent-Type : applic\n02520 2d 75 72 02 05 66 63 0f 64 65 64 6d 68 63 43 6f 6e ... urlenco ded.com\n02720 74 65 66 74 2d 4c 05 05 67 74 68 3a 2b 32 30 05 ... urlenco ded.com\n02820 04 0d 8a 75 66 3d 6d 79 72 74 67 2b 70 77 3d 33 ... ..allow r7o&pw1\n02820 04 0d 8a 75 66 3d 6d 79 72 74 67 2b 70 77 3d 33 ... ..allow r7o&pw1\n02820 04 0d 8a 75 66 3d 6d 79 72 74 67 2b 70 77 3d 33 ... ..allow r7o&pw1

wreshark: warning: eth0: No carrier

Packets: 17 - Dissected: 17 (100.0%)

Profile: Default

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...



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



# More complex systems needed...



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



## Replay attack



## ... 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 :(

ZDNet EDITION: UK

MICROSOFT STORAGE INNOVATION HARDWARE APPLE MORE > NEWSLETTERS ALL WRITERS

## FREAK: Another day, another serious SSL security hole

More than one third of encrypted Websites are open to attack via the FREAK security hole.

The Telegraph

Home Video News World Sport Business Money Comment Culture Travel Life W  
USA Asia China Europe Middle East Australasia Africa South America Central Asia

HOME > NEWS > WORLD NEWS > NORTH AMERICA > USA

### Hacker remotely crashes Jeep from 10 miles away

Security experts warn that more than 470,000 cars made by Fiat Chrysler could be at risk of being attacked by similar means – including those driven in the UK

