Filling in the Gaps:

Modelling Negotiation in the TLS Protocol

QUT

Queensland University of Technology
Brisbane Australia

BENJAMIN DOWLING AND DOUGLAS STEBILA

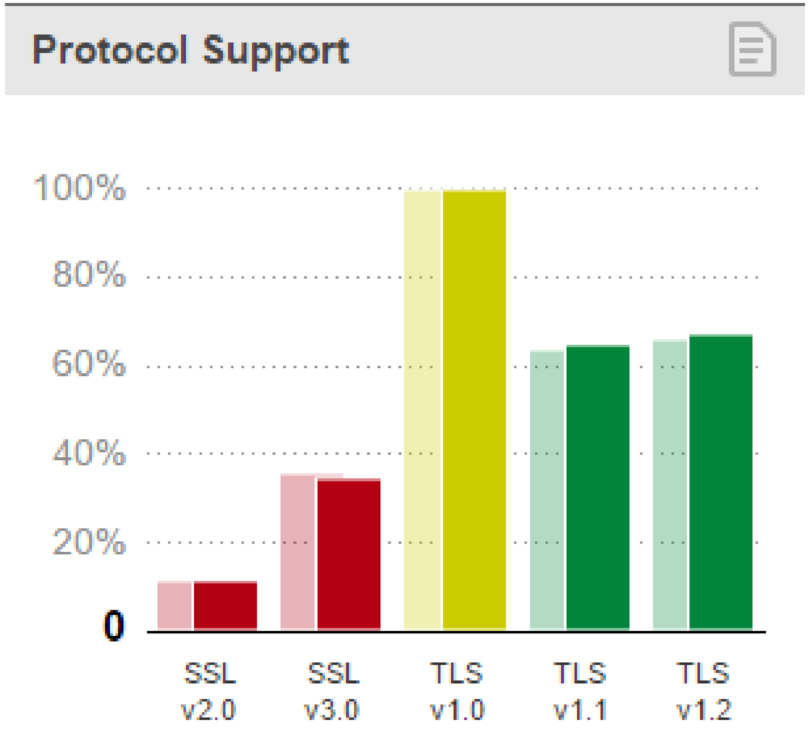
Outline

1. Motivation
2. Negotiation in the TLS Protocol
3. Modelling negotiation in a provable security framework
4. Analysis of TLS ciphersuite and version negotiation
5. Conclusions

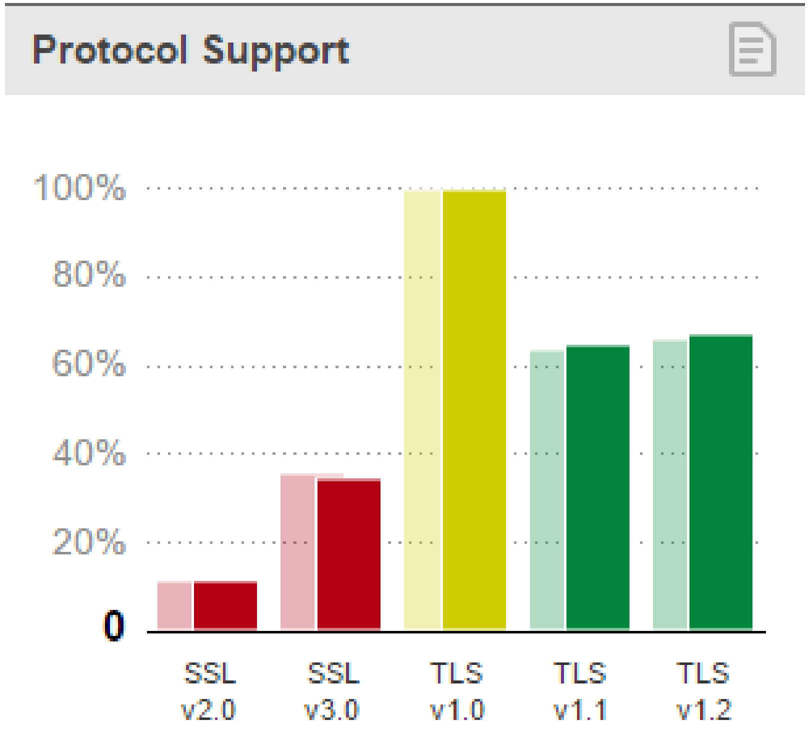
Motivation

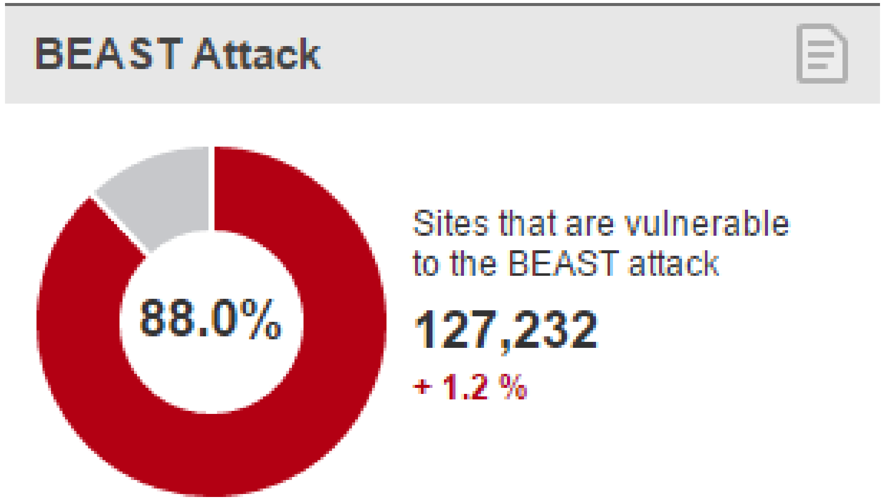
- TLS implementations have complex functionality
- Current analyses' of TLS protocol do not cover all aspects
- Algorithmic agility is desired to increase interoperability
- However, interoperability can affect security

Motivation

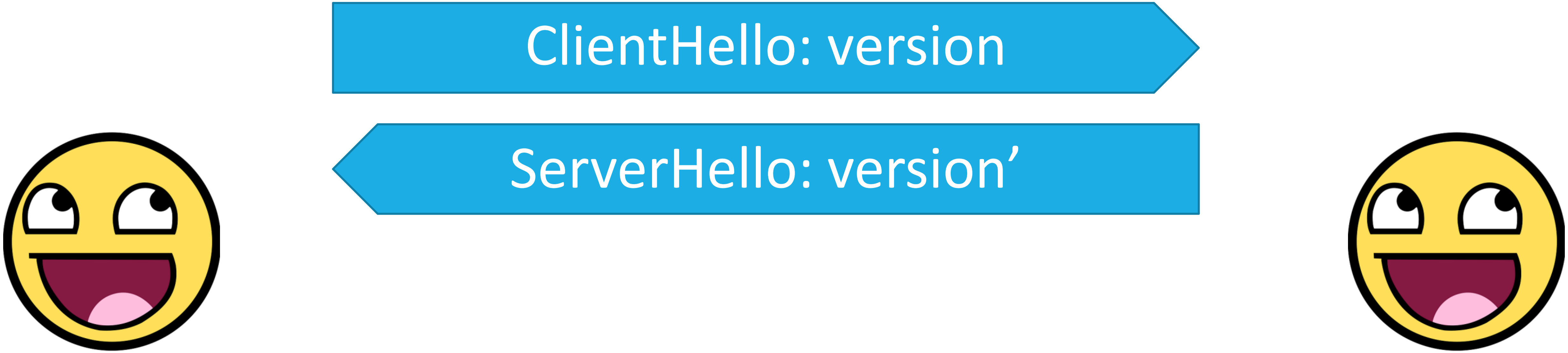

Protocol Support
100%
80%
60%
40%
20%
0
SSL v2.0
SSL v3.0
TLS v1.0
TLS v1.1
TLS v1.2

