UNIVERSITÄT PADERBORN
Die Universität der Informationsgesellschaft

Real-World Authenticated Key Exchange

Tibor Jager
Paderborn University

Summer School on Real-World Crypto and Privacy
Šibenik, Croatia
June 17th, 2019

Outline

- Security of the Diffie-Hellman Key Exchange
  - Man-in-the-Middle attacks
  - Forward Security
- TLS 1.3
  - Overview
  - The cryptographic core of TLS 1.3
- Real-World Problems
  - Problems arising from backwards compatibility
  - Middleboxes and ETS
- Further reading, open research problems

Diffie-Hellman Key Exchange

Public parameters:
Group description (G,g,q)

$a \leftarrow \{0, \ldots, q-1\}$
$t_A := g^a$

$b \leftarrow \{0, \ldots, q-1\}$
$t_B = g^b$

t_A

t_B

$k_{AB} := t_B^a$

$k_{AB} := t_A^b$

Man-in-the-middle Attack on DH

$a \leftarrow \{0, \ldots, q-1\}$

$t_A := g^a$

t_A

$a' \leftarrow \{0, \ldots, q-1\}$

$t_A' := g^{a'}$

t_A'

$b \leftarrow \{0, \ldots, q-1\}$

$t_B = g^b$

Man-in-the-middle Attack on DH

$a \leftarrow \{0, \ldots, q-1\}$
$t_A := g^a$

t_A

t_B'

$a' \leftarrow \{0, \ldots, q-1\}$
$t_A' := g^{a'}$

$b' \leftarrow \{0, \ldots, q-1\}$
$t_B' := g^{b'}$

t_A'

t_B

$b \leftarrow \{0, \ldots, q-1\}$
$t_B = g^b$

Man-in-the-middle Attack on DH

$a \leftarrow \{0, \ldots, q-1\}$

$t_A := g^a$

t_A

t_B'

$a' \leftarrow \{0, \ldots, q-1\}$

$t_A' := g^{a'}$

$b' \leftarrow \{0, \ldots, q-1\}$

$t_B' := g^{b'}$

$k_{AB'} = g^{ab'}$

$b \leftarrow \{0, \ldots, q-1\}$

$t_B = g^b$

t_A'

t_B

$k_{A'B} = g^{a'b}$

Man-in-the-middle Attack on DH

$a \leftarrow \{0, \ldots, q-1\}$
$t_A := g^a$

$b \leftarrow \{0, \ldots, q-1\}$
$t_B = g^b$

t_A

t_B'

$a' \leftarrow \{0, \ldots, q-1\}$
$t_A' := g^{a'}$

$b' \leftarrow \{0, \ldots, q-1\}$
$t_B' := g^{b'}$

t_A'

t_B

$k_{AB'} = g^{ab'}$

$k_{A'B} = g^{a'b}$

- This is an ***active*** attack
- DH is provably secure against ***passive*** (”eavesdropping”) attacks

Signed Diffie-Hellman

Public parameters:
Group description (G,g,q)

$(pk_A, sk_A) \leftarrow SigKeyGen()$

pk_A, pk_B

$(pk_B, sk_B) \leftarrow SigKeyGen()$

Signed Diffie-Hellman

$(pk_A, sk_A) \leftarrow SigKeyGen()$

Public parameters:
Group description (G,g,q)
pk_A, pk_B

$(pk_B, sk_B) \leftarrow SigKeyGen()$

$a \leftarrow \{0, \ldots, q-1\}$
$t_A := g^a$
$s_A := Sign(sk_A, t_A)$

t_A, s_A

Signed Diffie-Hellman

Public parameters:
Group description (G,g,q)
pk_A, pk_B

$(pk_A, sk_A) \leftarrow SigKeyGen()$

$(pk_B, sk_B) \leftarrow SigKeyGen()$

$a \leftarrow \{0, \ldots, q-1\}$
$t_A := g^a$
$s_A := Sign(sk_A, t_A)$

t_A, s_A

t_B, s_B

$b \leftarrow \{0, \ldots, q-1\}$
$t_B = g^b$
$s_B := Sign(sk_B, t_B)$

Signed Diffie-Hellman

Public parameters:
Group description (G,g,q)

$(pk_A, sk_A) \leftarrow SigKeyGen()$

pk_A, pk_B

$(pk_B, sk_B) \leftarrow SigKeyGen()$

$a \leftarrow \{0, \ldots, q-1\}$
$t_A := g^a$
$s_A := Sign(sk_A, t_A)$

t_A, s_A

t_B, s_B

$b \leftarrow \{0, \ldots, q-1\}$
$t_B = g^b$
$s_B := Sign(sk_B, t_B)$

If $Vfy(pk_B, t_B, s_B) = TRUE$ then:
$k_{AB} := t_B^a$

If $Vfy(pk_A, t_A, s_A) = TRUE$ then:
$k_{AB} := t_A^b$

Signed Diffie-Hellman

Public parameters:
Group description (G,g,q)
pk_A, pk_B

$(pk_A, sk_A) \leftarrow SigKeyGen()$

$(pk_B, sk_B) \leftarrow SigKeyGen()$

$a \leftarrow \{0, \ldots, q-1\}$
$t_A := g^a$
$s_A := Sign(sk_A, t_A)$

$b \leftarrow \{0, \ldots, q-1\}$
$t_B = g^b$
$s_B := Sign(sk_B, t_B)$

t_A, s_A

t_B, s_B

If $Vfy(pk_B, t_B, s_B) = TRUE$ then:
$k_{AB} := t_B^a$

If $Vfy(pk_A, t_A, s_A) = TRUE$ then:
$k_{AB} := t_A^b$

Security of the signature scheme prevents the MITM attack

Forward Security

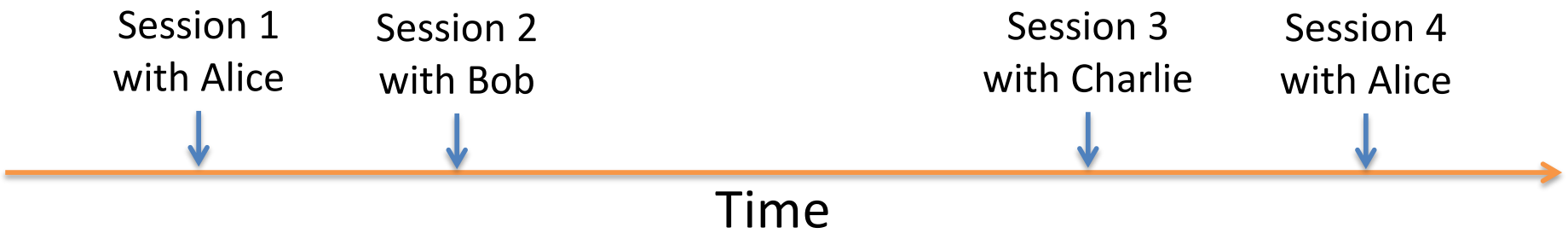
Objective:

Make large-scale collection of encrypted data useless

Forward Security

Objective:

Make large-scale collection of encrypted data useless



Forward Security

Objective:

Make large-scale collection of encrypted data useless

Secret key

Session 1 with Alice

Session 2 with Bob

Session 3 with Charlie

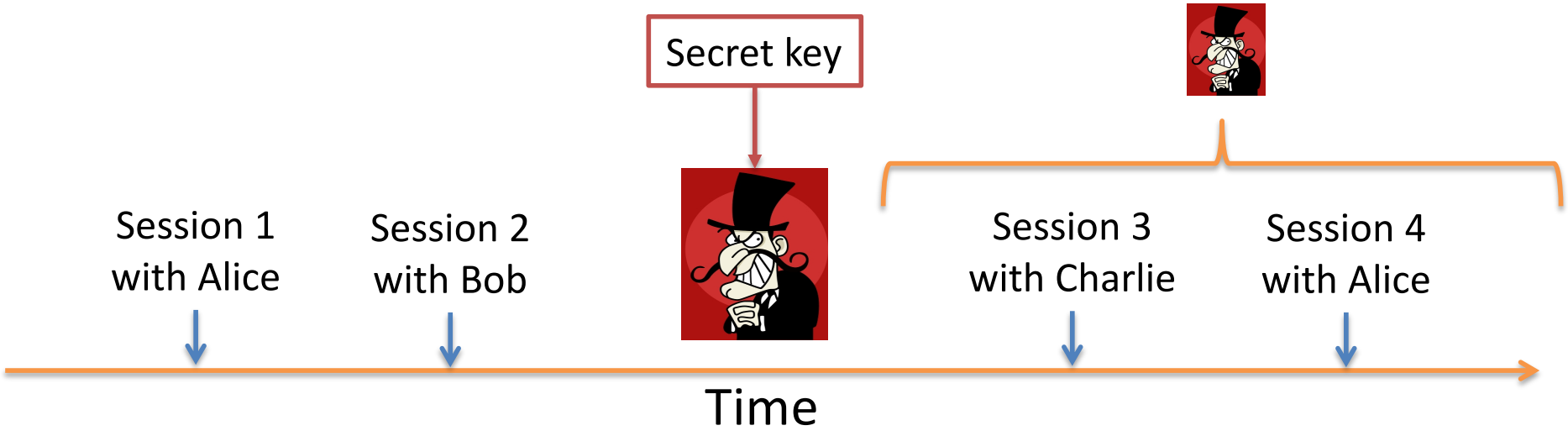
Session 4 with Alice

Time

Forward Security

Objective:

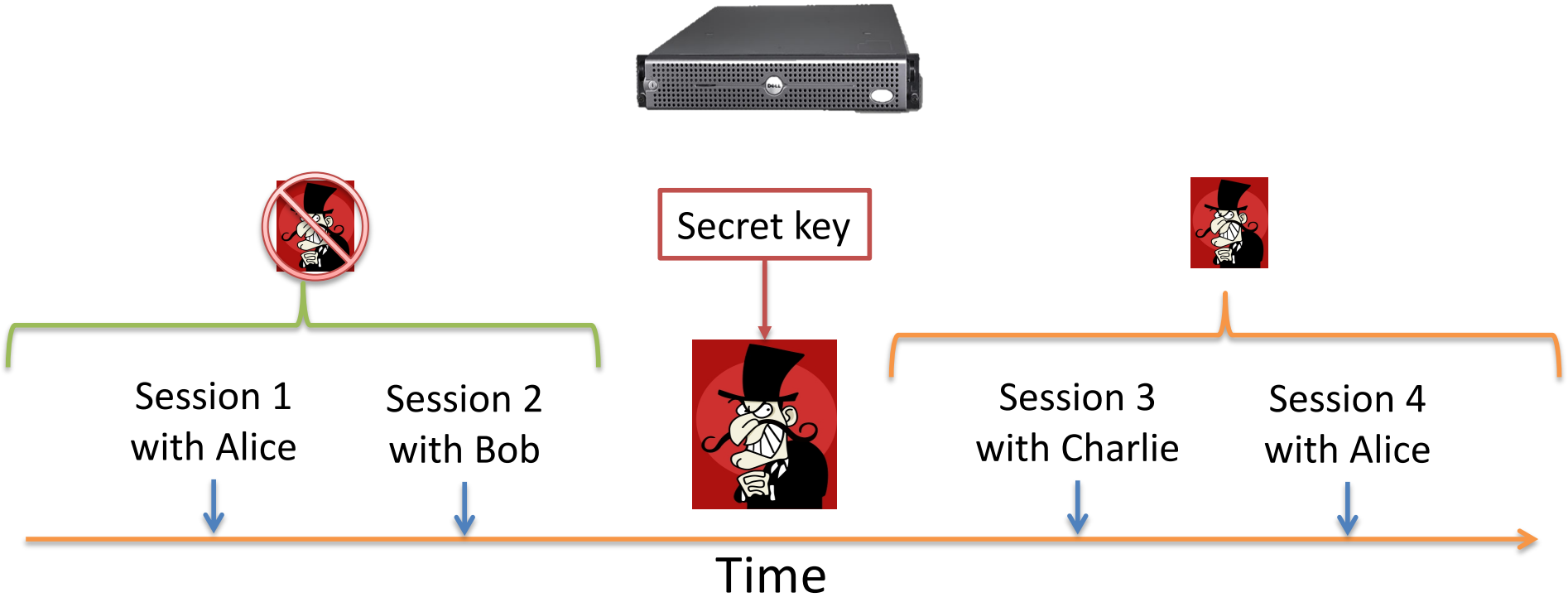
Make large-scale collection of encrypted data useless



Forward Security

Objective:

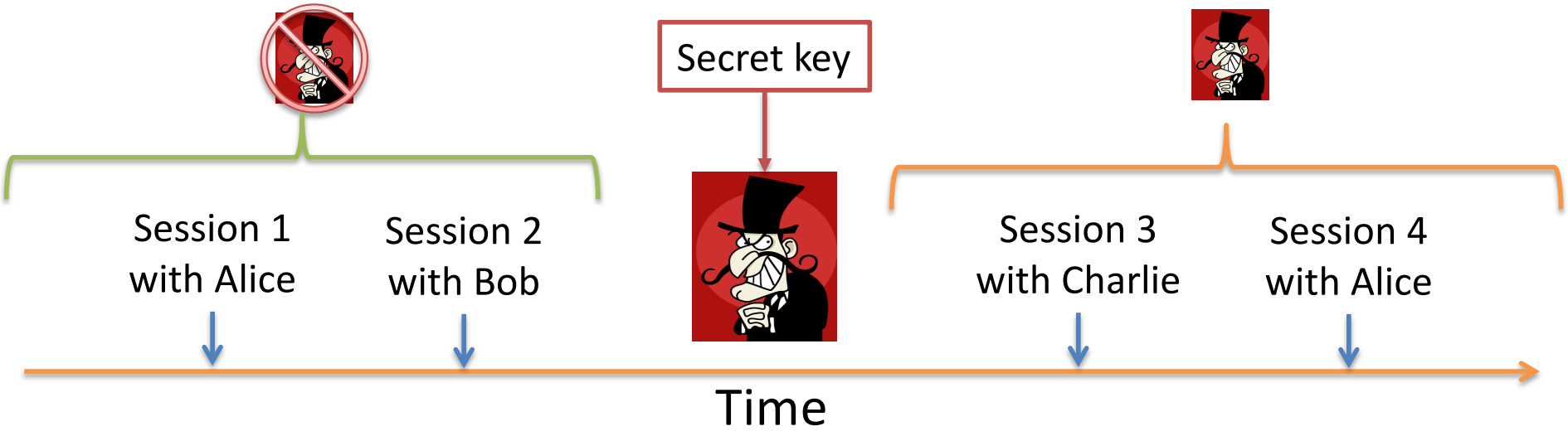
Make large-scale collection of encrypted data useless



Forward Security

Objective:

Make large-scale collection of encrypted data useless



- **Widely used:** Google facebook amazon.com
- **Standard security goal** of modern protocols

Forward Security of Signed DH

$(pk_A, \mathbf{sk_A}) \leftarrow SigKeyGen()$

Public parameters:
Group description (G,g,q)
pk_A, pk_B

$(pk_B, \mathbf{sk_B}) \leftarrow SigKeyGen()$

$a \leftarrow \{0, \ldots, q-1\}$
$t_A := g^a$
$s_A := Sign(sk_A, t_A)$

t_A, s_A →

← t_B, s_B

$b \leftarrow \{0, \ldots, q-1\}$
$t_B = g^b$
$s_B := Sign(sk_B, t_B)$

If $Vfy(pk_{Bob}, t_B, s_B) = TRUE$ then:
$\mathbf{k_{AB} := t_B^a}$

If $Vfy(pk_{Alice}, t_A, s_A) = TRUE$ then:
$\mathbf{k_{AB} := t_A^b}$

Forward secure (if ephemeral exponents are not stored)

Are we done?