The Register®  
Biting the hand that feeds IT

### Defects in e-passports allow real-time tracking

This threat brought to you by RFID

threatpost

CATEGORIES FEATURED PODCASTS VIDEOS

### TRIPLE HANDSHAKE ATTACKS TARGET TLS RESUMPTION, RENEGOTIATION

## 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

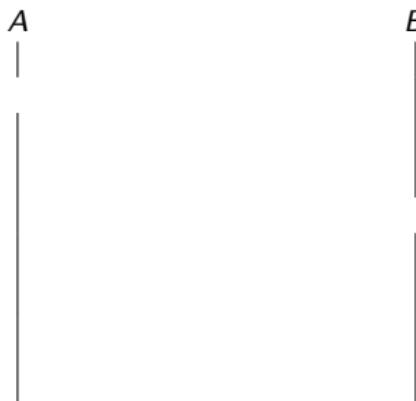
## Example of a logical attack

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



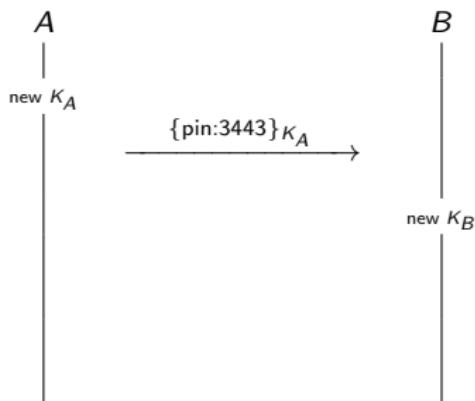
# Example of a logical attack

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



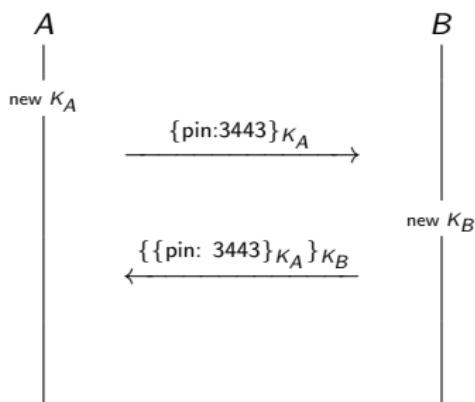
# Example of a logical attack

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



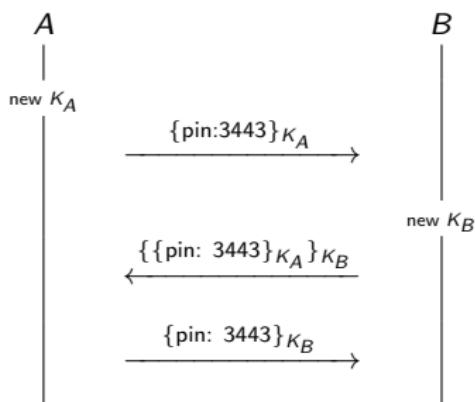
# Example of a logical attack

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

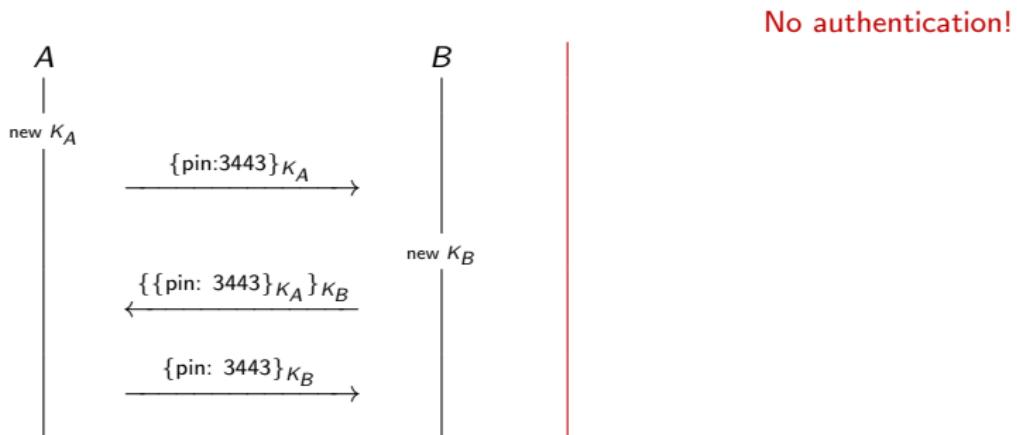
# Example of a logical attack

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

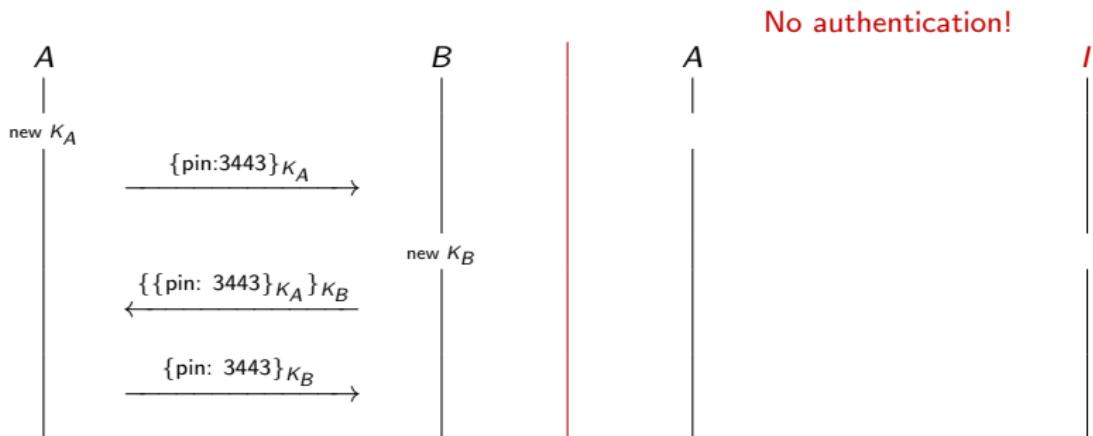
# Example of a logical attack

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

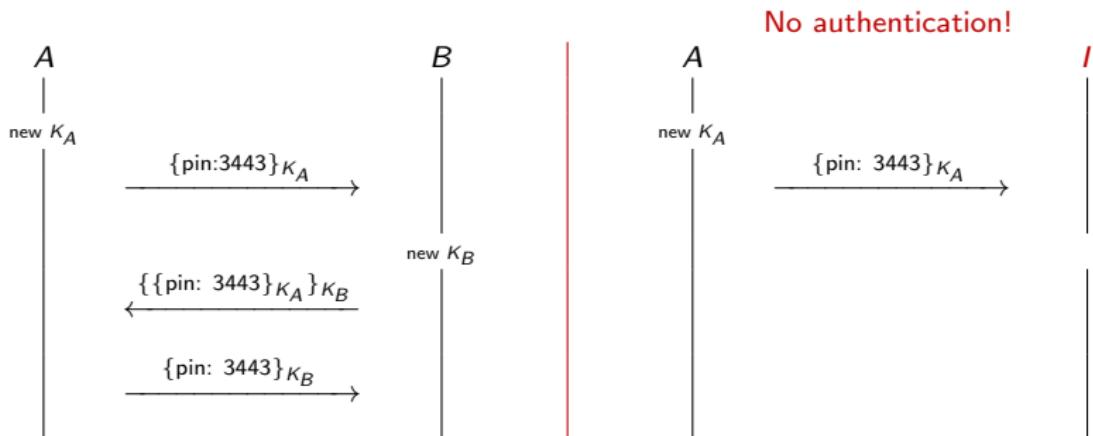
# Example of a logical attack

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

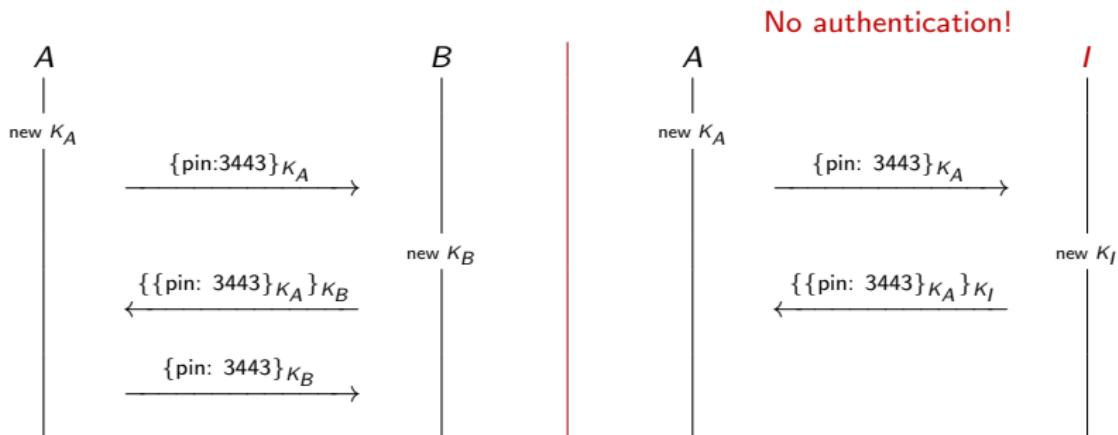
# Example of a logical attack

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

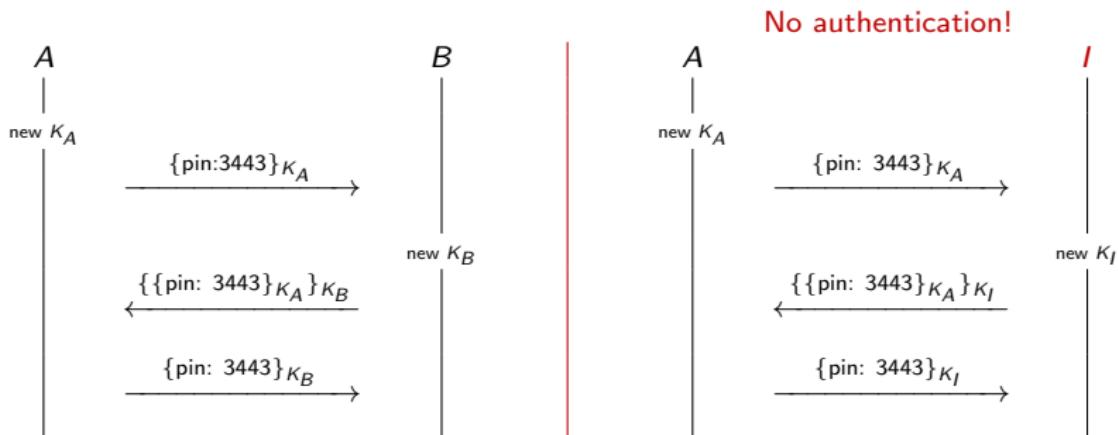
# Example of a logical attack

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 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
- ~~> 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)