Motivation


Protocol Support
100%
80%
60%
40%
20%
0
SSL v2.0
SSL v3.0
TLS v1.0
TLS v1.1
TLS v1.2


BEAST Attack
88.0%
Sites that are vulnerable to the BEAST attack
127,232
+ 1.2 %

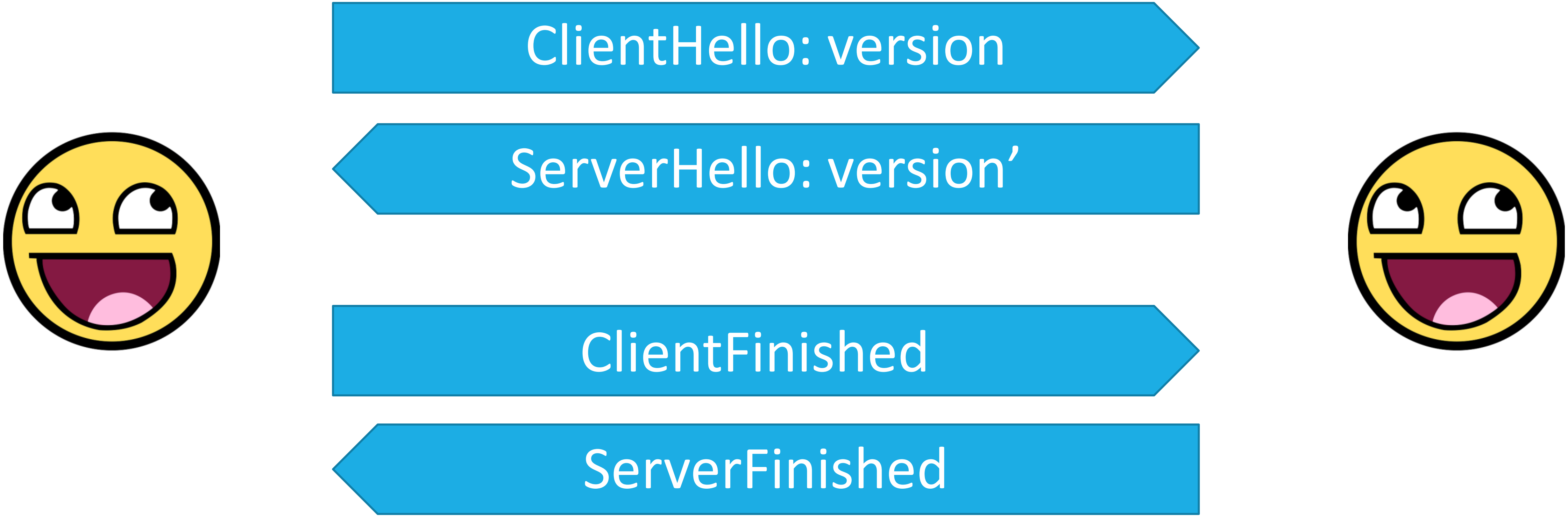
Version Negotiation



ClientHello: version

ServerHello: version’

Version Negotiation



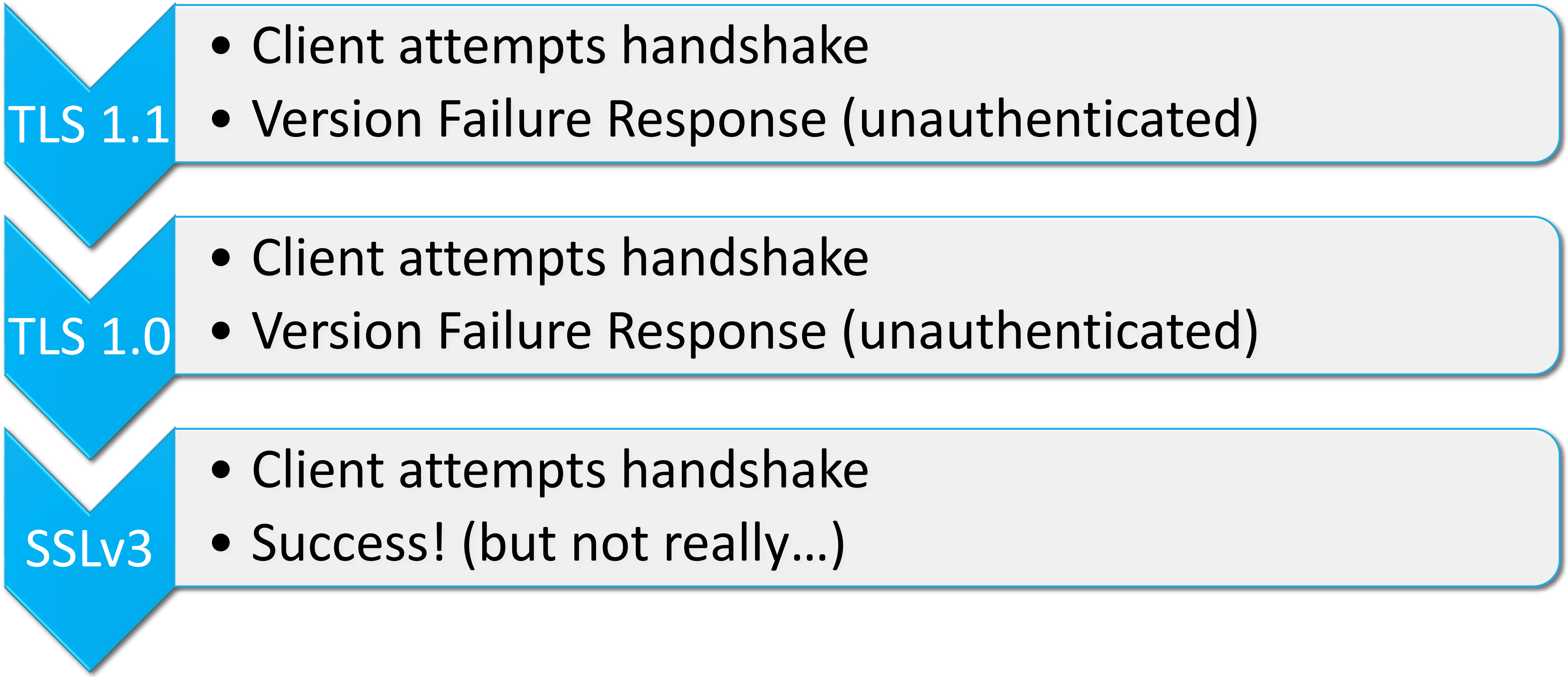
ClientHello: version

ServerHello: version’

ClientFinished

ServerFinished

Version Downgrade Dance



TLS 1.1

- Client attempts handshake
- Version Failure Response (unauthenticated)

TLS 1.0

- Client attempts handshake
- Version Failure Response (unauthenticated)