- Signed DH is a beautiful protocol…
  - Clean and simple
  - Easy to implement and analyze
  - Forward Security

Are we done?

- Signed DH is a beautiful protocol…
  - Clean and simple
  - Easy to implement and analyze
  - Forward Security

- … but lacking features considered important in the real world, for instance:
  - How are **public keys** distributed?
  - No **key confirmation**
  - **Fixed DH groups** and **signature schemes**
  - Protocol for **encryption of payload data** not specified

Are we done?

- Signed DH is a beautiful protocol…
  - Clean and simple
  - Easy to implement and anal
  - Forward Security
- … but lacking features cons
  the real world, for instance
  - How are **public keys** distribu
  - No **key confirmation**
  - **Fixed DH groups** and **signat**
  - Protocol for **encryption of p**

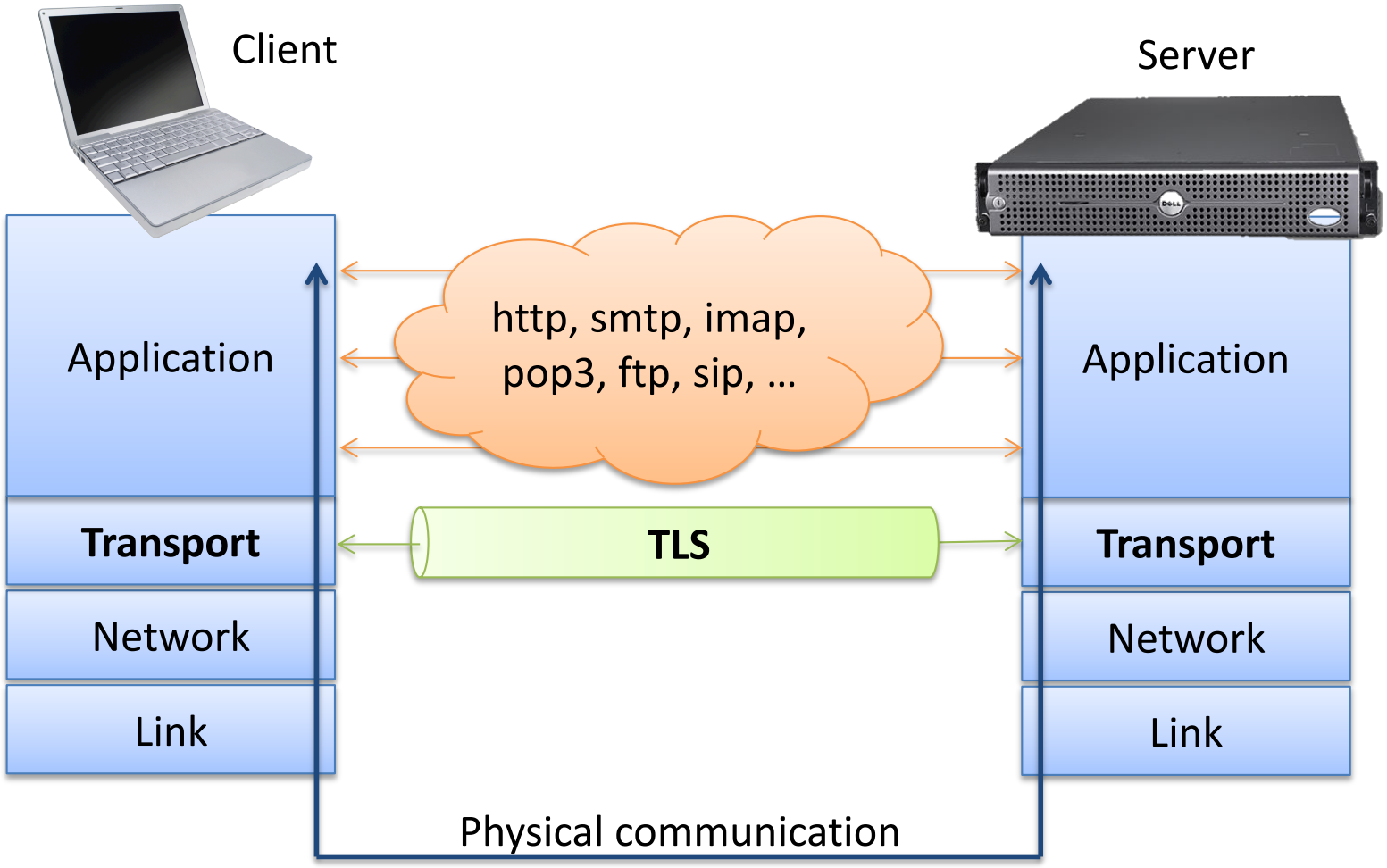
Further issues:

- How to deal with errors
  - Alert messages
  - Protocol spec.
- Interoperability
  - Message formats
  - Protocol headers
- Possible extensions
- Implementational issues
- …

Outline

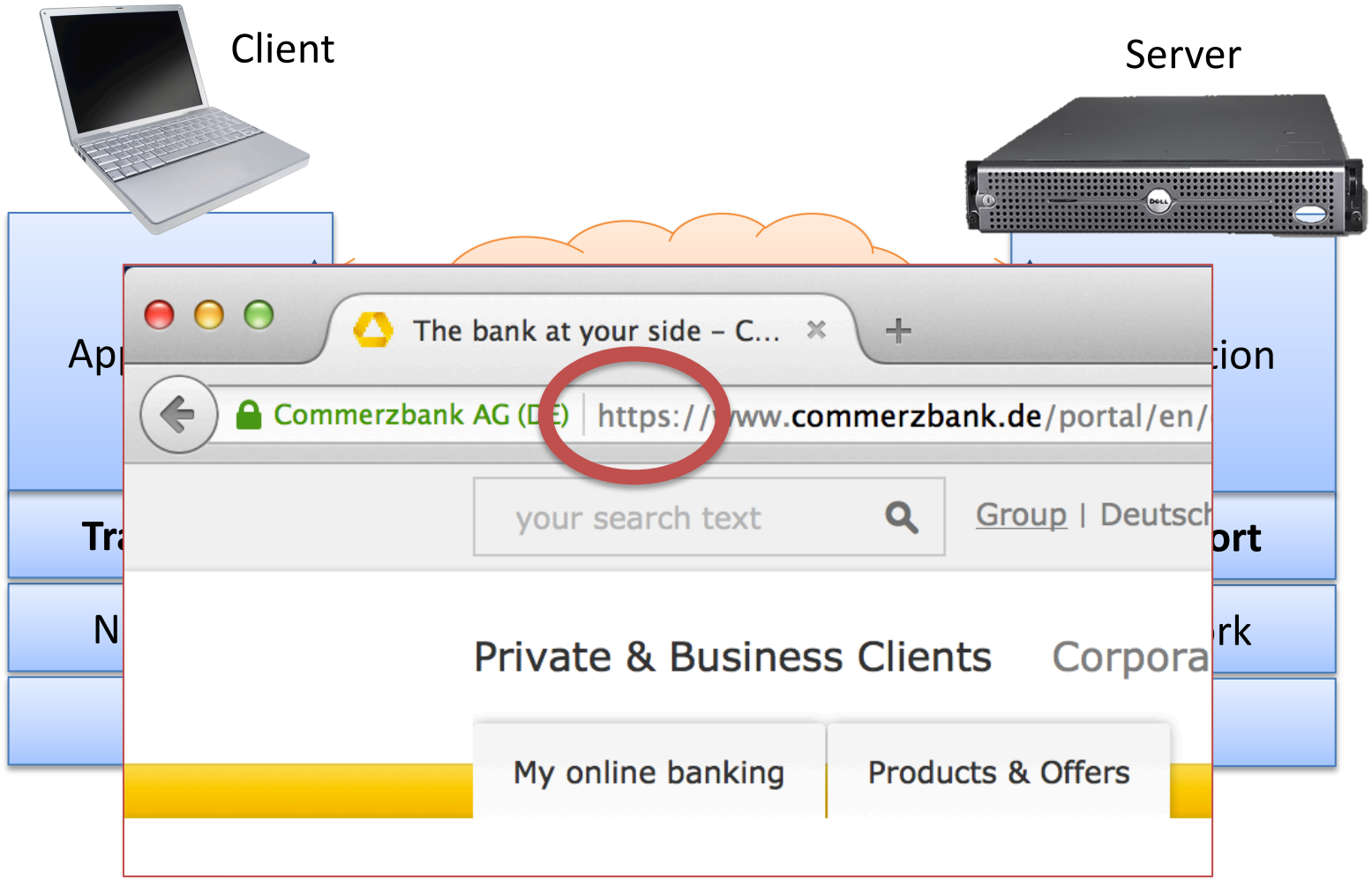
- Security of the Diffie-Hellman Key Exchange
  - Man-in-the-Middle attacks
  - Forward Security
- TLS 1.3
  - Overview
  - The cryptographic core of TLS 1.3
- Real-World Problems
  - Problems arising from backwards compatibility
  - Middleboxes and ETS
- Further reading, open research problems

Transport Layer Security (TLS)



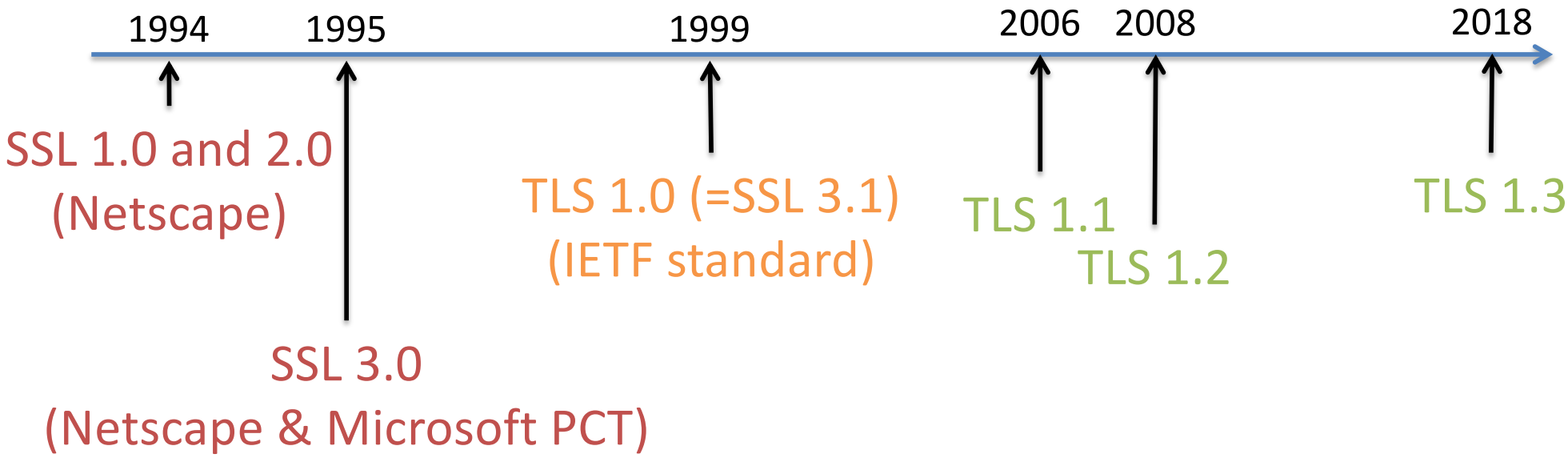
Goal: provide **confidential**, **authenticated**, **integrity-protected** **channel**

Transport Layer Security (TLS)

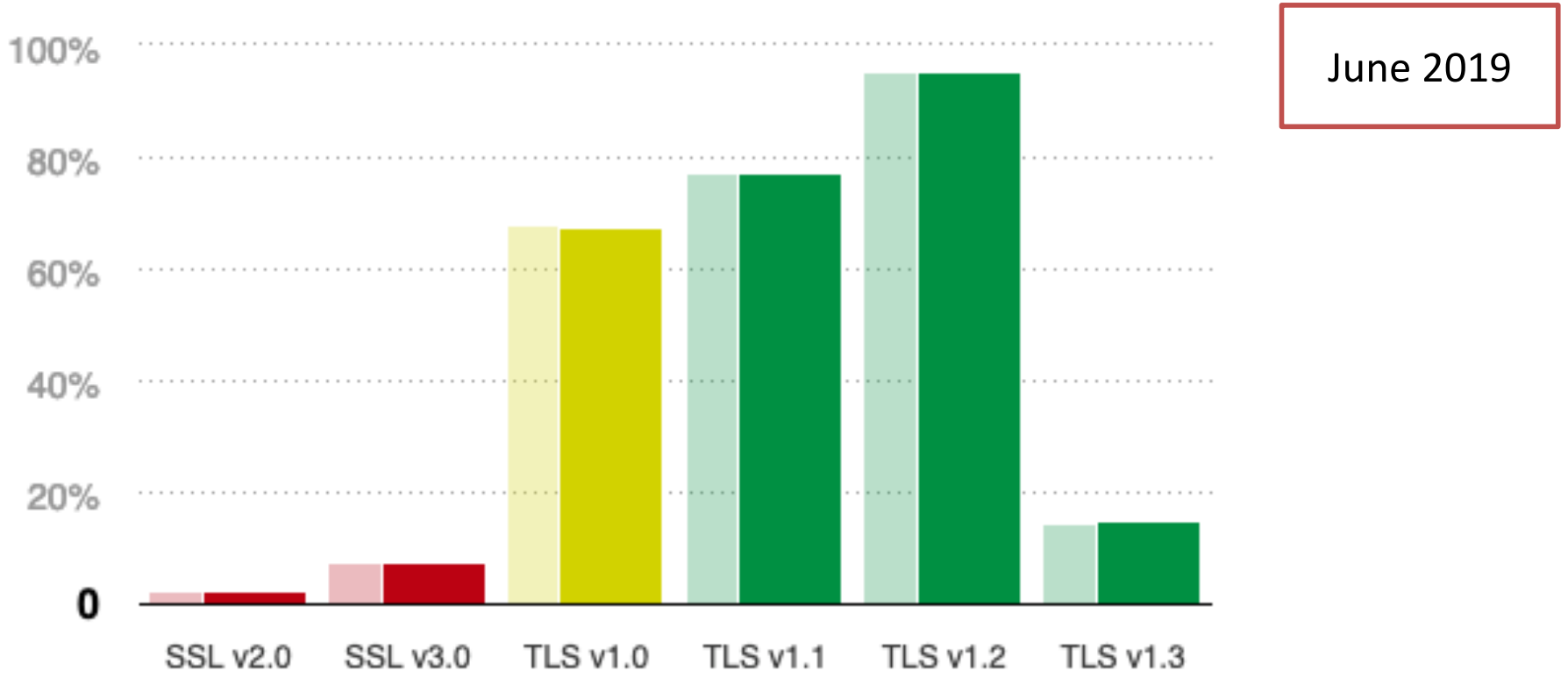


Goal: provide **confidential**, **authenticated**, **integrity-protected** **channel**