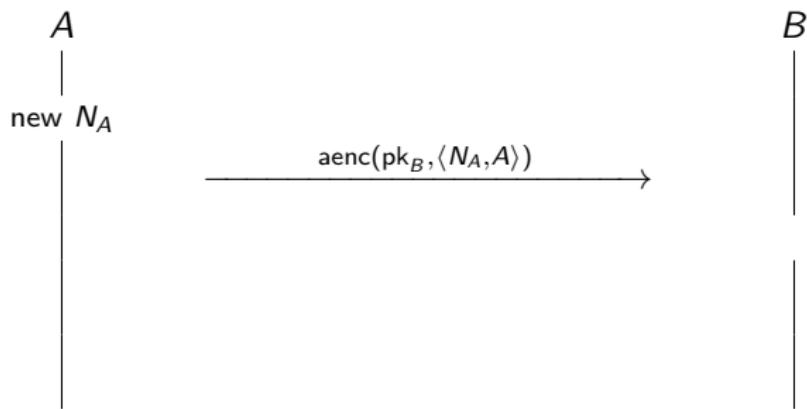
NSPK: authentication and key agreement protocol



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

# Needham-Schroeder Public Key (NSPK)

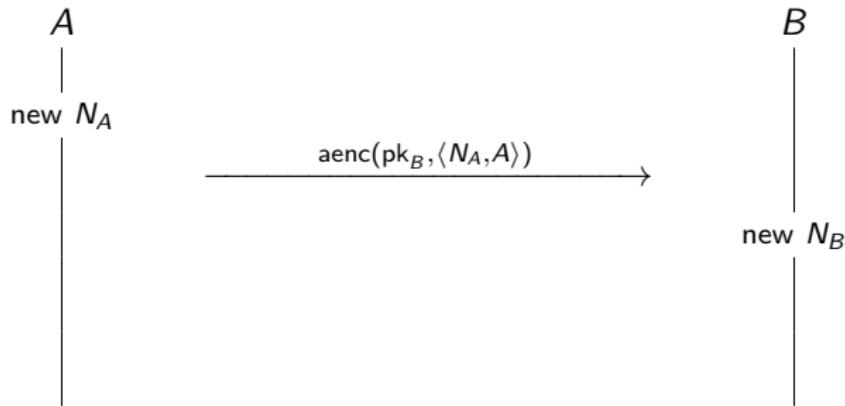
NSPK: authentication and key agreement protocol



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

# Needham-Schroeder Public Key (NSPK)

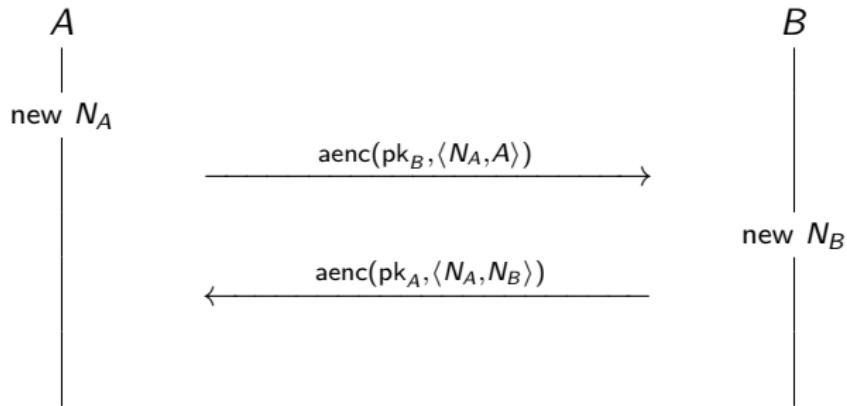
NSPK: authentication and key agreement protocol



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

# Needham-Schroeder Public Key (NSPK)

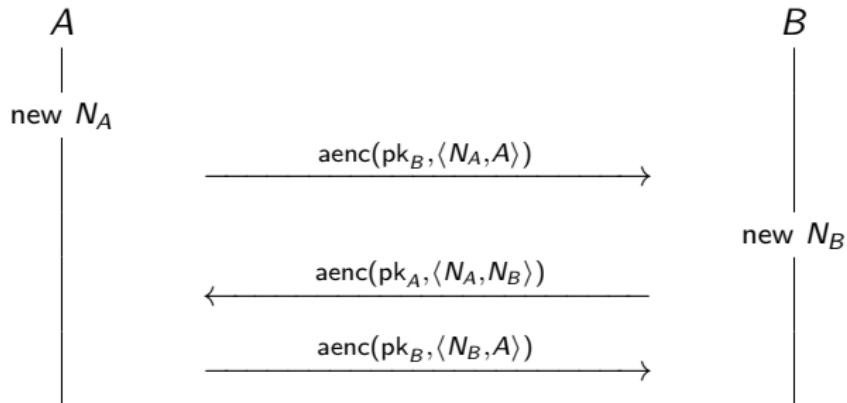
NSPK: authentication and key agreement protocol