SSLv3

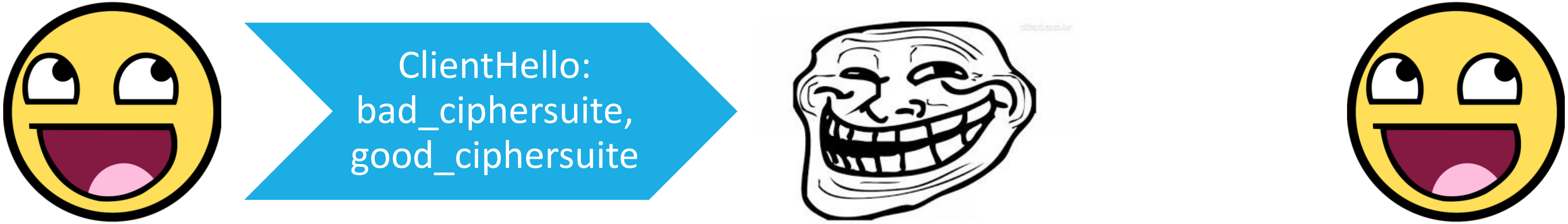
- Client attempts handshake
- Success! (but not really…)

Version Downgrade Attacks

- POODLE attack (Möller, Duong and Kotowicz, 2014)
  - Utilizes downgrade to SSLv3

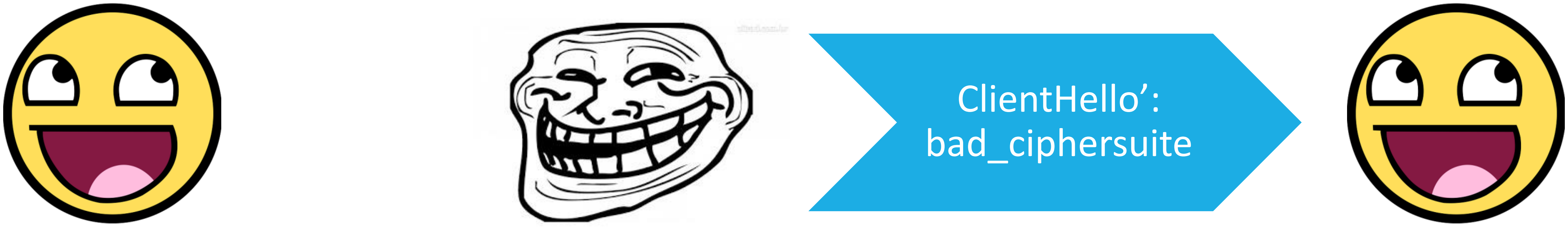
- Signalling Cipher Suite Value (Möller and Langley, 2015)
  - TLS extension to prevent downgrade attacks

Ciphersuite Downgrade Attacks



ClientHello:
bad_ciphersuite,
good_ciphersuite

Ciphersuite Downgrade Attacks

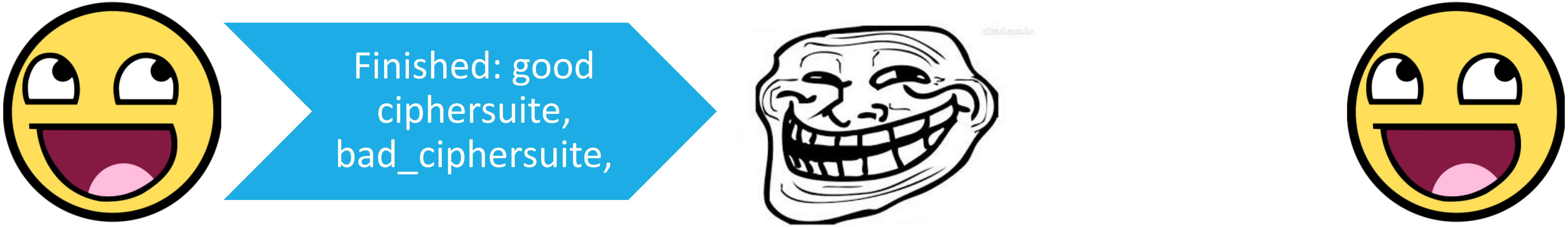


ClientHello’:
bad_ciphersuite

Ciphersuite Downgrade Attacks

ServerHello:
bad_ciphersuite

Ciphersuite Downgrade Attacks

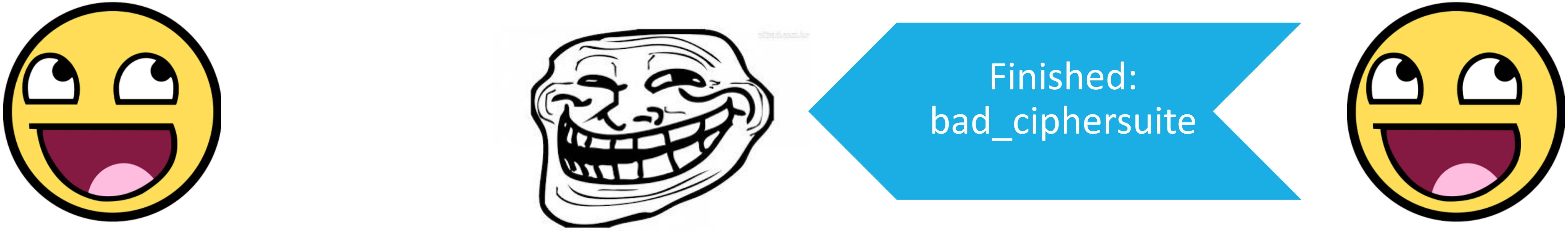


Finished: good ciphersuite, bad_ciphersuite,

Ciphersuite Downgrade Attacks

Finished’:
bad_ciphersuite

Ciphersuite Downgrade Attacks



Finished:
bad_ciphersuite

Ciphersuite Downgrade Attacks

Finished’: good ciphersuite, bad_ciphersuite

Ciphersuite Downgrade Attacks

- FREAK attack – (Beurdouche, Bhargavan, Delignat-Lavaud, Fournet, Kohlweiss, Pironti, Strub, Zinzindohoue, Zanella-Béguelin; 2015)
  - Implementation errors allow the negotiation of Export-RSA despite no indicated support

- Logjam Attack – (Adrian, Bhargavan, Durumeric, Gaudry, Green, Halderman, Heninger, Springall, Thomé, Valenta, VanderSloot, Wustrow, Zanella-Beguelin, and Zimmermann; 2015)
  - Protocol logic misinterprets export-DHE shares as “normal” DHE shares

Observations

- Clearly negotiation from a family of protocols can affect security of the protocol as a whole