TLS vs. SSL

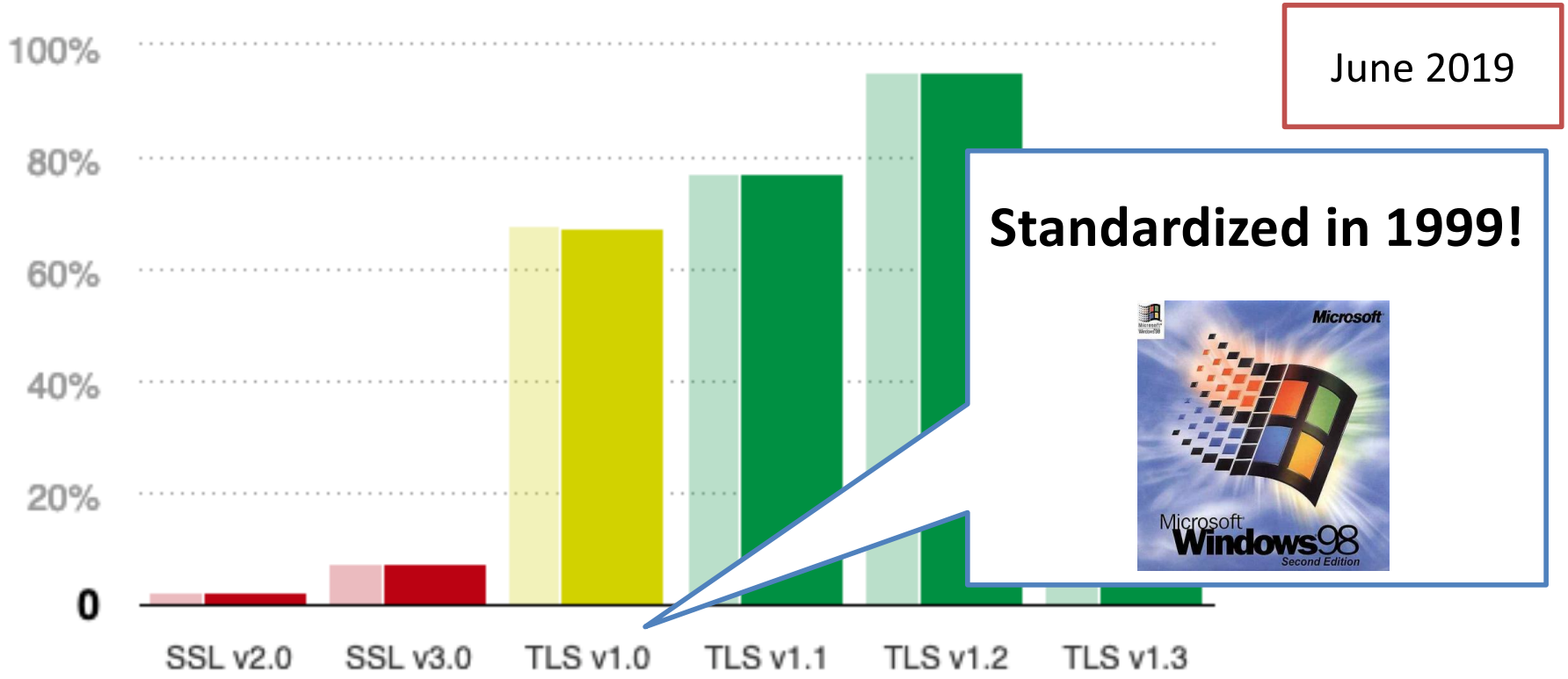

1994
1995
1999
2006
2008
2018
SSL 1.0 and 2.0
(Netscape)
SSL 3.0
(Netscape & Microsoft PCT)
TLS 1.0 (=SSL 3.1)
(IETF standard)
TLS 1.1
TLS 1.2
TLS 1.3

Use of SSL/TLS Versions in Practice



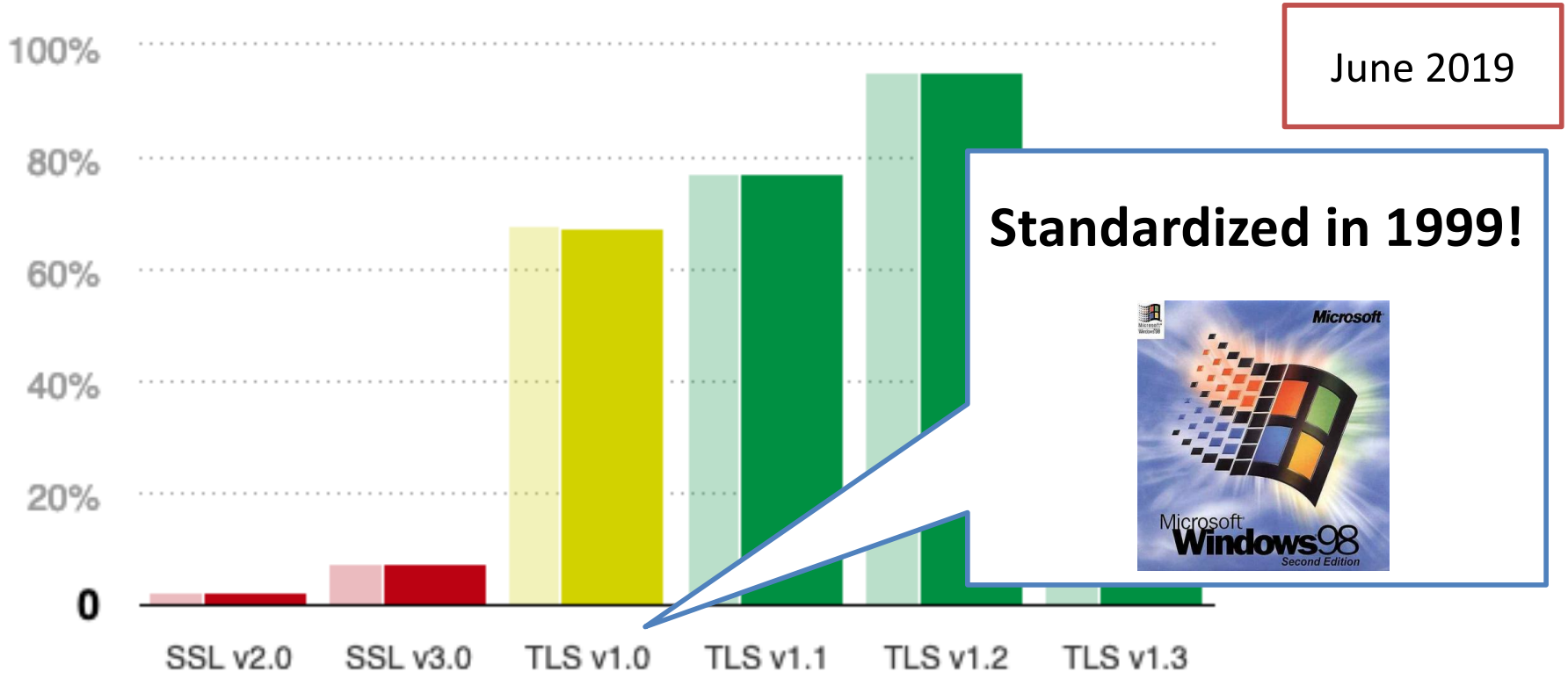
https://www.ssllabs.com/ssl-pulse/

Use of SSL/TLS Versions in Practice



https://www.ssllabs.com/ssl-pulse/

Use of SSL/TLS Versions in Practice



Security protocols have an **extremely long life time**

https://www.ssllabs.com/ssl-pulse/

TLS Sessions: Handshake + Record Layer Encryption

Handshake:

- Negotiation of **cryptographic algorithms** (KE, Sig., *Cipher Suite*)
- **Authentication** of comm. partners
- Establishment of **session key** k

TLS Sessions: Handshake + Record Layer Encryption

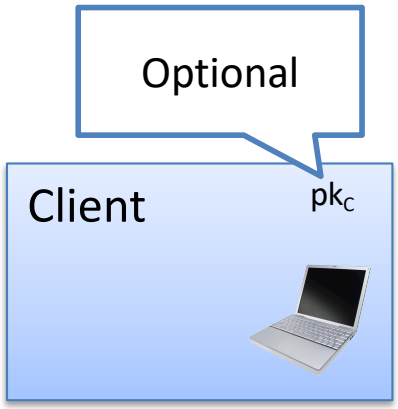
Handshake:

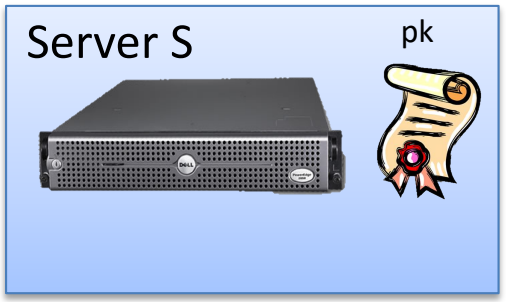
- Negotiation of **cryptographic algorithms** (KE, Sig., *Cipher Suite*)
- **Authentication** of comm. partners
- Establishment of **session key** k

Record Layer Encryption:

- Data **encryption** and **authentication** using key k

The Cryptographic Core of the TLS 1.3 Handshake





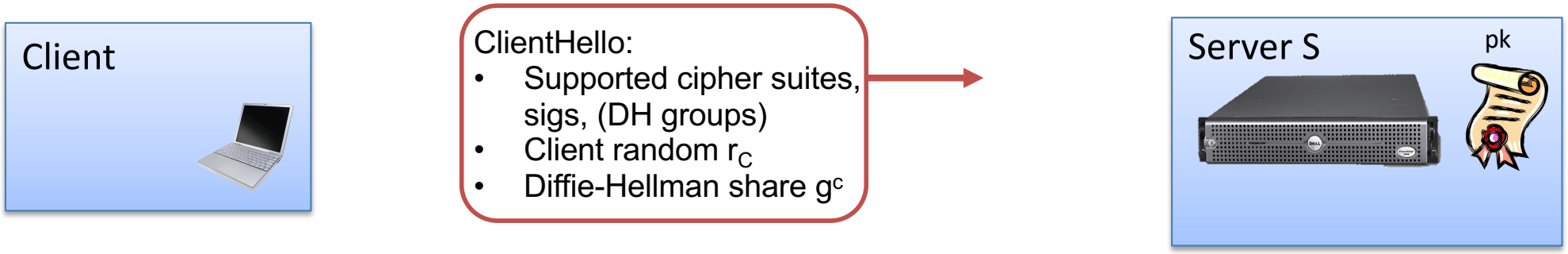
The Cryptographic Core of the TLS 1.3 Handshake

Client

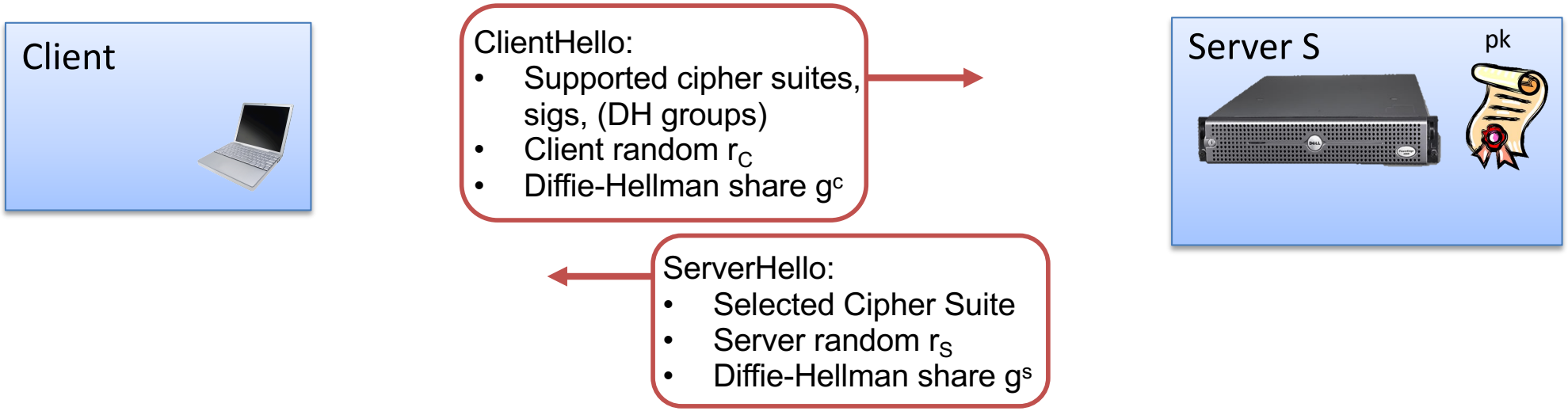
Server S

pk

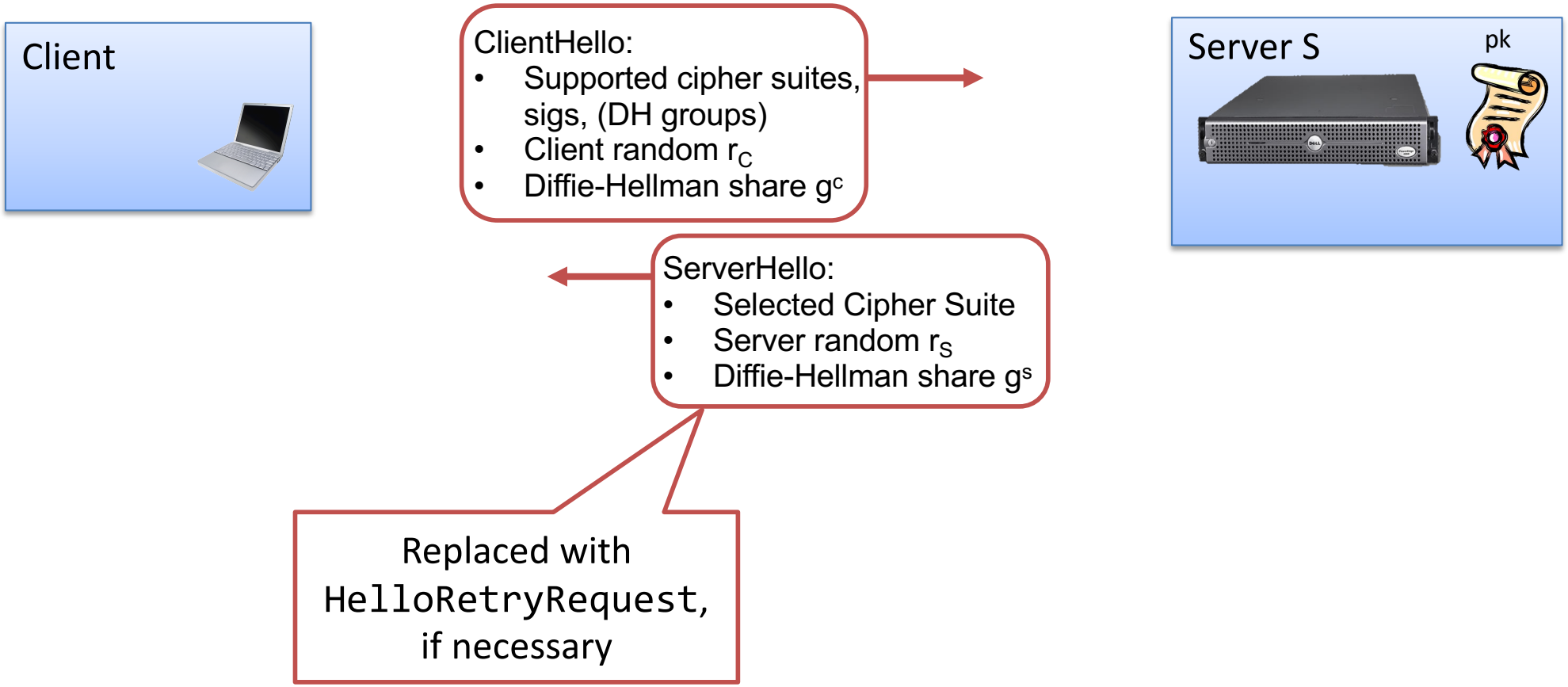
The Cryptographic Core of the TLS 1.3 Handshake



The Cryptographic Core of the TLS 1.3 Handshake



The Cryptographic Core of the TLS 1.3 Handshake



The Cryptographic Core of the TLS 1.3 Handshake

Client

ClientHello:
- Supported cipher suites, sigs, (DH groups)
- Client random r_C
- Diffie-Hellman share g^c

ServerHello:
- Selected Cipher Suite
- Server random r_S
- Diffie-Hellman share g^s

Server S

pk

$k = KDF(g^{cs}, r_C, r_S)$
$k' = KDF'(g^{cs}, r_C, r_S)$

$k = KDF(g^{cs}, r_C, r_S)$
$k' = KDF'(g^{cs}, r_C, r_S)$

The Cryptographic Core of the TLS 1.3 Handshake

Client

ClientHello:
- Supported cipher suites, sigs, (DH groups)
- Client random r_C
- Diffie-Hellman share g^c

Server S

pk

ServerHello:
- Selected Cipher Suite
- Server random r_S
- Diffie-Hellman share g^s

$k = KDF(g^{cs}, r_C, r_S)$
$k' = KDF'(g^{cs}, r_C, r_S)$

$k = KDF(g^{cs}, r_C, r_S)$
$k' = KDF'(g^{cs}, r_C, r_S)$

Certified public key

Certificate Verify

Signature over all previous messages

Encrypted with k

The Cryptographic Core of the TLS 1.3 Handshake

Client

Server S

pk

ClientHello:
- Supported cipher suites, sigs, (DH groups)
- Client random r_C
- Diffie-Hellman share g^c

ServerHello:
- Selected Cipher Suite
- Server random r_S
- Diffie-Hellman share g^s

Certified public key

Certificate Verify

Server Finished SFIN

$k = KDF(g^{cs}, r_C, r_S)$
$k' = KDF'(g^{cs}, r_C, r_S)$

Encrypted with k

$k = KDF(g^{cs}, r_C, r_S)$
$k' = KDF'(g^{cs}, r_C, r_S)$

Signature over all previous messages

SFIN = MAC(k', all prev. msgs.)