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

# Needham-Schroeder Public Key (NSPK)

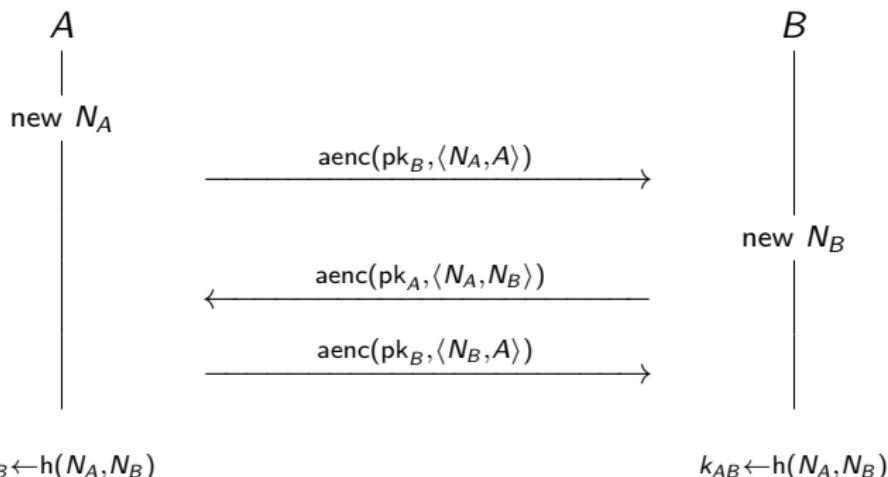
NSPK: authentication and key agreement protocol



[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

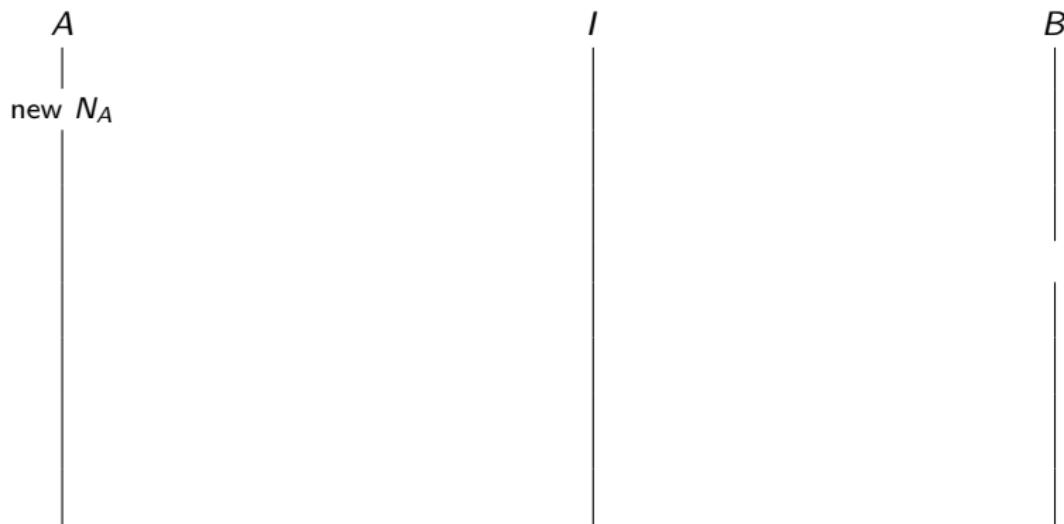
- ▶ **Authentication:** if Alice has completed the protocol, apparently with Bob, then Bob must also have completed the protocol with Alice.
- ▶ **Authentication:** If Bob has completed the protocol, apparently with Alice, then Alice must have completed the protocol with Bob.
- ▶ **Confidentiality:**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

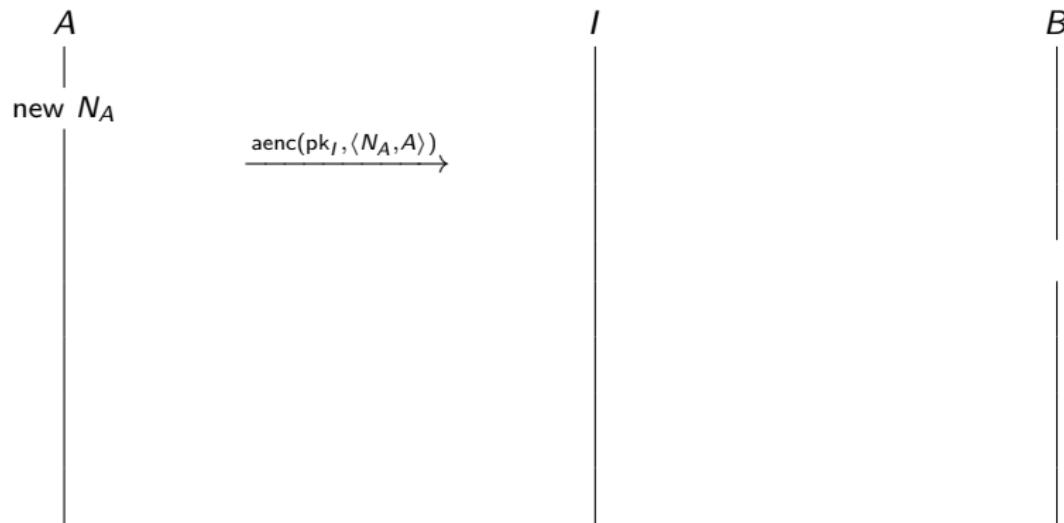
Attack found 17 years after the publication of the NS protocol!!