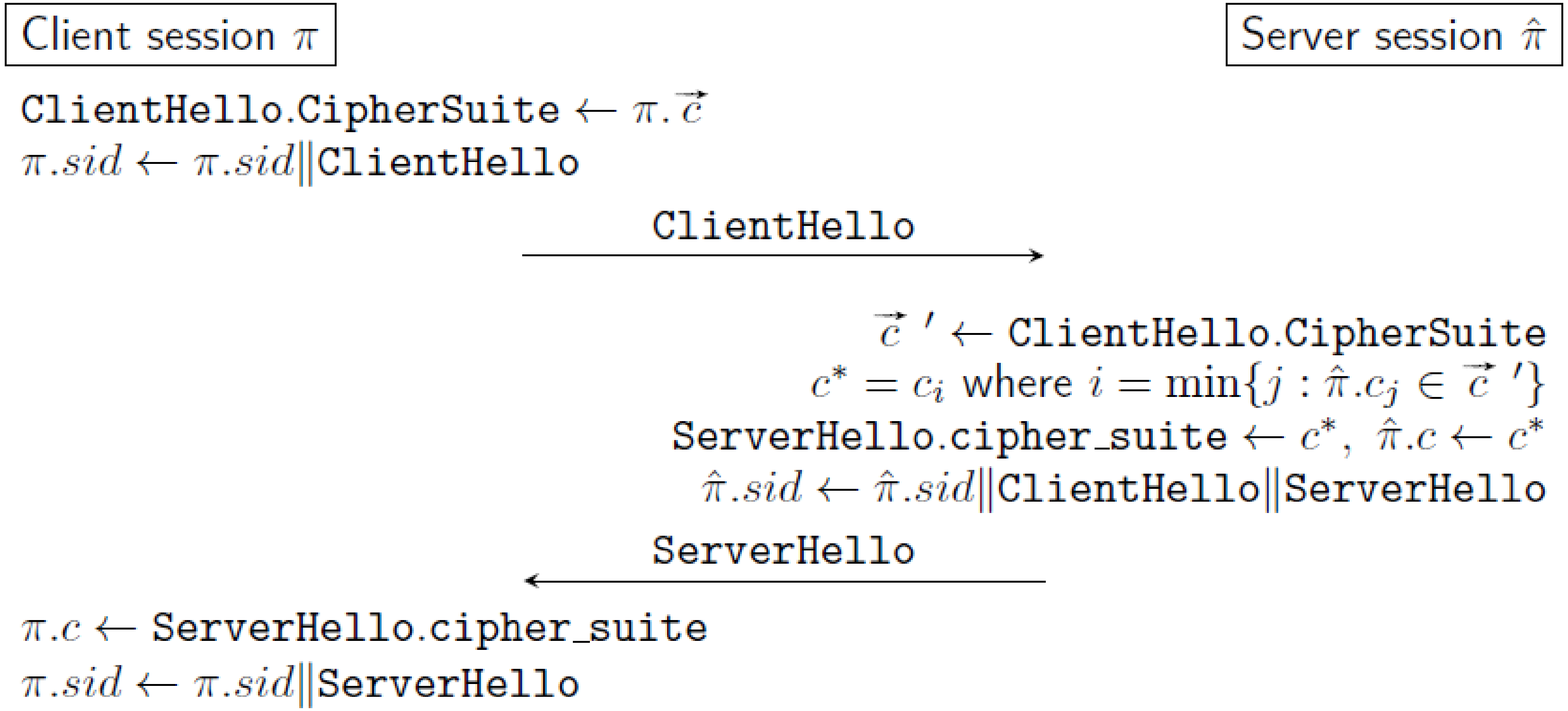
- What can we say about the security of the collection of protocols?

Talking ‘bout negotiation

- Treat the handshake as two phases:
  - A negotiation phase: common to all handshake runs
  - A sub-protocol phase: uses negotiated values to execute key-exchange/authentication, etc.

- “Optimal” negotiation:
  - Both parties have ordered list of elements/preferences
  - Also have an “optimality function”
  - Negotiation is optimal if they output same value and it’s the output of *opt(list, list’)*

Ciphersuite Negotiation Phase



Client session π

Server session $\hat{\pi}$

$\texttt{ClientHello.CipherSuite} \leftarrow \pi.\vec{c}$

$\pi.sid \leftarrow \pi.sid \| \texttt{ClientHello}$

$\xrightarrow{\texttt{ClientHello}}$

$\vec{c}\,' \leftarrow \texttt{ClientHello.CipherSuite}$

$c^* = c_i$ where $i = \min\{j : \hat{\pi}.c_j \in \vec{c}\,'\}$

$\texttt{ServerHello.cipher\_suite} \leftarrow c^*, \ \hat{\pi}.c \leftarrow c^*$

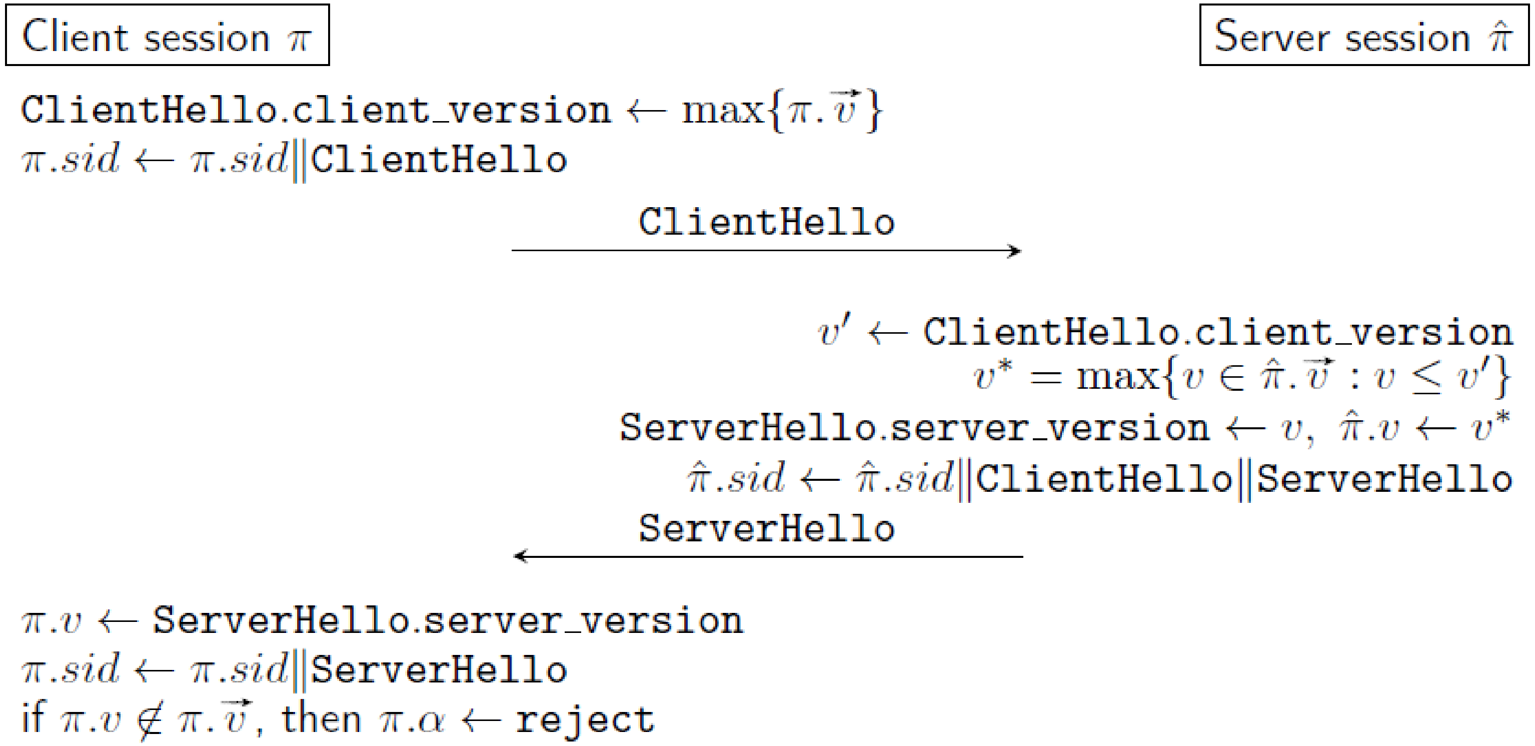
$\hat{\pi}.sid \leftarrow \hat{\pi}.sid \| \texttt{ClientHello} \| \texttt{ServerHello}$