The Cryptographic Core of the TLS 1.3 Handshake

Client

ClientHello:
- Supported cipher suites, sigs, (DH groups)
- Client random r_C
- Diffie-Hellman share g^c

Server S

pk

$k = KDF(g^{cs}, r_C, r_S)$
$k' = KDF'(g^{cs}, r_C, r_S)$

ServerHello:
- Selected Cipher Suite
- Server random r_S
- Diffie-Hellman share g^s

$k = KDF(g^{cs}, r_C, r_S)$
$k' = KDF'(g^{cs}, r_C, r_S)$

Encrypted with k

Certified public key

Certificate Verify

Signature over all previous messages

Server Finished SFIN

SFIN = MAC(k', all prev. msgs.)

CFIN = MAC(k', all prev. msgs.)

Client Finished CFIN

The Cryptographic Core of the TLS 1.3 Handshake

Client

ClientHello:
- Supported cipher suites, sigs, (DH groups)
- Client random r_C
- Diffie-Hellman share g^c

Server S

pk

$k = KDF(g^{cs}, r_C, r_S)$
$k' = KDF'(g^{cs}, r_C, r_S)$

ServerHello:
- Selected Cipher Suite
- Server random r_S
- Diffie-Hellman share g^s

$k = KDF(g^{cs}, r_C, r_S)$
$k' = KDF'(g^{cs}, r_C, r_S)$

Encrypted with k

Certified public key

Signature over all previous messages

Certificate Verify

Server Finished SFIN

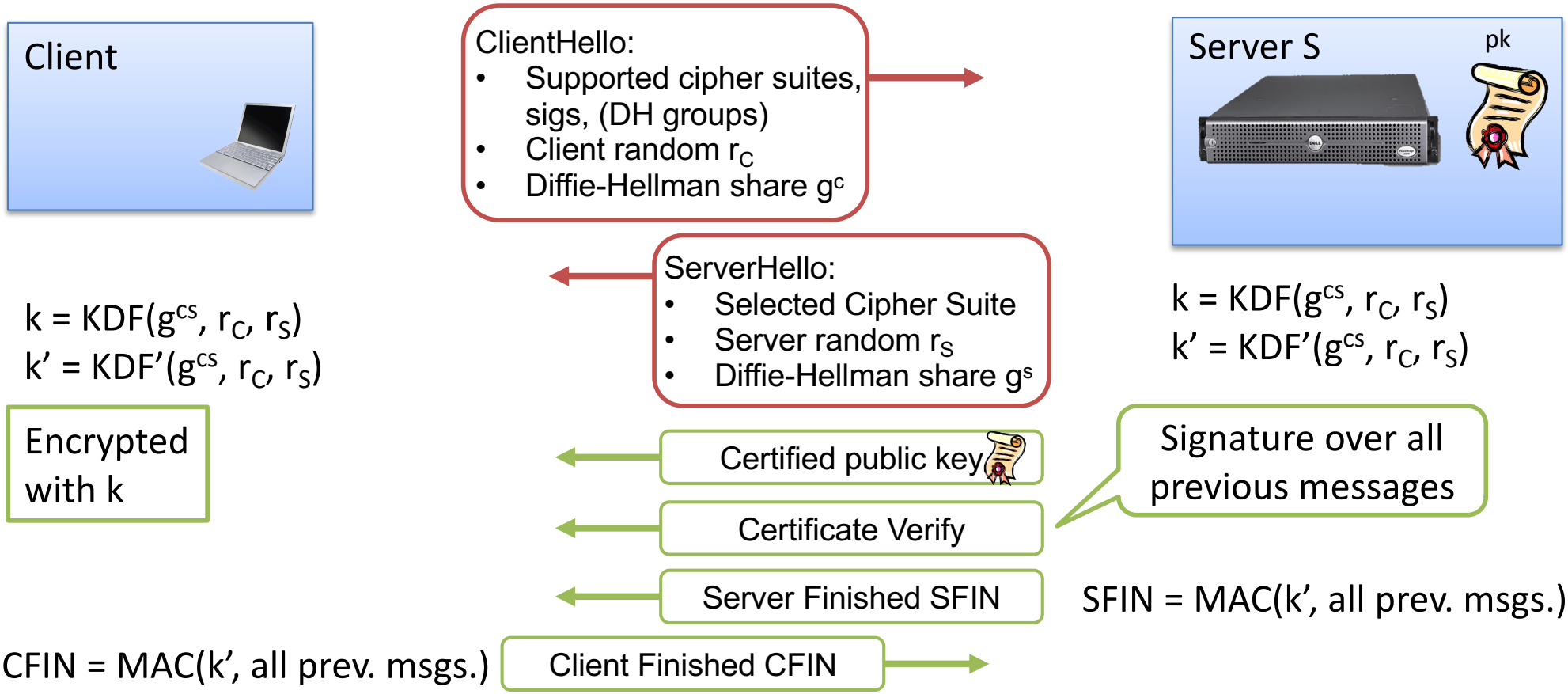
$SFIN = MAC(k', \text{all prev. msgs.})$

$CFIN = MAC(k', \text{all prev. msgs.})$

Client Finished CFIN

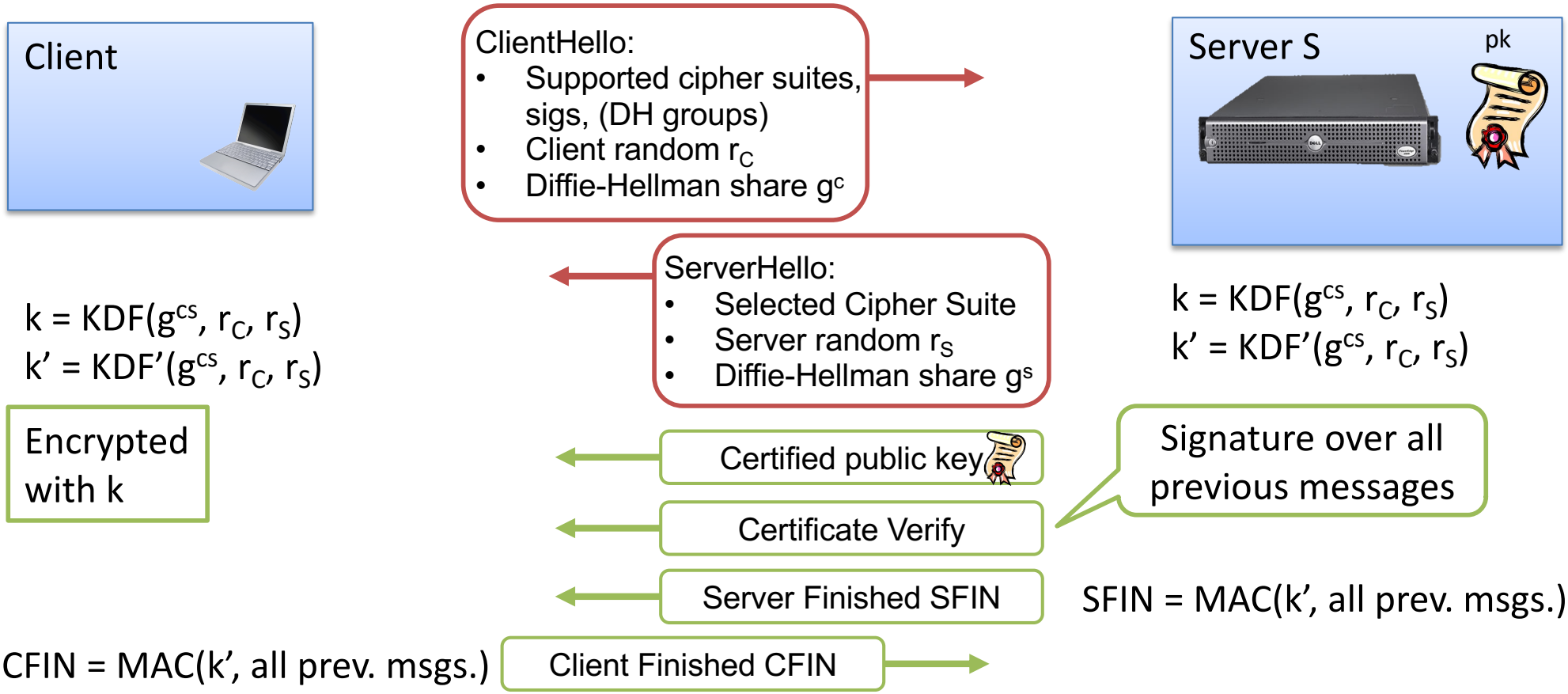
- Public key distribution ✓

The Cryptographic Core of the TLS 1.3 Handshake



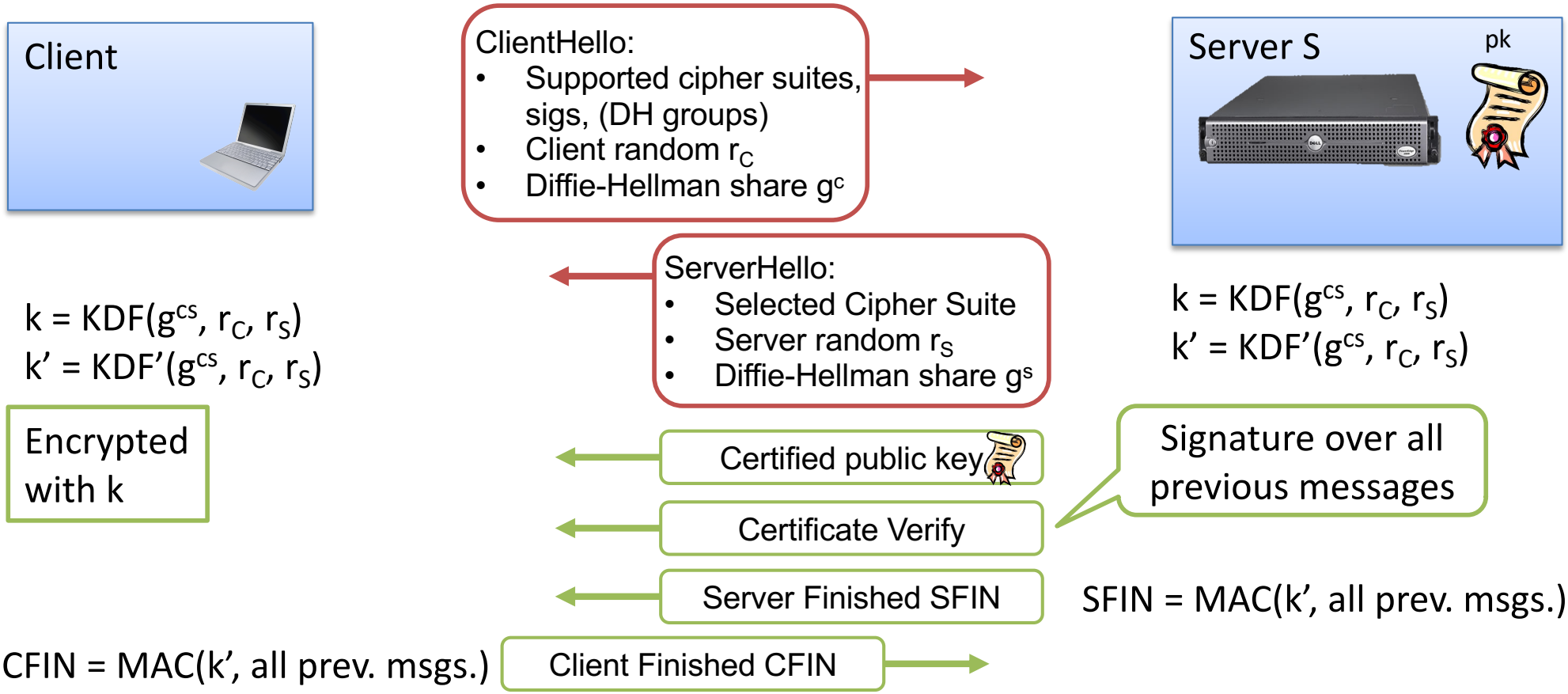
- Public key distribution ✓
- Key confirmation ✓

The Cryptographic Core of the TLS 1.3 Handshake



- Public key distribution ✓
- Key confirmation ✓
- Possible to use various DH groups, signature and encryption schemes ✓

The Cryptographic Core of the TLS 1.3 Handshake



- Public key distribution ✓
- Key confirmation ✓
- Possible to use various DH groups, signature and encryption schemes ✓
- 1 RTT before cryptographically protected payload can be sent ✓

Cool Innovations of TLS 1.3

- Removed:
  - RSA-PKCS #1 v1.5 encryption
    - 20 years after Bleichenbacher‘s attack from 1998
  - Compression of plaintext data (e.g., CRIME attack)
  - RC4 encryption
  - CBC-mode encryption, only Authenticated Encryption
  - Custom DH groups
    - Only 5 ECC and 5 finite field groups supported
  - …

Cool Innovations of TLS 1.3

- Removed:
  - RSA-PKCS #1 v1.5 encryption
    - 20 years after Bleichenbacher‘s attack from 1998
  - Compression of plaintext data (e.g., CRIME attack)
  - RC4 encryption
  - CBC-mode encryption, only Authenticated Encryption
  - Custom DH groups
    - Only 5 ECC and 5 finite field groups supported
  - …
- New:
  - Forward Security as an explicit design goal
  - 1 RTT key establishment (2 RTT in TLS 1.2), “0-RTT” mode
  - First version where the modern “provable security” approach was used to justify design decisions