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

# NSPK: Lowe's attack on authentication

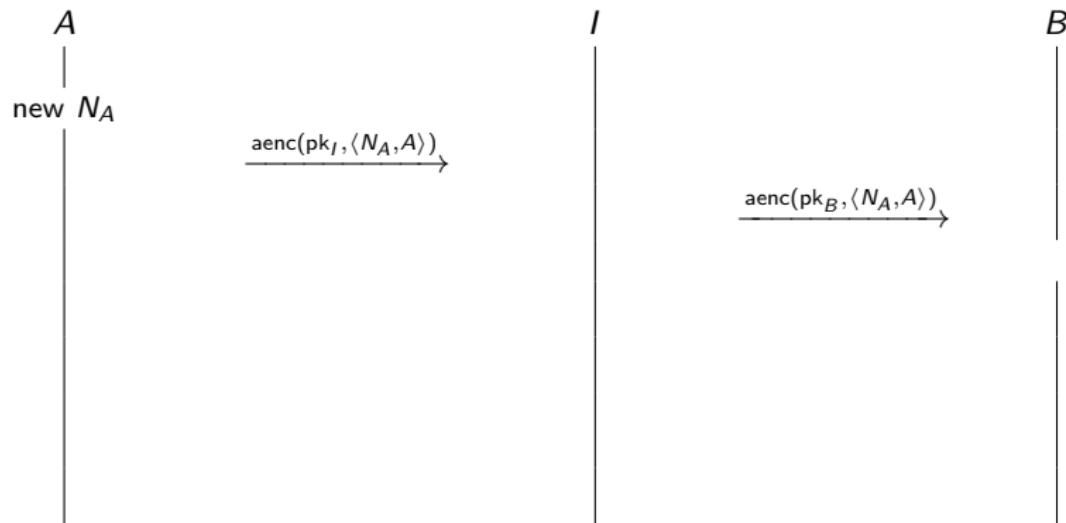
Attack found 17 years after the publication of the NS protocol!!



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

# NSPK: Lowe's attack on authentication

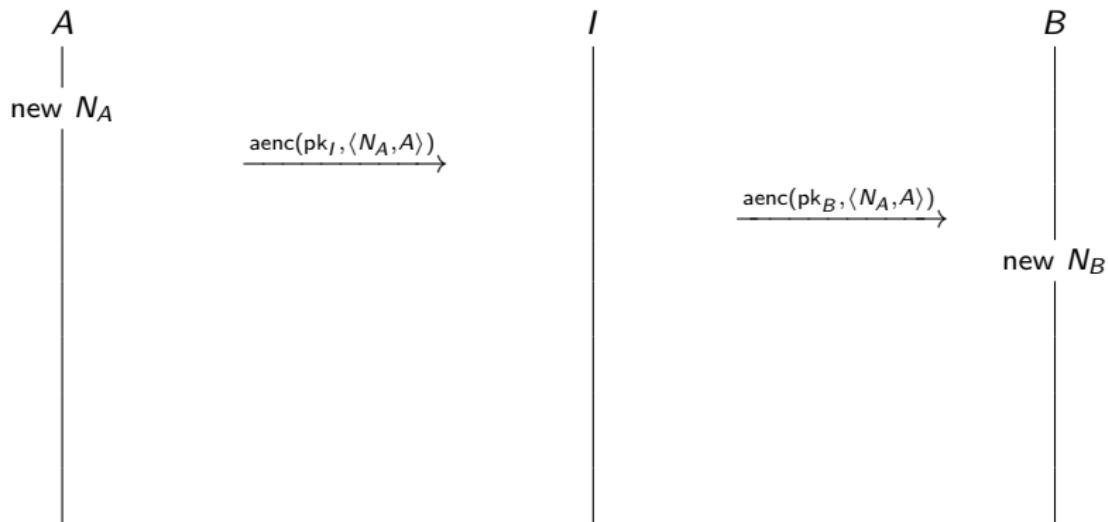
Attack found 17 years after the publication of the NS protocol!!



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

# NSPK: Lowe's attack on authentication

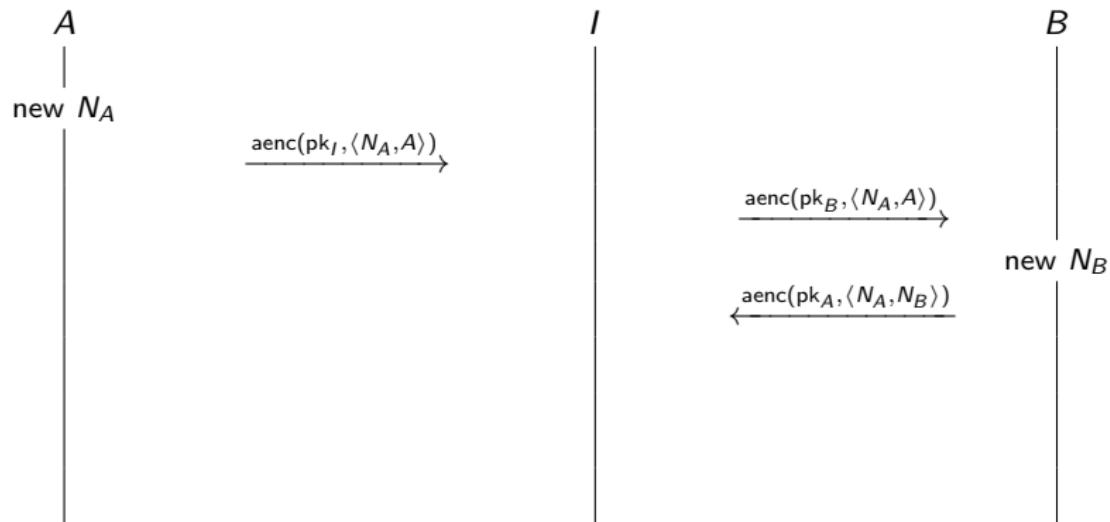
Attack found 17 years after the publication of the NS protocol!!