$\xleftarrow{\texttt{ServerHello}}$

$\pi.c \leftarrow \texttt{ServerHello.cipher\_suite}$

$\pi.sid \leftarrow \pi.sid \| \texttt{ServerHello}$

Version Negotiation Phase



Client session π

Server session $\hat{\pi}$

$\texttt{ClientHello.client\_version} \leftarrow \max\{\pi.\vec{v}\}$

$\pi.sid \leftarrow \pi.sid \| \texttt{ClientHello}$

$\xrightarrow{\texttt{ClientHello}}$

$v' \leftarrow \texttt{ClientHello.client\_version}$

$v^* = \max\{v \in \hat{\pi}.\vec{v} : v \leq v'\}$

$\texttt{ServerHello.server\_version} \leftarrow v,\ \hat{\pi}.v \leftarrow v^*$

$\hat{\pi}.sid \leftarrow \hat{\pi}.sid \| \texttt{ClientHello} \| \texttt{ServerHello}$

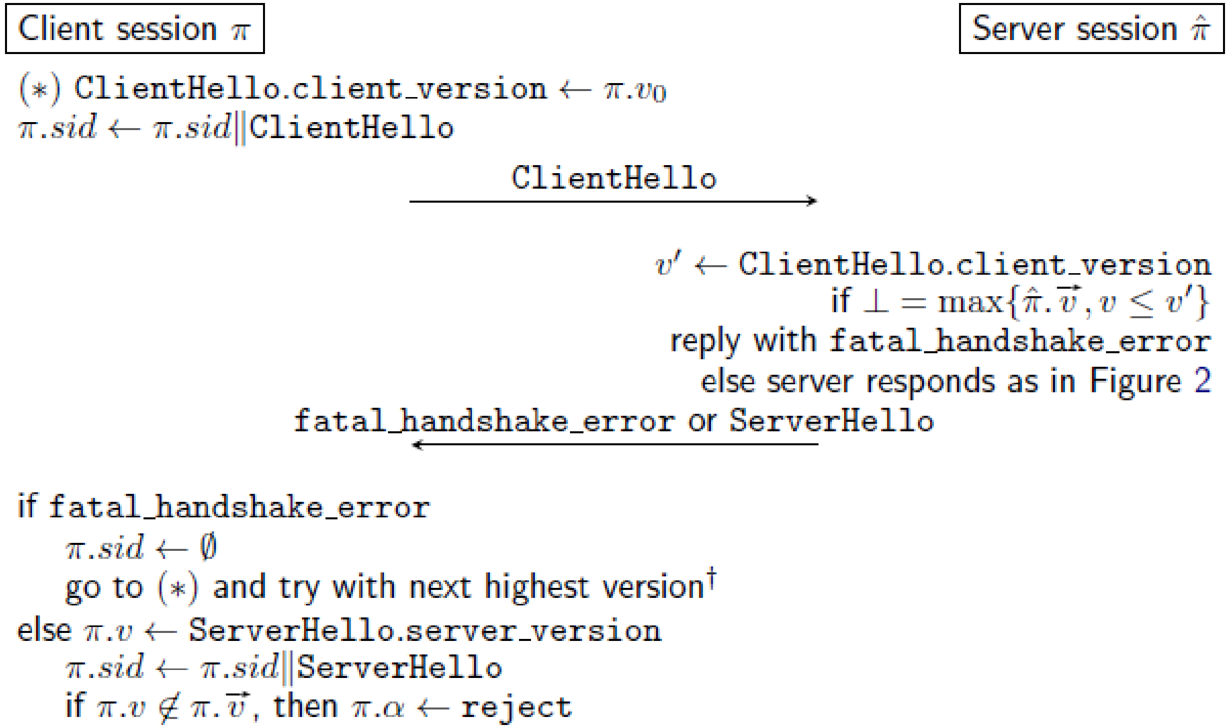
$\xleftarrow{\texttt{ServerHello}}$

$\pi.v \leftarrow \texttt{ServerHello.server\_version}$

$\pi.sid \leftarrow \pi.sid \| \texttt{ServerHello}$

if $\pi.v \notin \pi.\vec{v}$, then $\pi.\alpha \leftarrow \texttt{reject}$

Version Negotiation Phase - Fallback



Client session π

Server session $\hat{\pi}$

```
(∗) ClientHello.client_version ← $\pi.v_0$
$\pi.sid \leftarrow \pi.sid \|$ClientHello
```

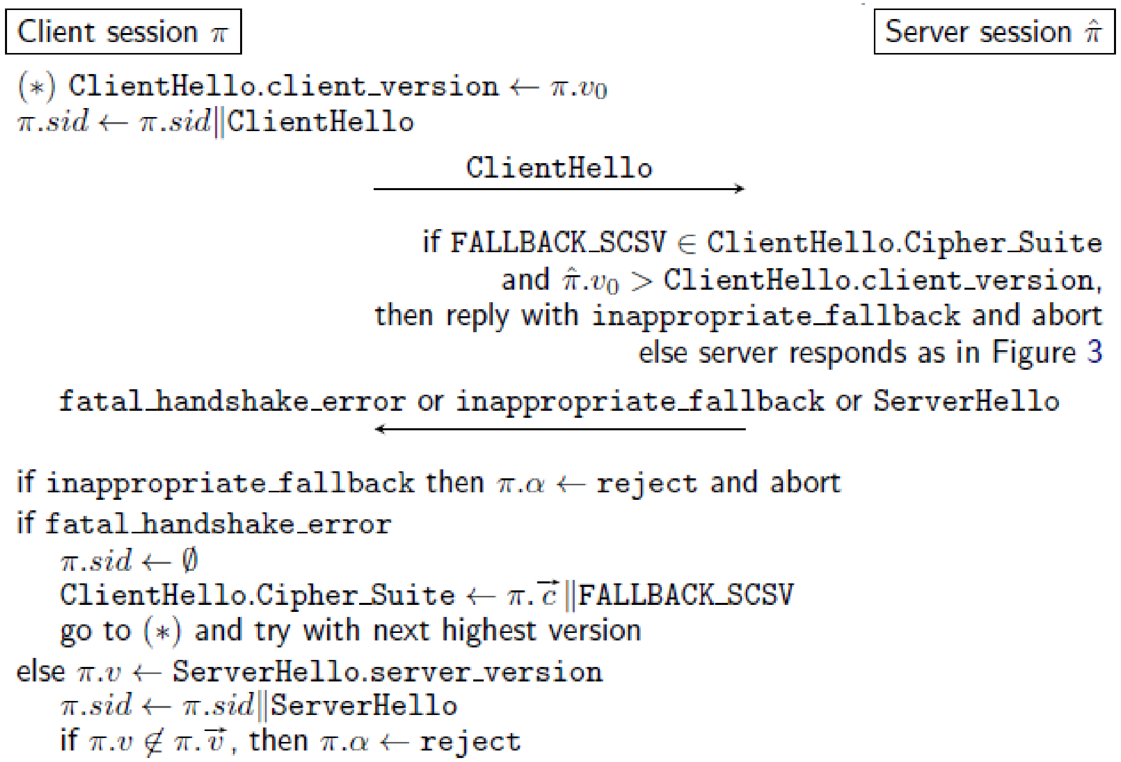
ClientHello $\longrightarrow$

```
$v' \leftarrow$ ClientHello.client_version
if $\bot = \max\{\hat{\pi}.\vec{v}, v \le v'\}$
reply with fatal_handshake_error
else server responds as in Figure 2
```