Cool Innovations of TLS 1.3

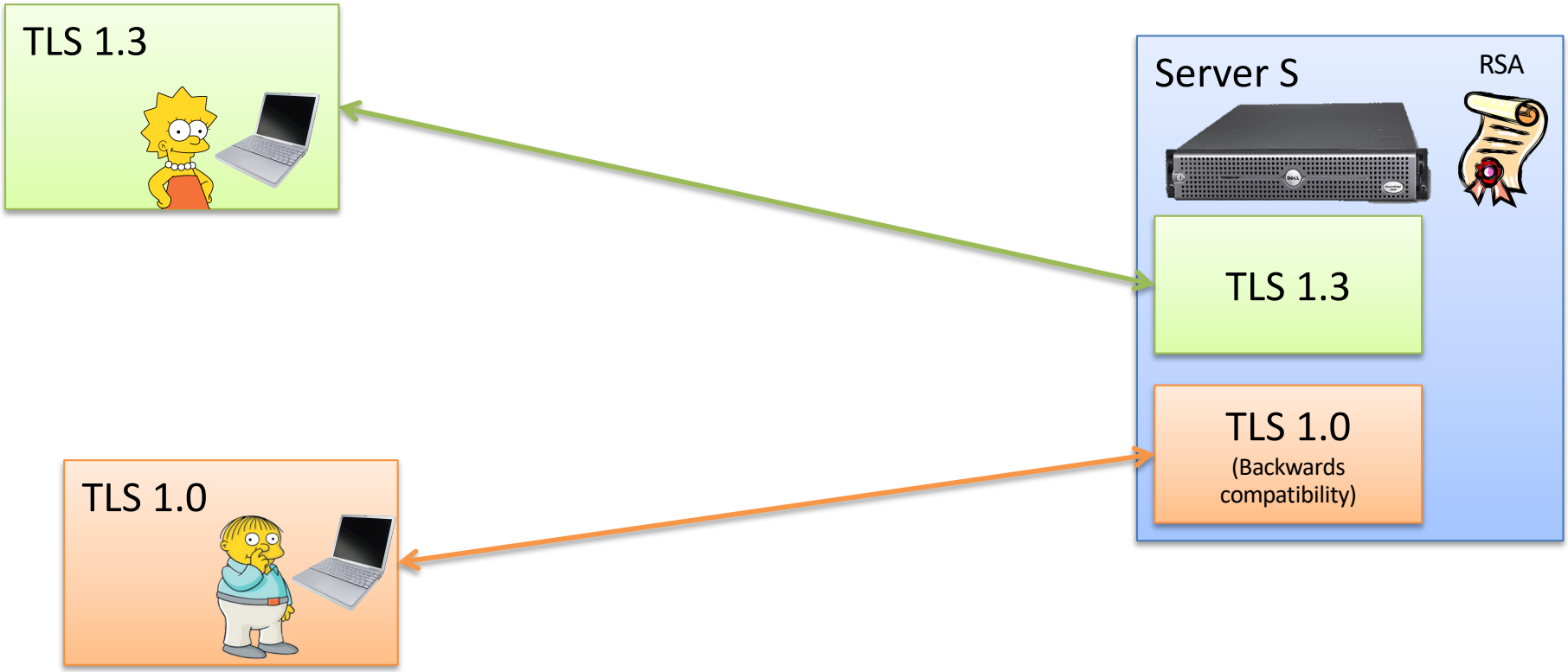
- Removed:
  - RSA-PKCS #1 v1.5 encryption
    - 20 years after Bleichenbacher‘s attack from 1998
  - Compression of plaintext data (e.g., CRIME attack)
  - RC4 encryption
  - CBC-mode encryption, only Authenticated Encryption
  - Custom DH groups
    - Only 5 ECC and 5 finite field groups supported
  - …
- New:
  - Forward Security as an explicit design goal
  - 1 RTT key establishment (2 RTT in TLS 1.2), “0-RTT” mode
  - First version where the modern “provable security” approach was used to justify design decisions

TLS 1.3 is simpler, more efficient, **and** more secure

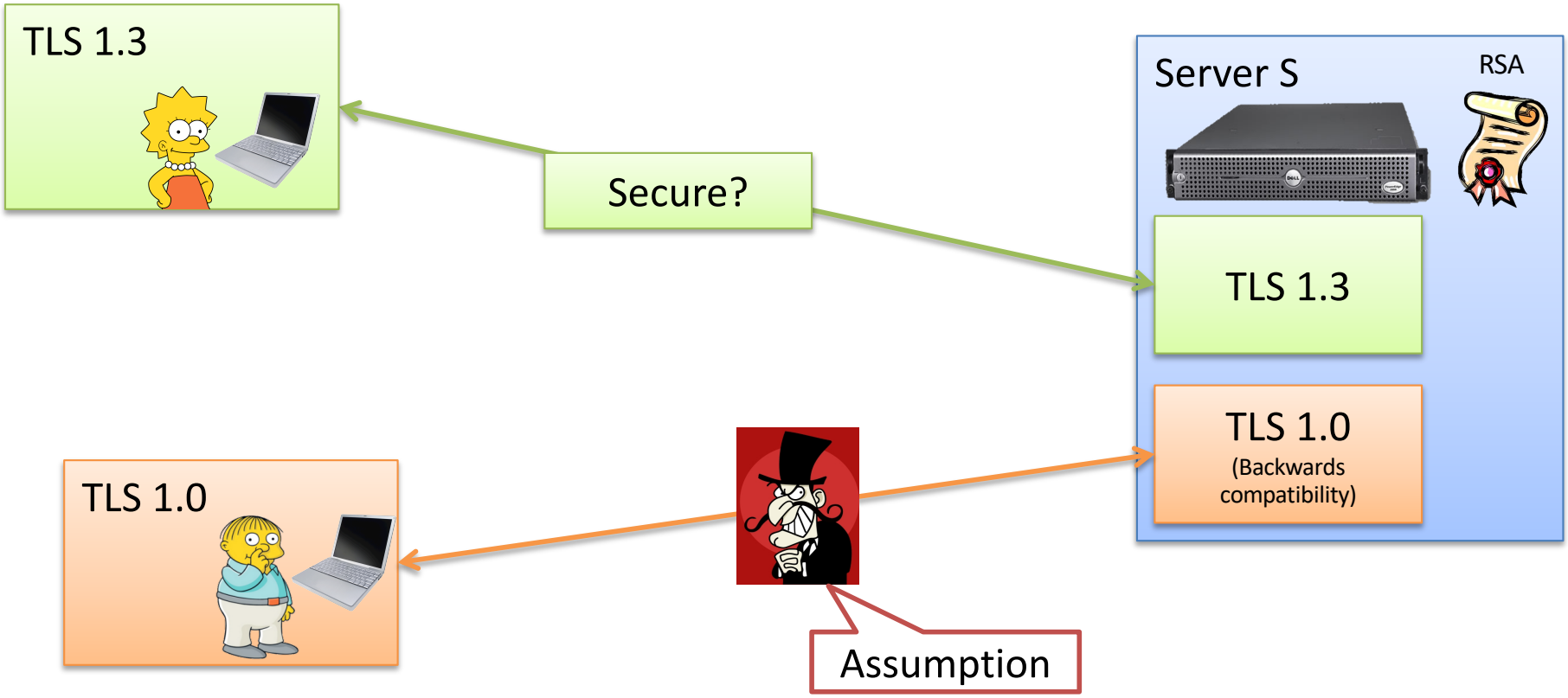
Outline

- Security of the Diffie-Hellman Key Exchange
  - Man-in-the-Middle attacks
  - Forward Security
- TLS 1.3
  - Overview
  - The cryptographic core of TLS 1.3
- Real-World Problems
  - Problems arising from backwards compatibility
  - Middleboxes and ETS
- Further reading, open research problems

Typical use of TLS 1.3 in practice

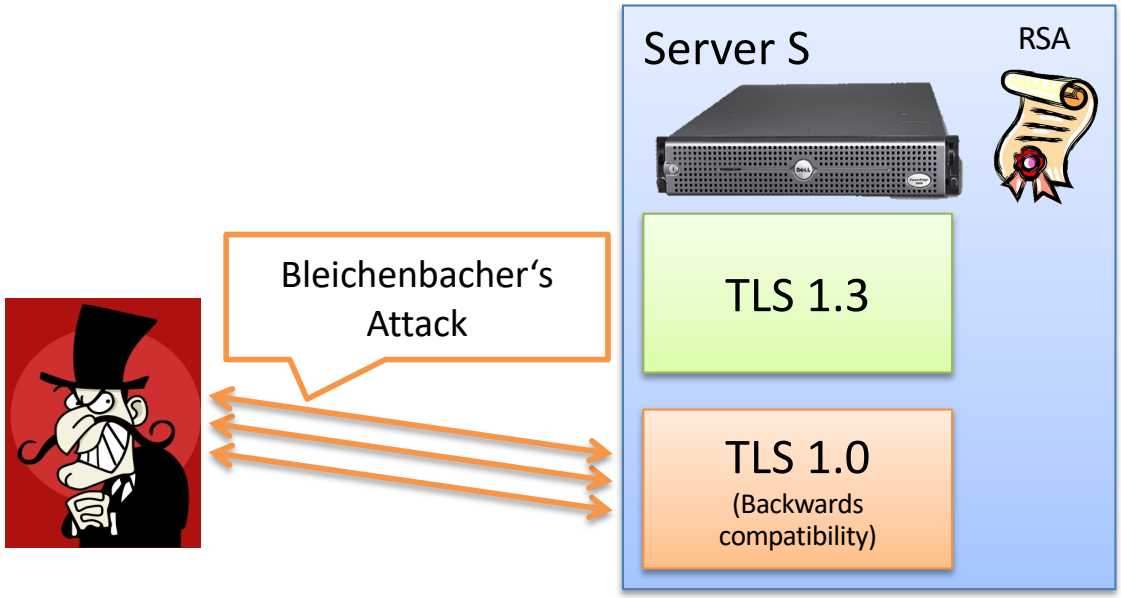


Typical use of TLS 1.3 in practice



Bleichenbacher‘s Attack

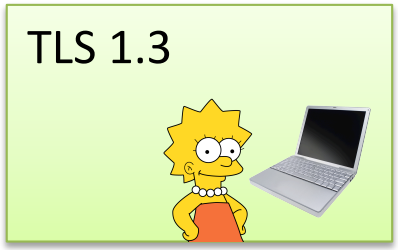
Daniel Bleichenbacher, CRYPTO 1998

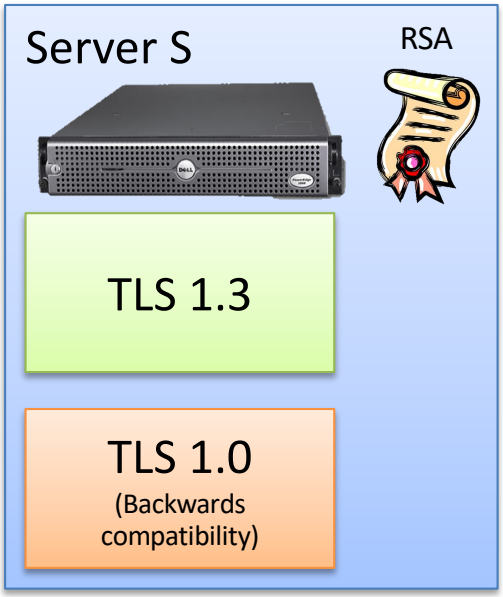


Enables the attacker to create a **valid digital signature** w.r.t. the server’s RSA public key, for an **arbitrary message**

Backwards Compatibility Attack on TLS 1.3

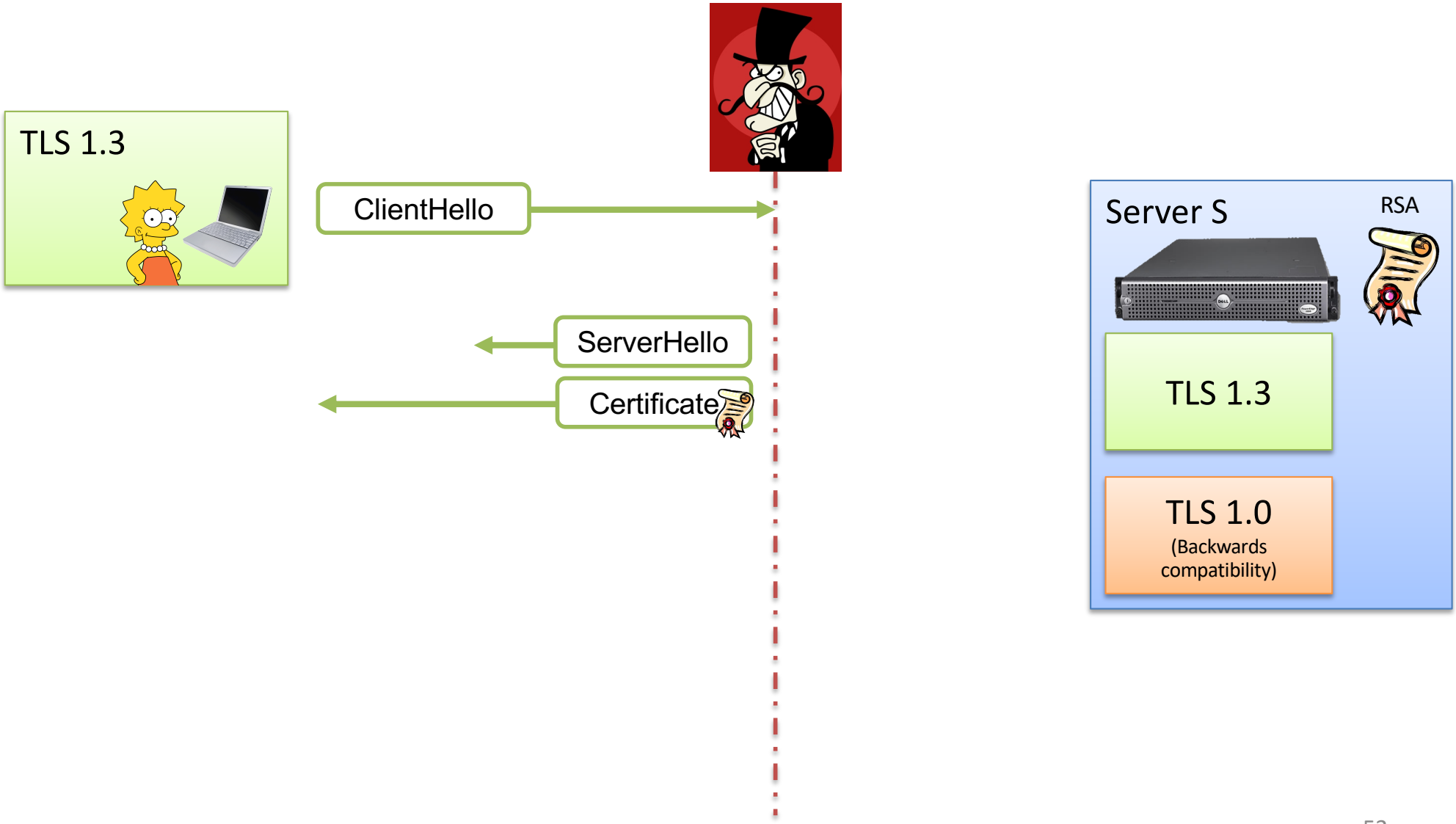
[J., Schwenk, Somorovsky; ACM CCS 2015]