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

# NSPK: Lowe's attack on authentication

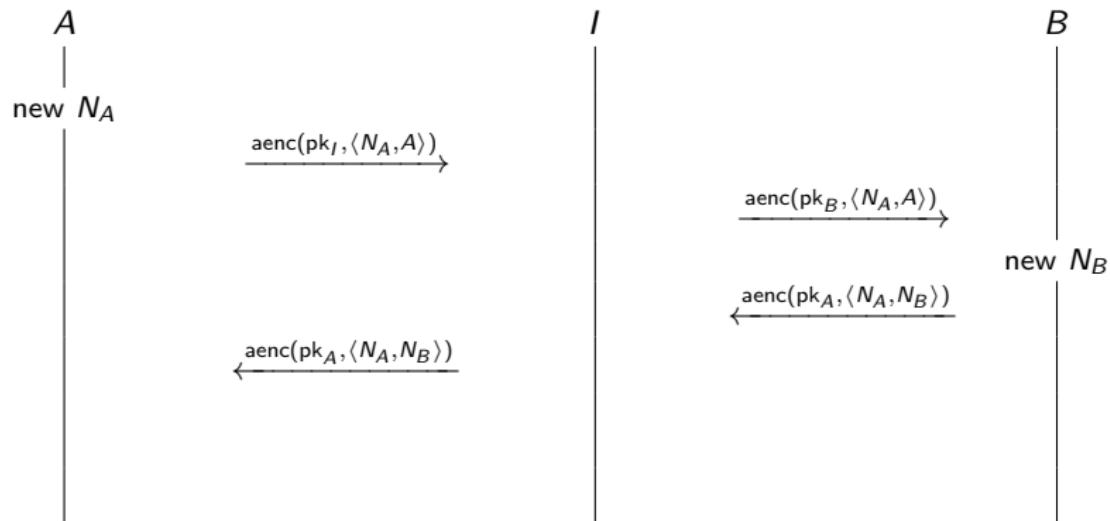
Attack found 17 years after the publication of the NS protocol!!



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

# NSPK: Lowe's attack on authentication

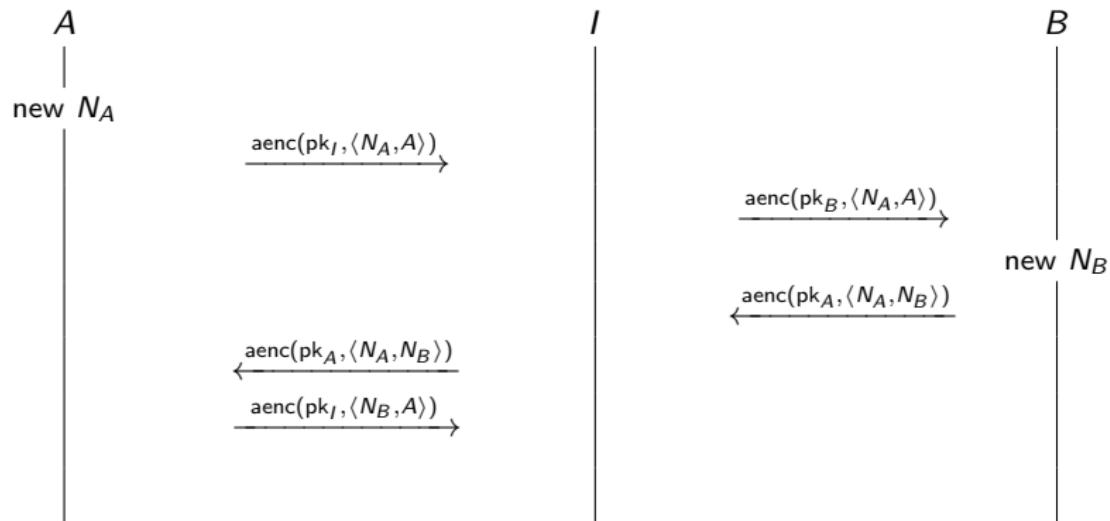
Attack found 17 years after the publication of the NS protocol!!



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

# NSPK: Lowe's attack on authentication

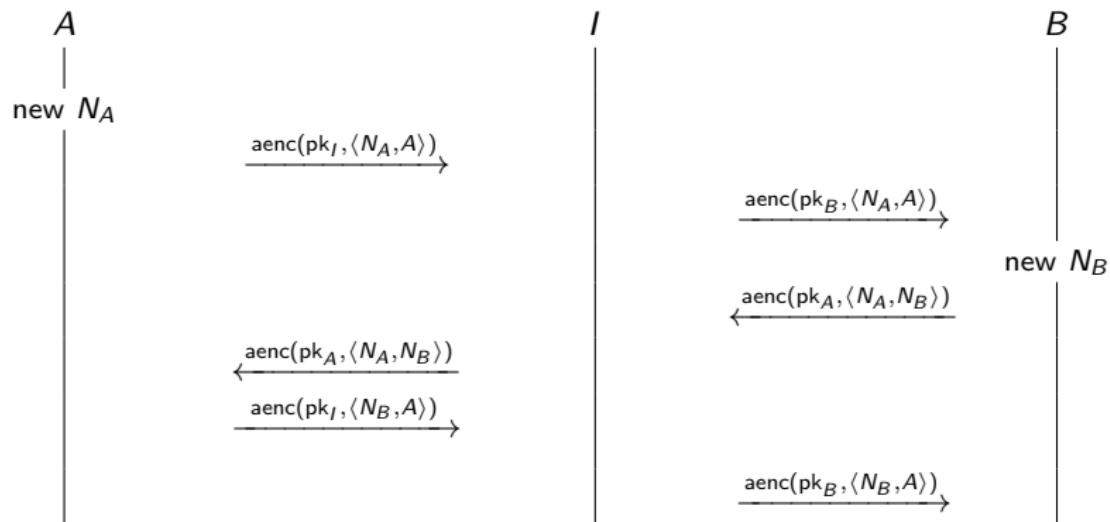
Attack found 17 years after the publication of the NS protocol!!