$\longleftarrow$ fatal_handshake_error or ServerHello

```
if fatal_handshake_error
    $\pi.sid \leftarrow \emptyset$
    go to (∗) and try with next highest version$^\dagger$
else $\pi.v \leftarrow$ ServerHello.server_version
    $\pi.sid \leftarrow \pi.sid \|$ServerHello
    if $\pi.v \notin \pi.\vec{v}$, then $\pi.\alpha \leftarrow$ reject
```

Version Negotiation Phase - SCSV



Client session π

Server session $\hat{\pi}$

$(*)$ ClientHello.client_version $\leftarrow \pi.v_0$

$\pi.sid \leftarrow \pi.sid \| $ ClientHello

$\xrightarrow{\text{ClientHello}}$

if FALLBACK_SCSV $\in$ ClientHello.Cipher_Suite

and $\hat{\pi}.v_0 >$ ClientHello.client_version,

then reply with inappropriate_fallback and abort

else server responds as in Figure 3

$\xleftarrow{\text{fatal\_handshake\_error or inappropriate\_fallback or ServerHello}}$

```
if inappropriate_fallback then $\pi.\alpha \leftarrow$ reject and abort
if fatal_handshake_error
    $\pi.sid \leftarrow \emptyset$
    ClientHello.Cipher_Suite $\leftarrow \pi.\vec{c} \|$FALLBACK_SCSV
    go to $(*)$ and try with next highest version
else $\pi.v \leftarrow$ ServerHello.server_version
    $\pi.sid \leftarrow \pi.sid \|$ServerHello
    if $\pi.v \notin \pi.\vec{v}$, then $\pi.\alpha \leftarrow$ reject
```

Using previous results

- Can see from downgrade attacks that security of the negotiation relates to the authentication of transcript

- Negotiation-Authentication Theorem:
  - Condition 1: All Negotiation Phase messages are in the session identifier
  - Condition 2: If no modification of messages, negotiation always “optimal”
    - Then:

$$\mathrm{Adv}^{\mathsf{neg},\omega}_{\mathrm{NP}\|\overrightarrow{\mathrm{SP}},n}(\mathcal{A}) = \mathrm{Adv}^{\mathsf{acce\text{-}auth}}_{\mathrm{NP}\|\mathrm{SP}_n}(\mathcal{A})$$

Ciphersuite Negotiation “secure”

- 1. All negotiation messages contained in transcript
- 2. Ciphersuite negotiation optimal without active adversary

- If all ciphersuites result in secure authentication properties
  then negotiating to any given ciphersuite is secure

Version Negotiation (no fallback) “secure”

- 1. All negotiation messages contained in transcript
- 2. Version negotiation optimal without active adversary

- If all versions result in secure authentication properties

  then negotiating to any given version is secure

Version Negotiation (w/ fallback) “secure”

- 1. All negotiation messages contained in transcript?

Version Negotiation (w/ fallback) insecure

- 1. All negotiation messages contained in transcript?

NOOOOOOOOOOOOOOOOOOOOOOOOO

Version Negotiation (w/ fallback) insecure

- 1. All negotiation messages contained in transcript?

NOOOOOOOOOOOOOOOOOOOOOOOOO

- Negotiation occurs across multiple handshakes, session identifier is only the transcript of the most recent handshake

Version Negotiation (w/ SCSV) “secure”

- 1. All negotiation messages contained in transcript?

Version Negotiation (w/ SCSV) “secure”

- 1. All negotiation messages contained in transcript?

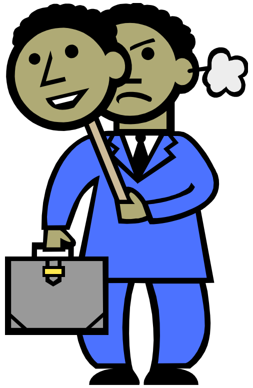
  Nope!

- Can prove security more directly

Version Negotiation (w/ SCSV) “secure”

Adversary

Version-SCSV

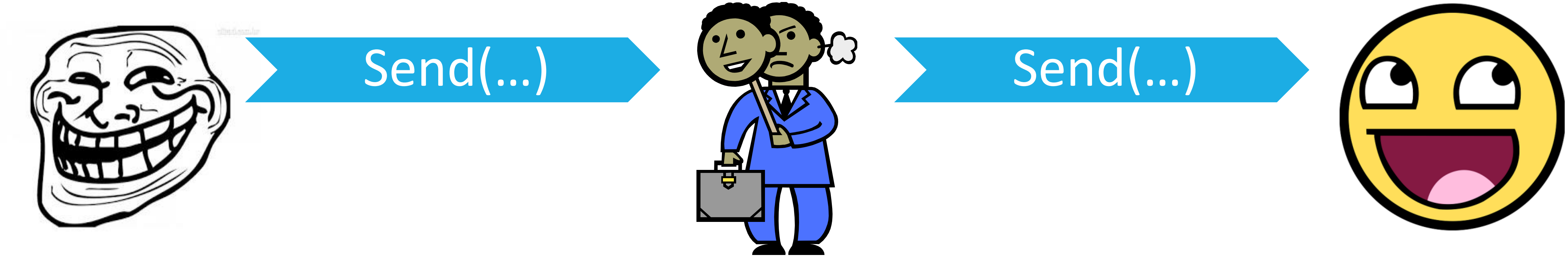


Simulator

Challenger

Version-No-Fallback

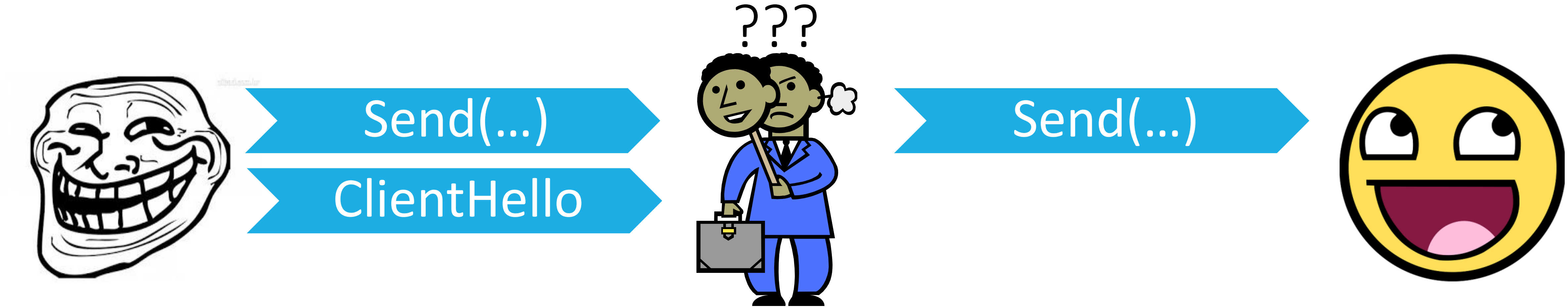
Version Negotiation (w/ SCSV) “secure”



Send(…)

Send(…)

Version Negotiation (w/ SCSV) “secure”