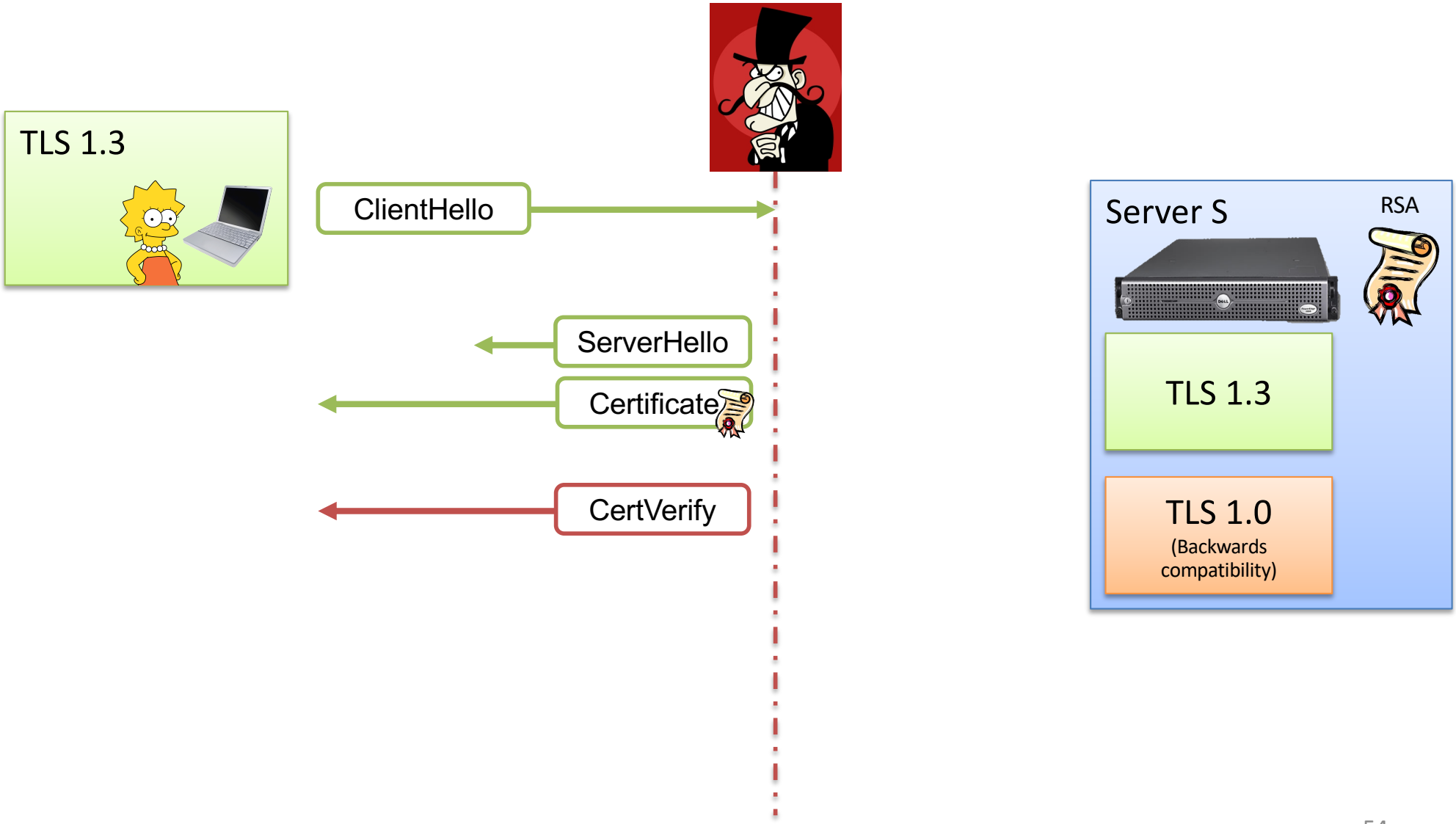
Backwards Compatibility Attack on TLS 1.3

[J., Schwenk, Somorovsky; ACM CCS 2015]



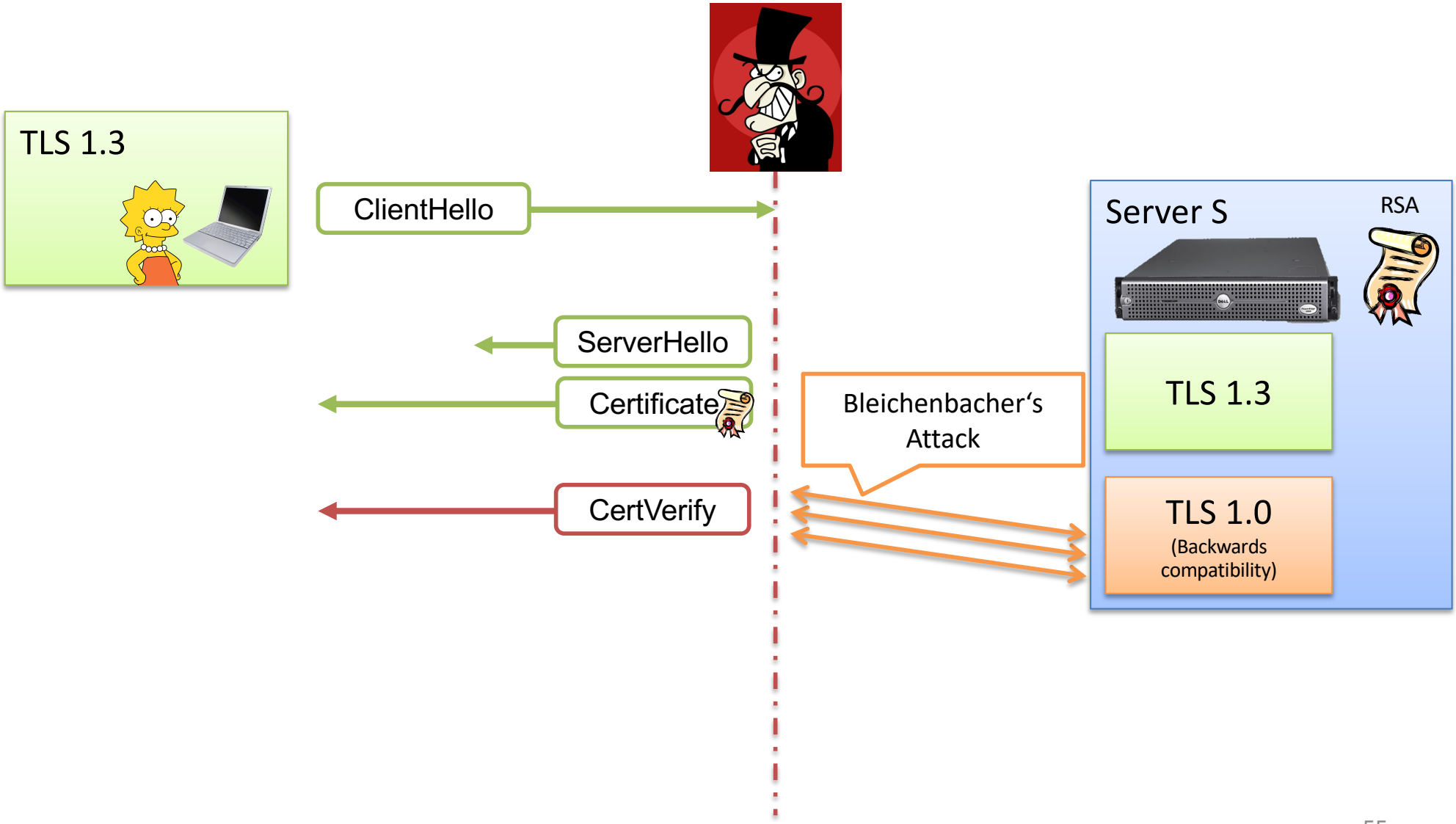
Backwards Compatibility Attack on TLS 1.3

[J., Schwenk, Somorovsky; ACM CCS 2015]



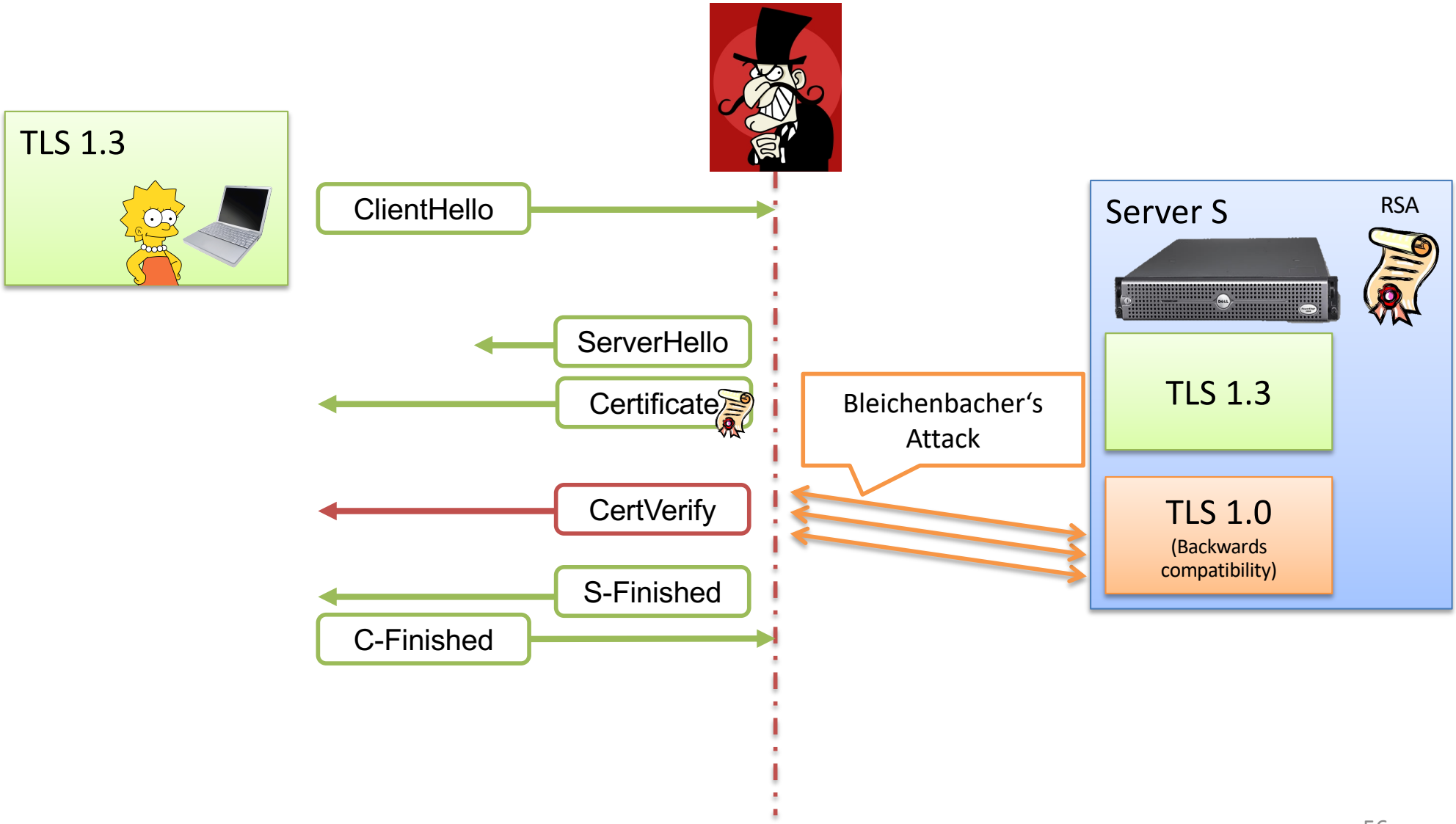
Backwards Compatibility Attack on TLS 1.3

[J., Schwenk, Somorovsky; ACM CCS 2015]



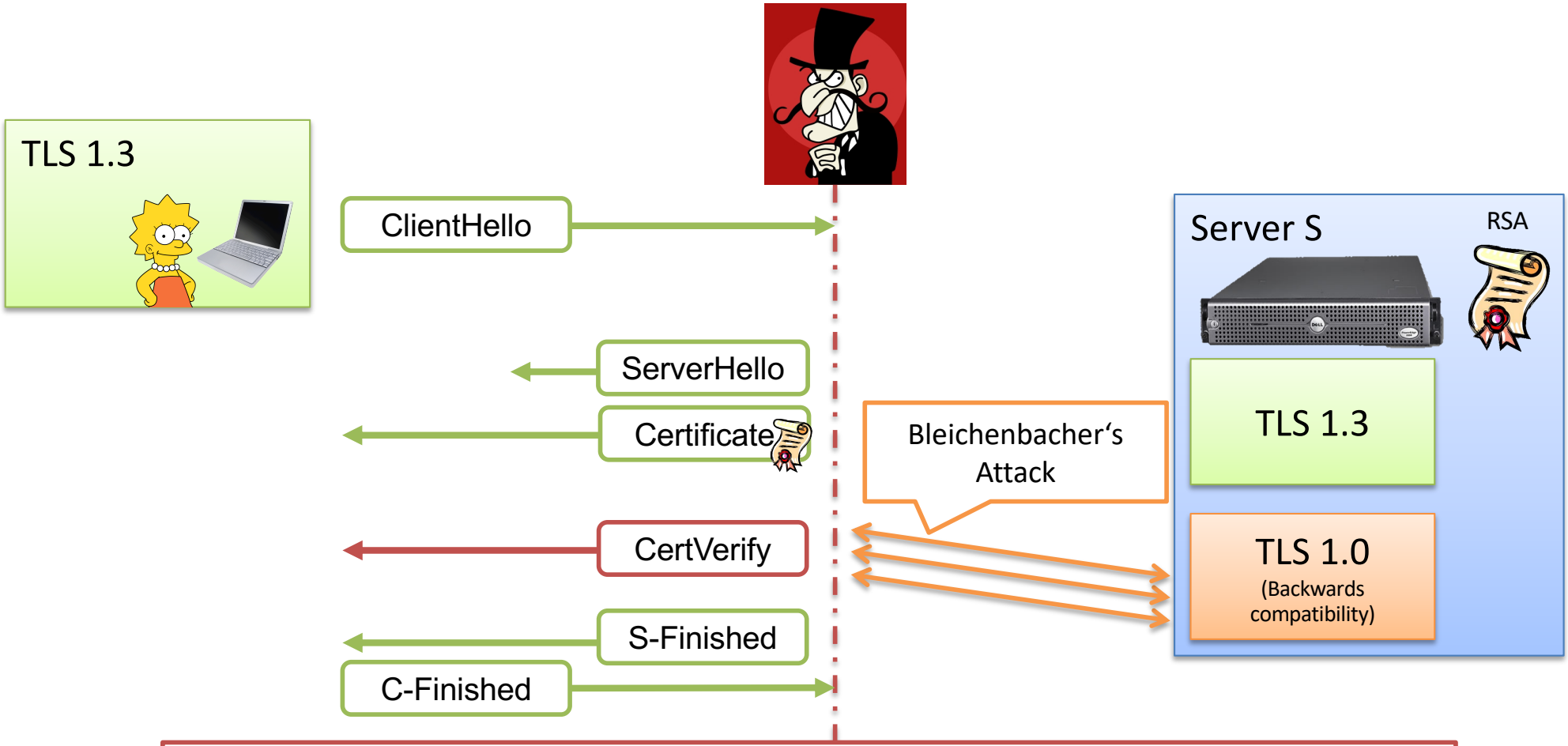
Backwards Compatibility Attack on TLS 1.3

[J., Schwenk, Somorovsky; ACM CCS 2015]



Backwards Compatibility Attack on TLS 1.3

[J., Schwenk, Somorovsky; ACM CCS 2015]



TLS 1.3 may be vulnerable to Bleichenbacher's attack,
even though PKCS#1 v1.5 encryption is not used!

Practical Impact

- Practical impact on TLS 1.3 **rather limited**
  - Typical Bleichenbacher-attacks take **hours or days**
  - Machine-to-machine communication?