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

# NSPK: Lowe's attack on authentication

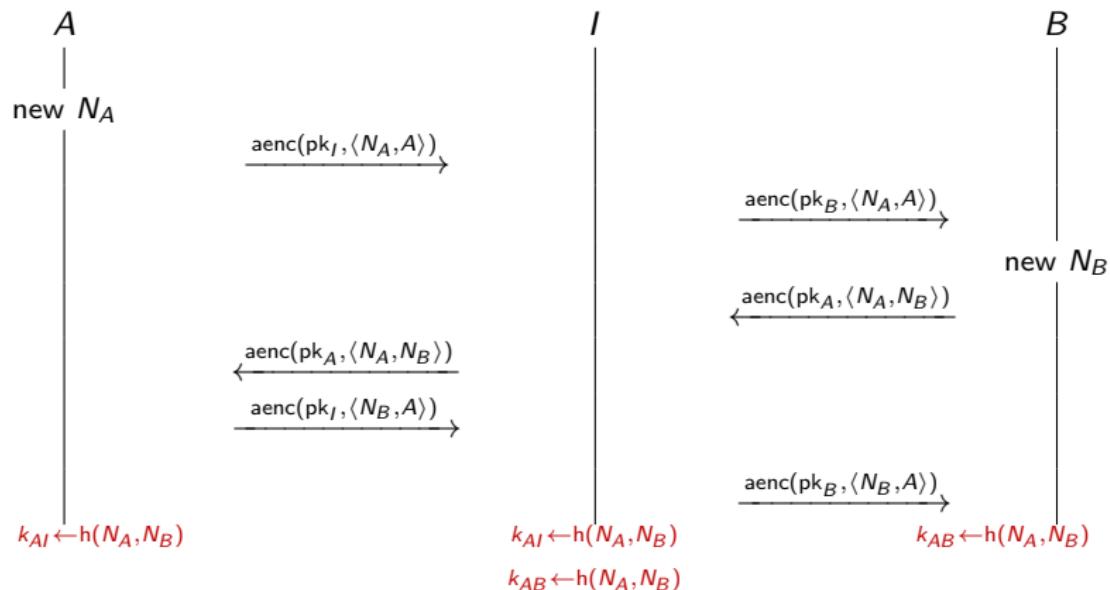
Attack found 17 years after the publication of the NS protocol!!



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

# NSPK: Lowe's attack on authentication

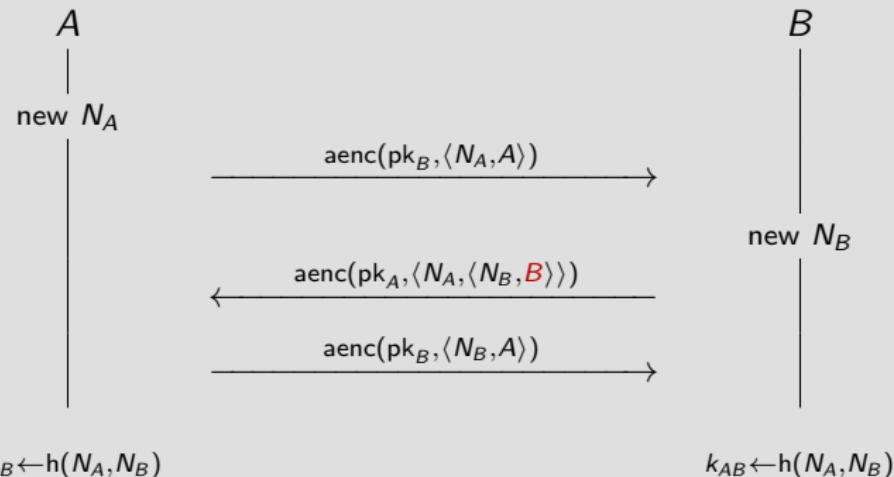
Attack found 17 years after the publication of the NS protocol!!



[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 interfered.

# The Station-to-Station (StS) protocol

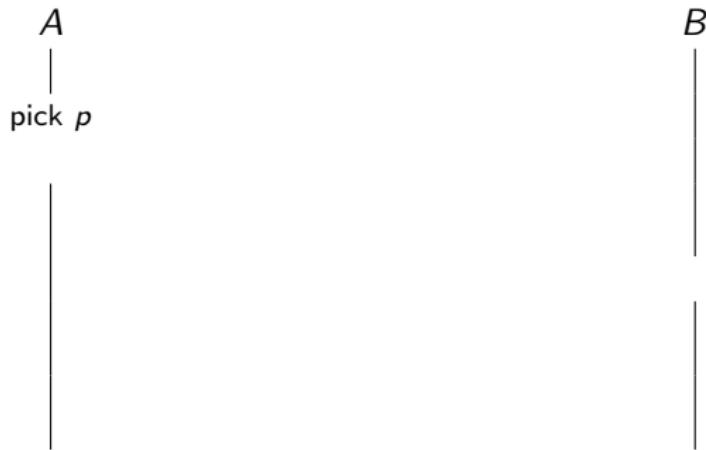
A



B



# The Station-to-Station (StS) protocol



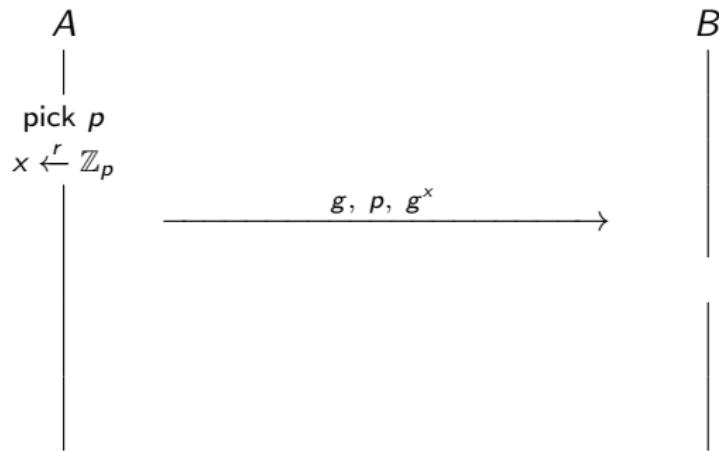
- ▶ where  $p$  is a large prime
- ▶ and  $g$  a generator of  $\mathbb{Z}_p^*$