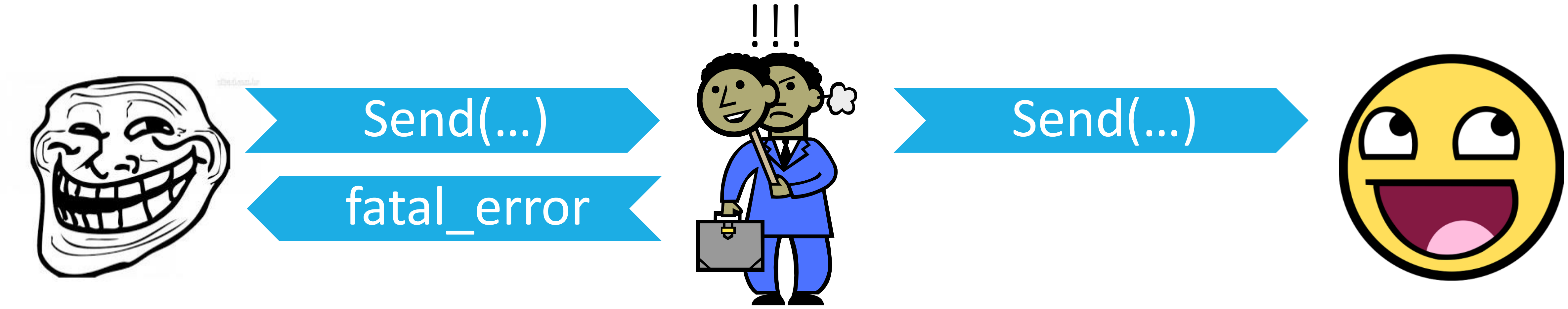
Send(…)

ClientHello

???

Send(…)

Version Negotiation (w/ SCSV) “secure”

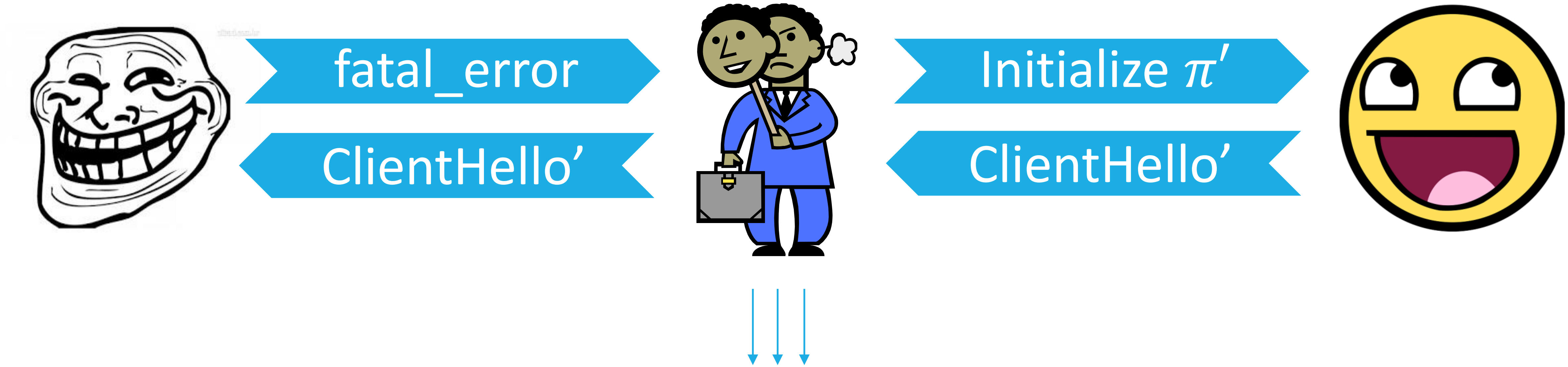


Send(…)

fatal_error

Send(…)

Version Negotiation (w/ SCSV) “secure”



fatal_error

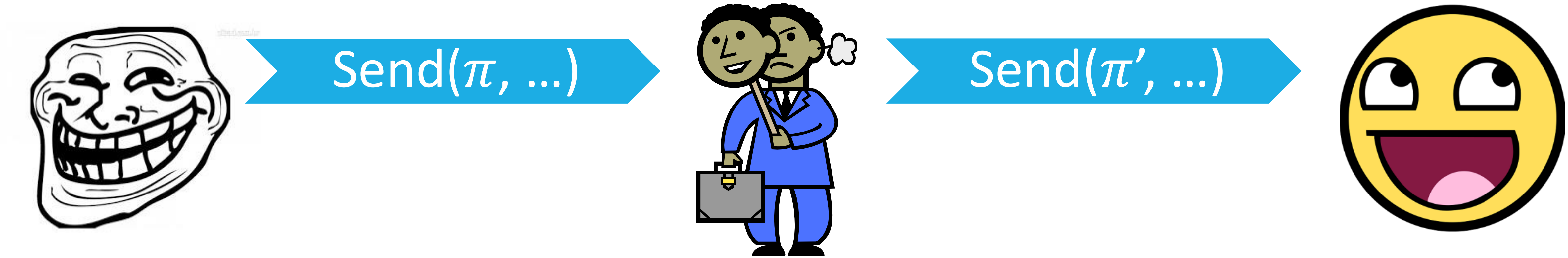
ClientHello’

Initialize π'

ClientHello’

Fallback List: Session π : π'

Version Negotiation (w/ SCSV) “secure”

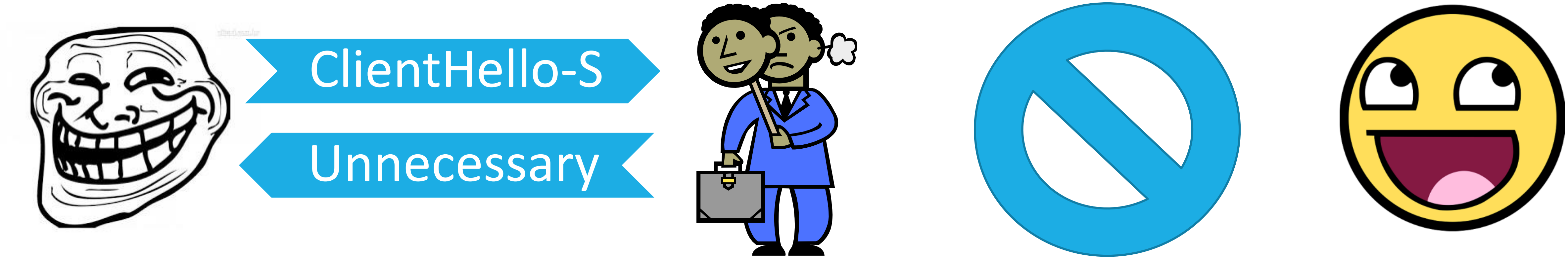


Send(π, …)

Send(π’, …)

Fallback List: Session π : π’

Version Negotiation (w/ SCSV) “secure”



ClientHello-S

Unnecessary

Fallback List: Session π : π'

Version Negotiation (w/ SCSV) “secure”

- 2 cases exist if successful adversary:
  - Breaking a session on the Fallback-List
  - Breaking a session not on the Fallback List

- Each case bounds the success of the adversary with the success of breaking ACCE authentication

Version Negotiation (w/ SCSV) “secure”

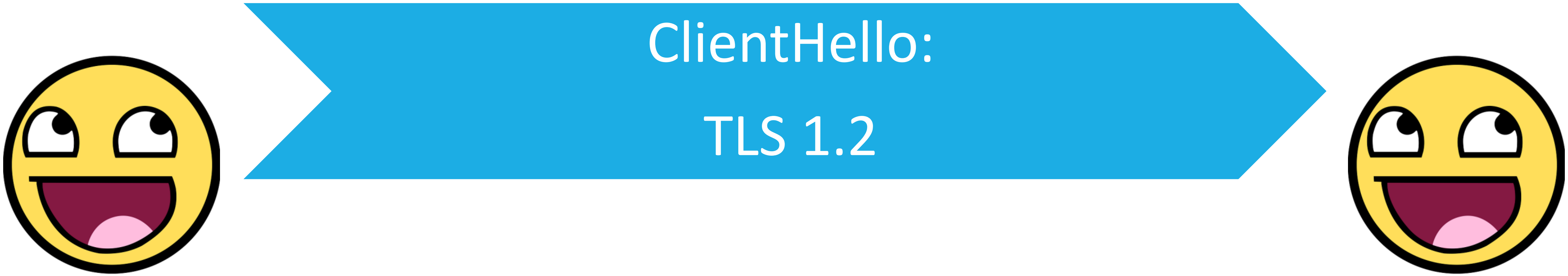
- 1. All negotiation messages contained in transcript?