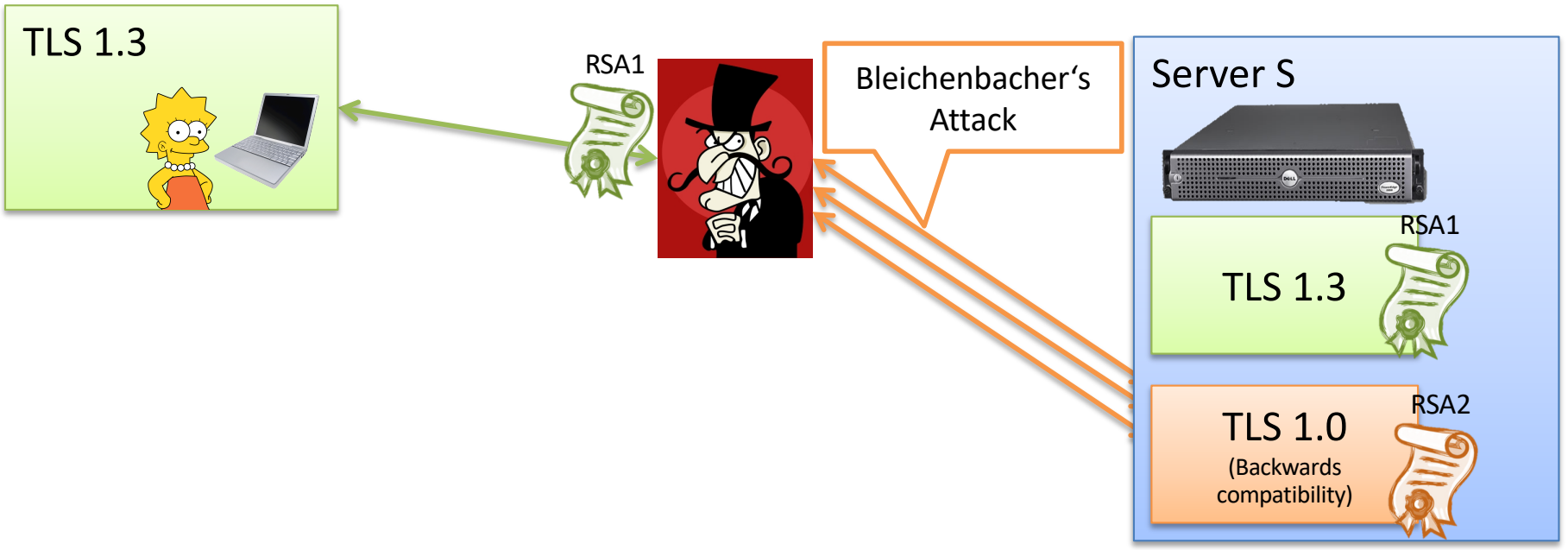
Practical Impact

- Practical impact on TLS 1.3 **rather limited**
  - Typical Bleichenbacher-attacks take **hours or days**
  - Machine-to-machine communication?
- Nevertheless:
  - **Backwards compatibility** must be considered (cf. Jager, Paterson, Somorovsky, NDSS 2013)
  - Future **improvements of Bleichenbacher's** attack? (Bardou *et al.*, CRYPTO 2012)

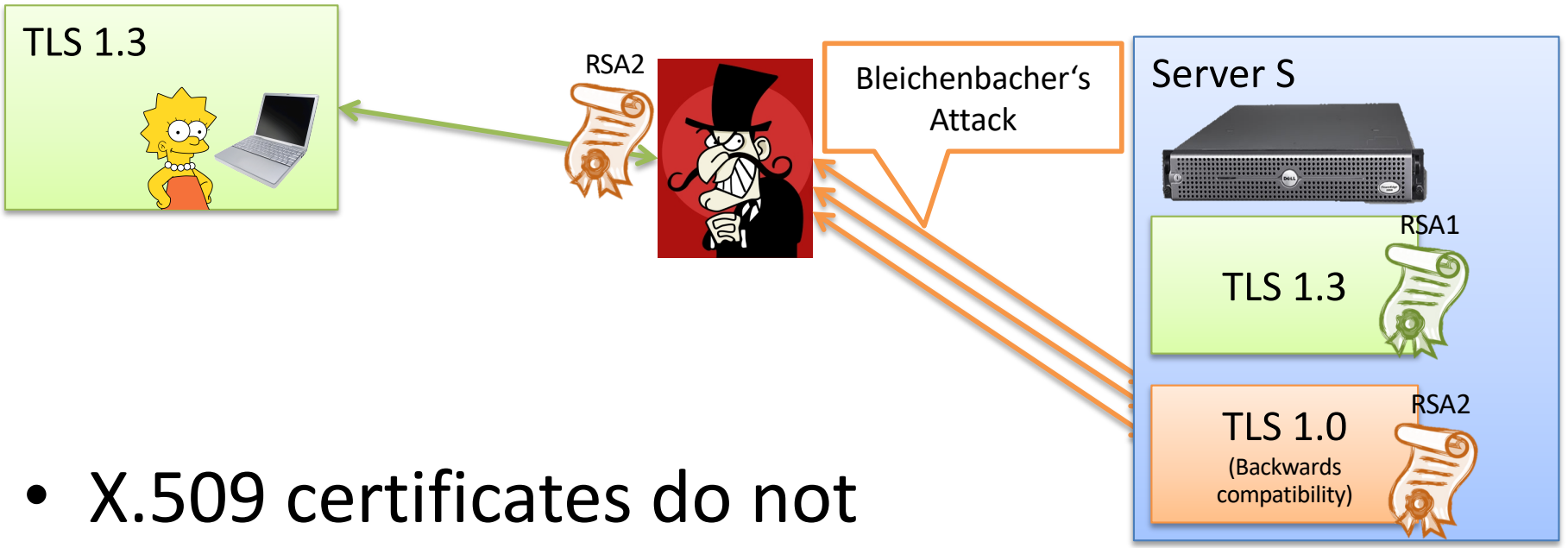
Practical Impact

- Practical impact on TLS 1.3 **rather limited**
  - Typical Bleichenbacher-attacks take **hours or days**
  - Machine-to-machine communication?
- Nevertheless:
  - **Backwards compatibility** must be considered (cf. Jager, Paterson, Somorovsky, NDSS 2013)
  - Future **improvements of Bleichenbacher's** attack? (Bardou *et al.*, CRYPTO 2012)
- Use **DROWN's approach** to **forge signature** in **one minute** on a single CPU (Aviram *et al.*, USENIX Security 2016)
  - Leverages vulnerability in openSSL
  - All openSSL versions from 1998 to early 2015
  - 26% of HTTPS servers were vulnerable

The difficulty of preventing such attacks (example)



The difficulty of preventing such attacks (example)



- X.509 certificates do not contain protocol version

Further difficulties

- X.509 supports “sign/encrypt-only” certs
  - “Sign-only” certs for “signing” cipher suites (incl. TLS 1.3)
  - “Encrypt-only” keys for TLS-RSA cipher suites

Further difficulties

- X.509 supports “sign/encrypt-only” certs
  - “Sign-only” certs for “signing” cipher suites (incl. TLS 1.3)
  - “Encrypt-only” keys for TLS-RSA cipher suites
- Key separation **not supported** by major server implementations

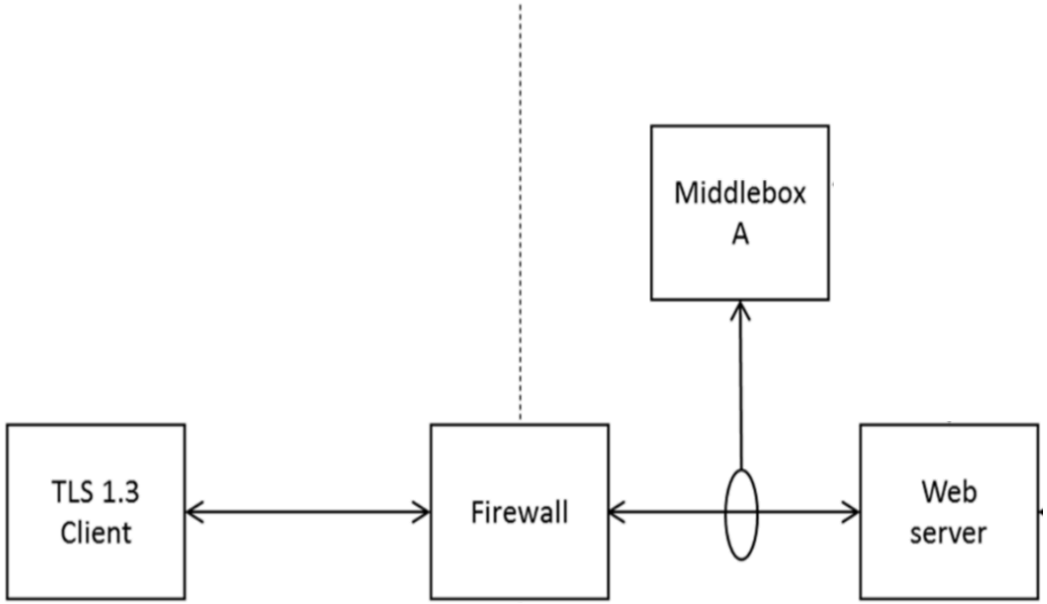
Further difficulties

- X.509 supports “sign/encrypt-only” certs
  - “Sign-only” certs for “signing” cipher suites (incl. TLS 1.3)
  - “Encrypt-only” keys for TLS-RSA cipher suites
- Key separation **not supported** by major server implementations
- **Do browsers really check this?**
  - Mozilla developer: “**No.** And we have no intention to change this, because of **usability and to maximise compatibility**.”

Take home message

- Removing RSA-PKCS#1 v1.5 from TLS was an **excellent decision**
  - Not sufficient to protect **completely** against its weaknesses
- **Proper key separation** is difficult in practice
  - Support in future versions of X.509?
  - Support by browsers?
- **Backwards compatibility requirements** must be taken into account when designing protocols!

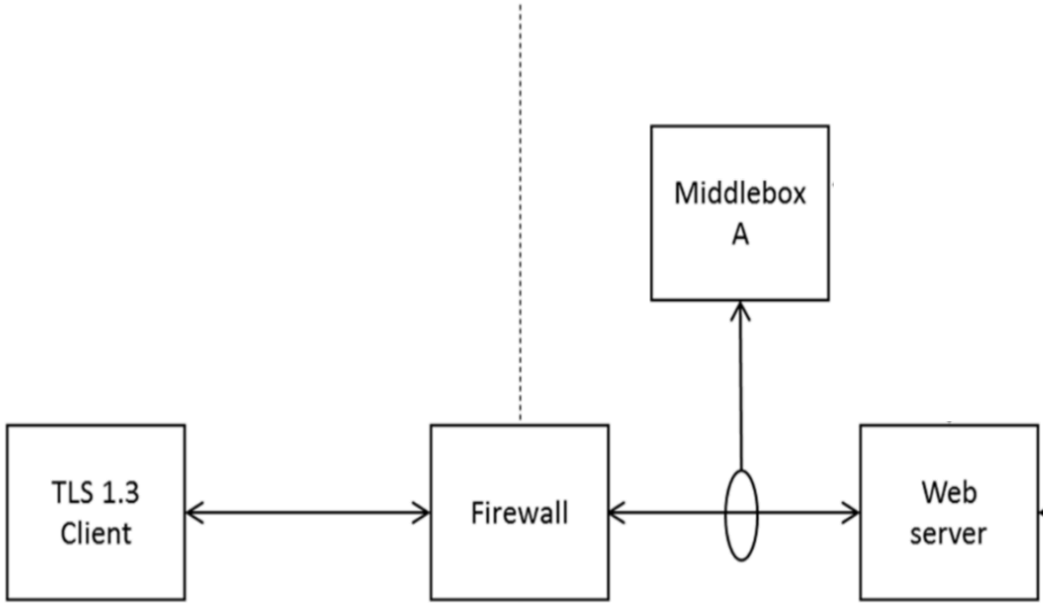
Server-Side Middleboxes



- Purpose: **traffic inspection**
- For **regulatory compliance** in some industry branches?

Figure from ETSI TS 103 523-3 V1.2.1 (03/2019)

Server-Side Middleboxes



- Purpose: **traffic inspection**
- For **regulatory compliance** in some industry branches?

- Solution: perform MITM “attack”
- Difficulty: requires ***active*** attack for TLS 1.3
  - Performance
  - Server’s secret key must be stored on middlebox

Figure from ETSI TS 103 523-3 V1.2.1 (03/2019)

Middlebox-Support of TLS 1.3?

- TLS is standardized by **IETF** (RFC 8446)
  - Support of middleboxes was requested for inclusion in TLS 1.3
  - Rejected by the TLS 1.3 working group
    - Forward security considered more important

Re: [TLS] Industry Concerns about TLS 1.3

"Paterson, Kenny" <Kenny.Paterson@rhul.ac.uk> | Thu, 22 September 2016 19:14 UTC | Show header

Hi Andrew,

My view concerning your request: no.

Rationale: We're trying to build a more secure internet.

Meta-level comment:

You're a bit late to the party. We're metaphorically speaking at the stage of emptying the ash trays and hunting for the not quite empty beer cans.

More exactly, we are at draft 15 and RSA key transport disappeared from the spec about a dozen drafts ago. I know the banking industry is usually a bit slow off the mark, but this takes the biscuit.

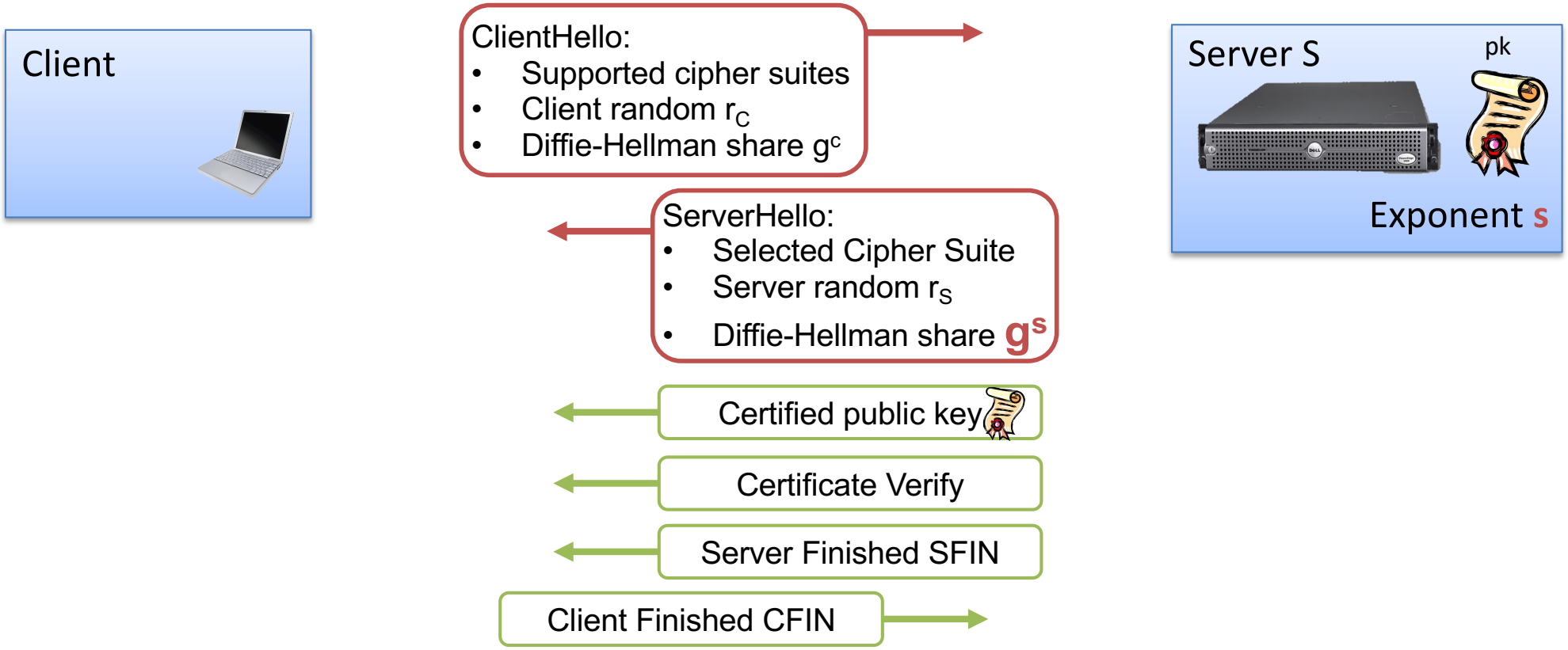
Cheers,

Kenny

Middlebox-Support of ~~TLS 1.3?~~ ETS

- ETSI: European Telecommunications Standards Institute
- Standardized variant of TLS 1.3
  - Called ETS, original name: eTLS
  - IETF objected to this use of the name TLS
  - Without proper security analysis