# The Station-to-Station (StS) protocol



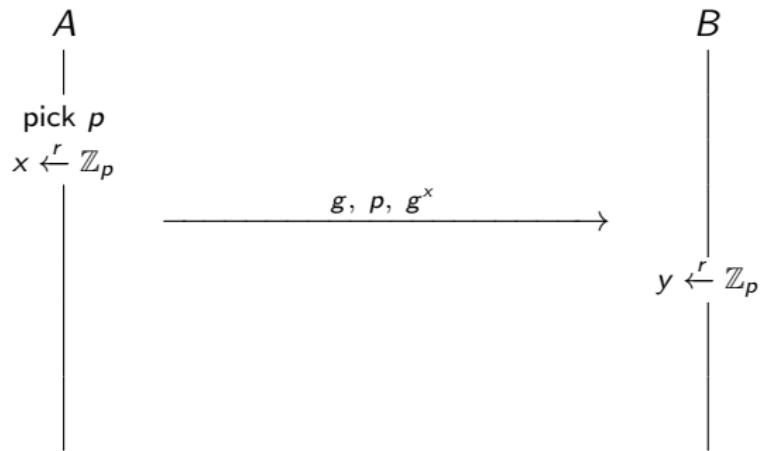
- ▶ where  $p$  is a large prime
- ▶ and  $g$  a generator of  $\mathbb{Z}_p^*$

# The Station-to-Station (StS) protocol



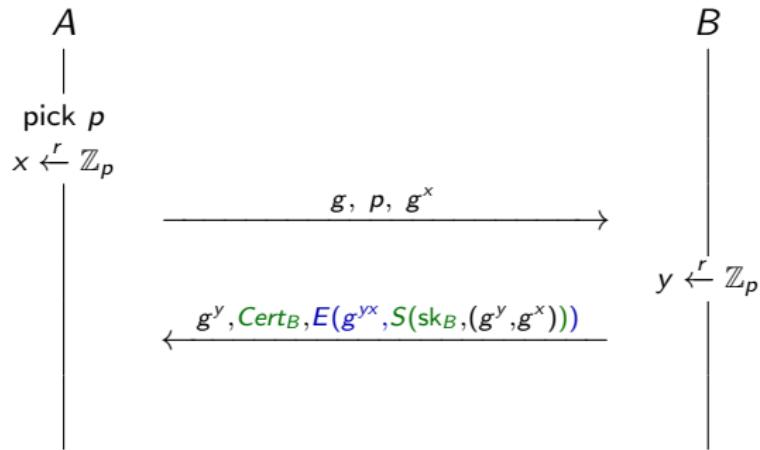
- ▶ where  $p$  is a large prime
- ▶ and  $g$  a generator of  $\mathbb{Z}_p^*$

# The Station-to-Station (StS) protocol



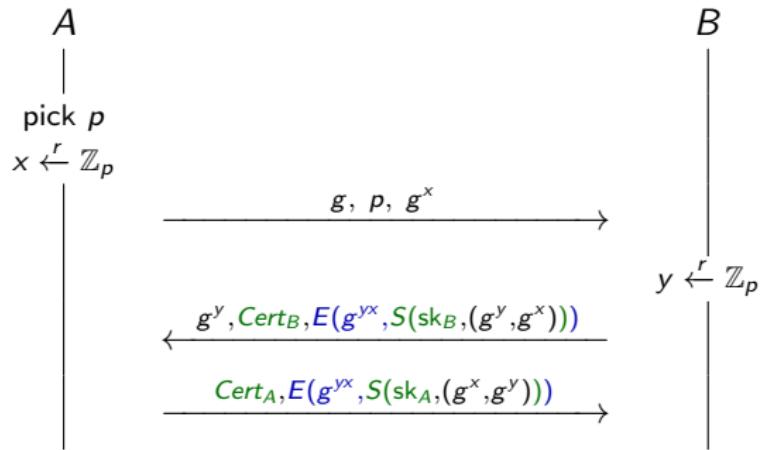
- ▶ where  $p$  is a large prime
- ▶ and  $g$  a generator of  $\mathbb{Z}_p^*$

# The Station-to-Station (StS) protocol



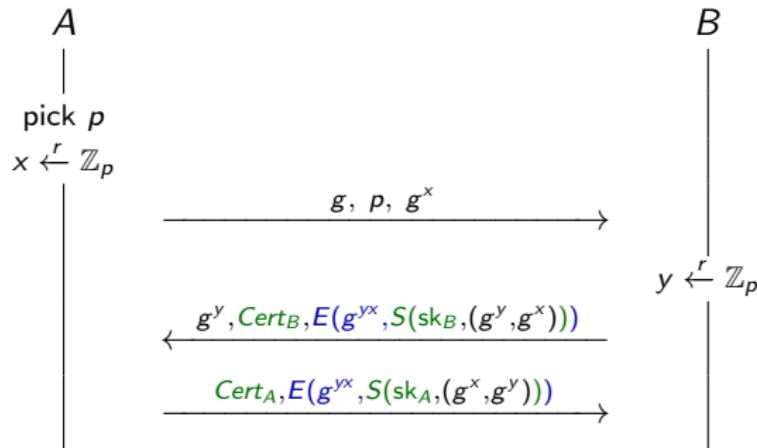
- ▶ where  $p$  is a large prime
- ▶ and  $g$  a generator of  $\mathbb{Z}_p^*$

# 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  $\mathbb{Z}_p^*$

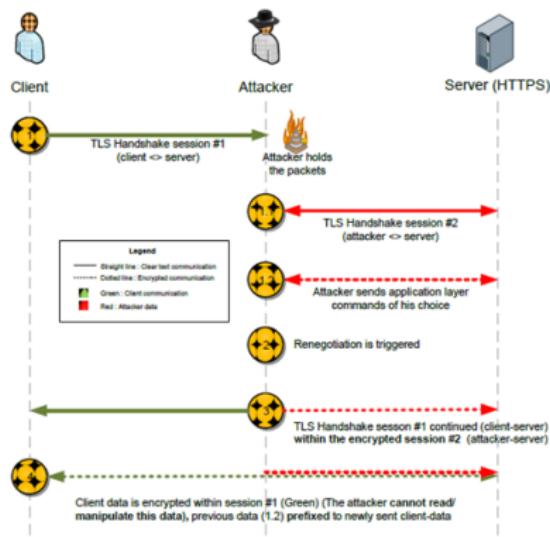
The StS ensures mutual authentication, key agreement, and forward secrecy

# 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 assumption:** 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
```

⇒ 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...
```

⇒ the attacker gets the user name and password of the victim!