  Nope!

- Can prove security more directly

- HOWEVER: Non-contiguous support of TLS version (i.e. supporting 1.2 and 1.0 but not 1.1) can break version negotiation with SCSV

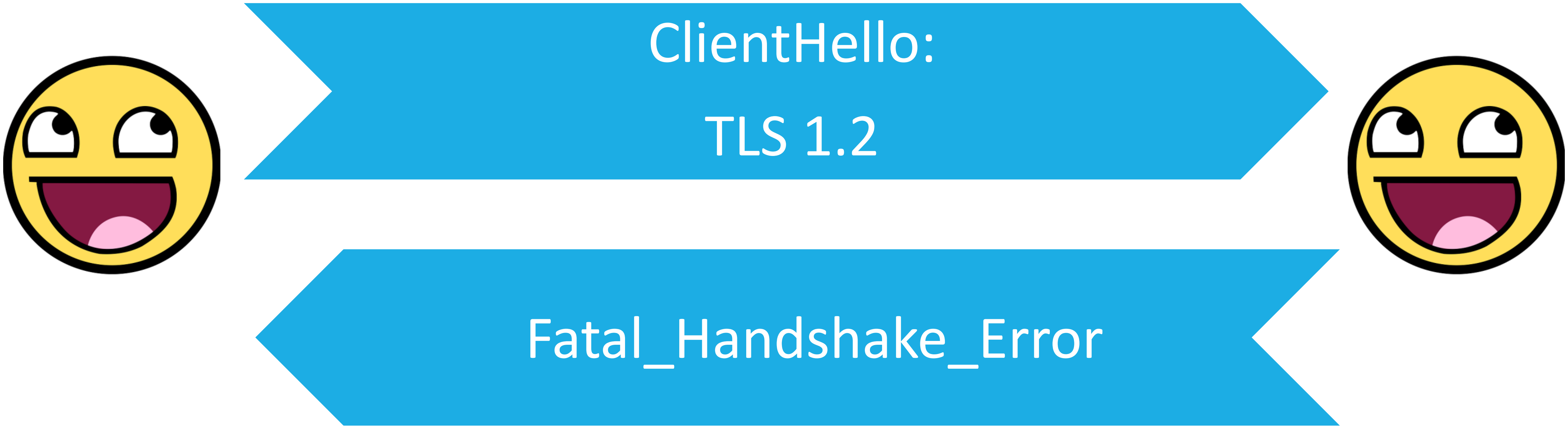
SCSV Non-Contiguous Example



ClientHello:

TLS 1.2

SCSV Non-Contiguous Example

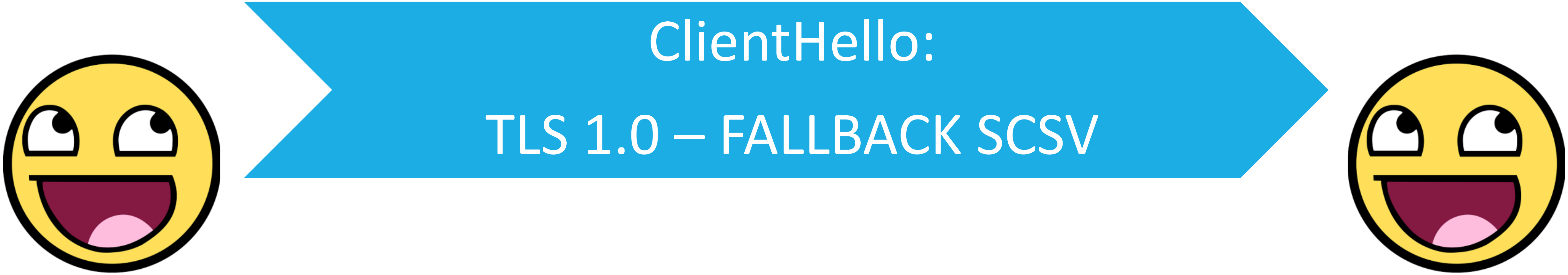


ClientHello:

TLS 1.2

Fatal_Handshake_Error

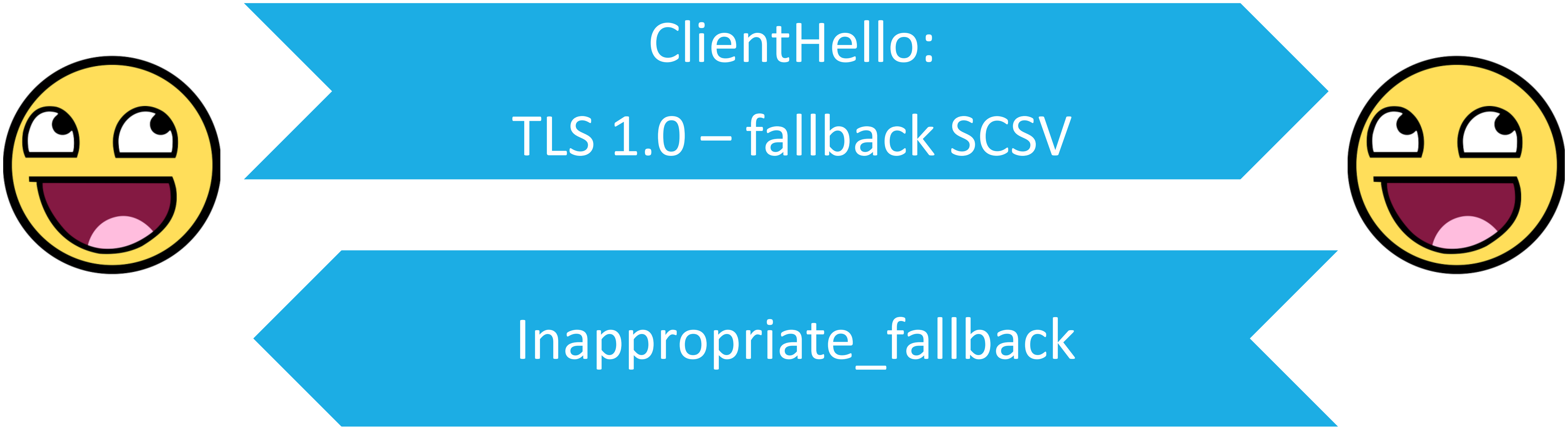
SCSV Non-Contiguous Example



ClientHello:

TLS 1.0 – FALLBACK SCSV

SCSV Non-Contiguous Example



ClientHello:

TLS 1.0 – fallback SCSV

Inappropriate_fallback

Conclusions

- When considering negotiation security, think:

Weakest Link Security

- Additionally:

Always authenticate everything

Thanks!

Questions?