The Cryptographic Core of the ~~TLS 1.3~~ ETS Handshake



- Fixed exponent **s**
- "Master key" to **decrypt all sessions**
- **No forward security!**

ETSI Standard ETS (aka. eTLS)

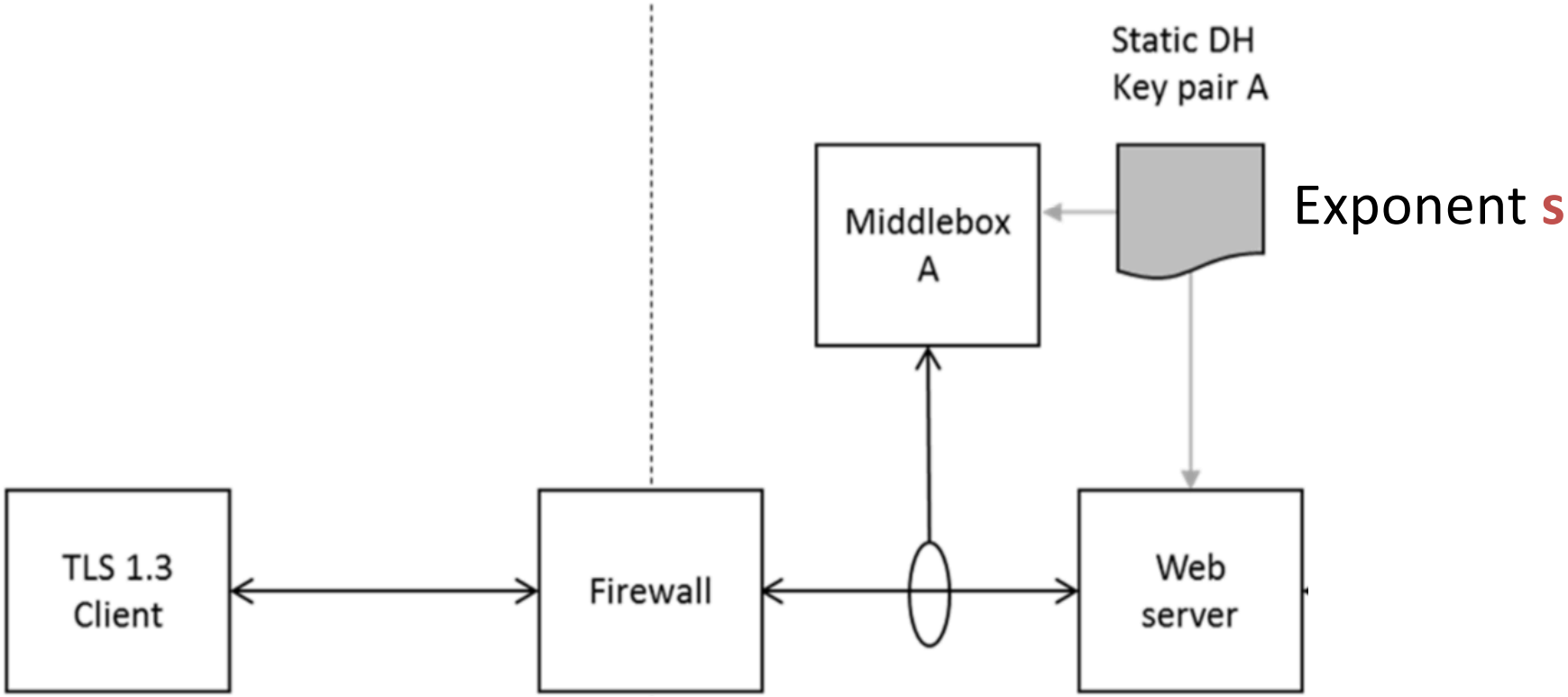


Figure from ETSI TS 103 523-3 V1.2.1 (03/2019)

A Statement of the EFF

"Don't use ETS, don't implement it, and don't standardize it."

https://www.eff.org/deeplinks/2019/02/ets-isnt-tls-and-you-shouldnt-use-it

More Fun with Middleboxes

- The protocol version field:
  - Used in TLS 1.2 (and before) to indicate the protocol version

More Fun with Middleboxes

- The protocol version field:
  - Used in TLS 1.2 (and before) to indicate the protocol version
  - **Some middleboxes fail when presented with new values**
    - TLS 1.3 disguises as TLS 1.2 (version = `0x0303`), and adds an additional `supported_versions` extension

More Fun with Middleboxes

- The protocol version field:
  - Used in TLS 1.2 (and before) to indicate the protocol version
  - **Some middleboxes fail when presented with new values**
    - TLS 1.3 disguises as TLS 1.2 (version = `0x0303`), and adds an additional `supported_versions` extension

- TLS 1.3 introduces a message type ("opaque")
  - This **confuses middleboxes** that are "accustomed to parsing previous versions"
    - TLS 1.3 disguises these messages as "`application data`"
    - The actual content type of a message can be found somewhere else

Outline

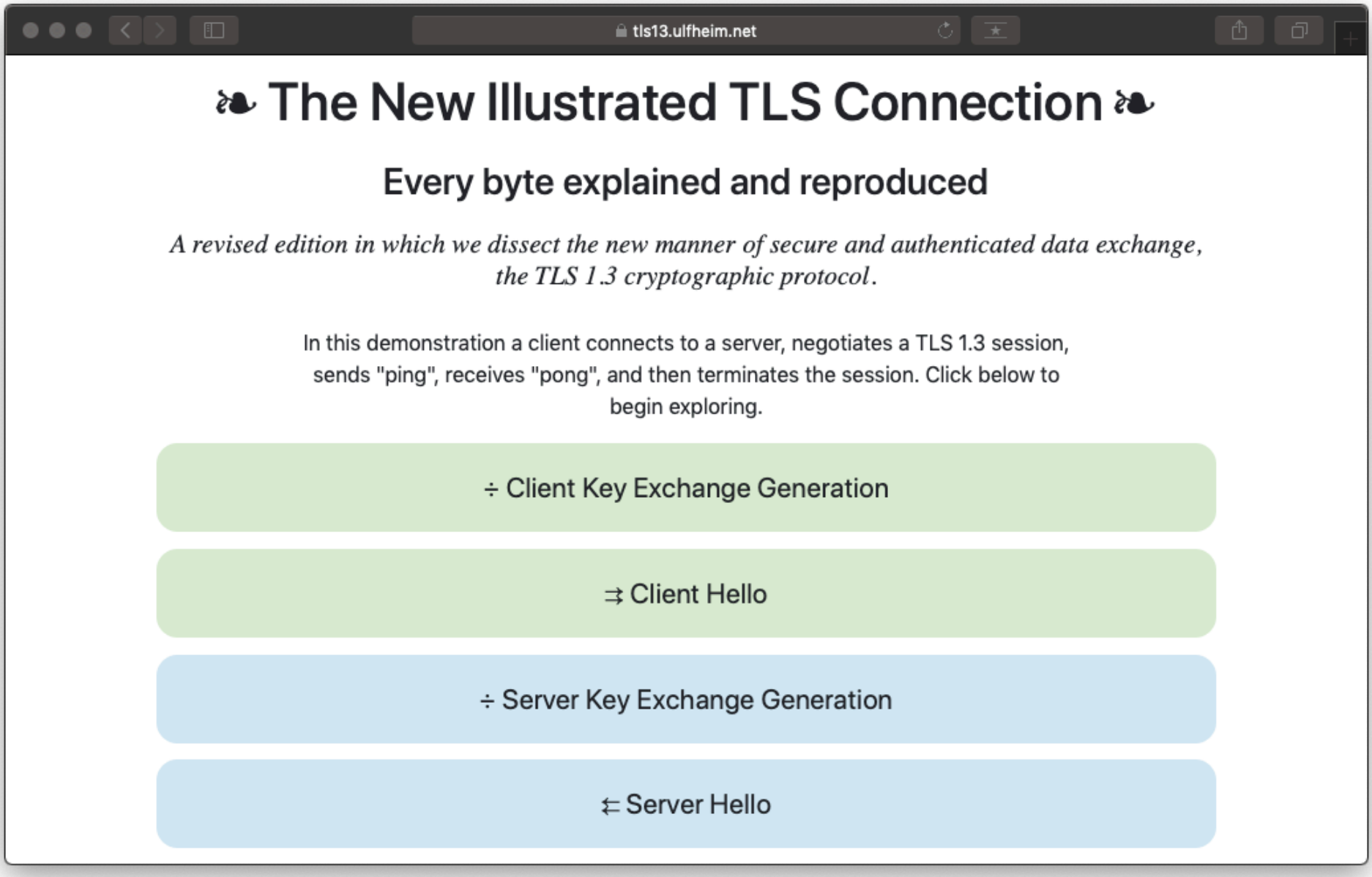
- Security of the Diffie-Hellman Key Exchange
  - Man-in-the-Middle attacks
  - Forward Security
- TLS 1.3
  - Overview
  - The cryptographic core of TLS 1.3
- Real-World Problems
  - Problems arising from backwards compatibility
  - Middleboxes and ETS
- Further reading, open research problems

Further Reading

- Dowling et al.: *A Cryptographic Analysis of the TLS 1.3 Handshake Protocol Candidates* https://eprint.iacr.org/2015/914.pdf
- Bhargavan et al.: *Implementing and Proving the TLS 1.3 Record Layer* https://eprint.iacr.org/2016/1178.pdf
- Jager et al.: *On the Security of TLS 1.3 and QUIC Against Weaknesses in PKCS#1 v1.5 Encryption* https://www.nds.ruhr-uni-bochum.de/media/nds/veroeffentlichungen/2015/08/21/Tls13QuicAttacks.pdf
- Aviram et al., *DROWN: Breaking TLS using SSLv2*, https://drownattack.com/
- Kelsey: *Compression and Information Leakage of Plaintext* https://www.iacr.org/cryptodb/archive/2002/FSE/3091/3091.pdf

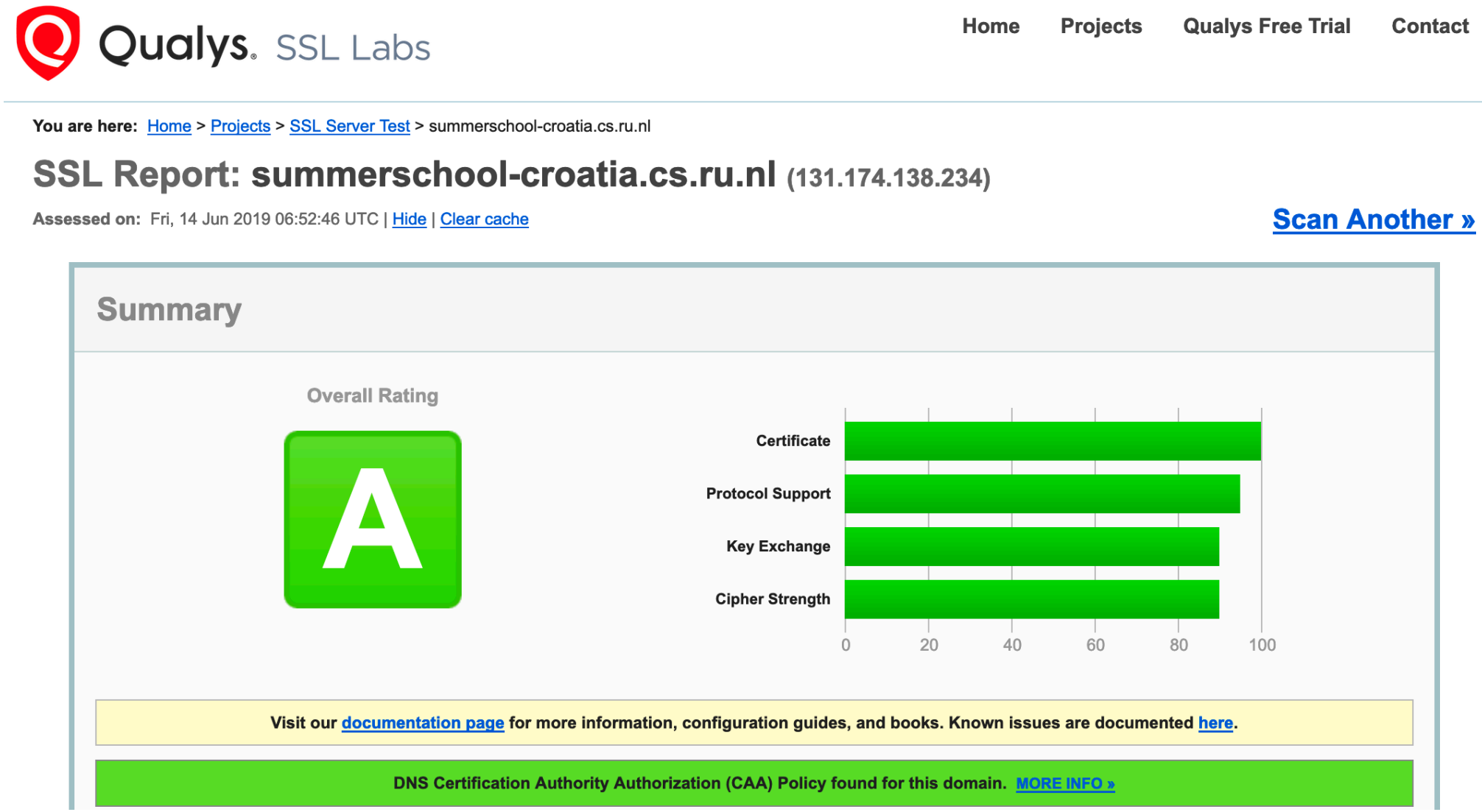
- **...and many references therein**

https://tls13.ulfheim.net/



Qualys SSL Labs

- https://www.ssllabs.com/



Interesting Research Problems

- Many AKE protocols with “provable security”
  - “Asymptotically secure”, but for reasonable concrete parameters often meaningless
  - **Efficient protocols for provably-secure *large-scale* deployments?** (Cf. [BHJKL15, GJ18, CCGJJ19])

Interesting Research Problems

- Many AKE protocols with “provable security”
  - “Asymptotically secure”, but for reasonable concrete parameters often meaningless
  - **Efficient protocols for provably-secure *large-scale* deployments?** (Cf. [BHJKL15, GJ18, CCGJJ19])
- Security against state compromise
  - “Coarse-grained”: eCK model [LLM01]
  - “Fine-grained”: considered mostly for instant messaging (**“ratcheting”**, cf. [PS18, JS18, ACD19])

Interesting Research Problems

- Many AKE protocols with "provable security"
  - "Asymptotically secure", but for reasonable concrete parameters often meaningless
  - **Efficient protocols for provably-secure *large-scale* deployments?** (Cf. [BHJKL15, GJ18, CCGJJ19])
- Security against state compromise
  - "Coarse-grained": eCK model [LLM01]
  - "Fine-grained": considered mostly for instant messaging (**"ratcheting"**, cf. [PS18, JS18, ACD19])
- **Simplify** security models and formal analysis
  - Via modularity?

tibor.jager@upb.de

@tibor_jager

Thank you for your attention!

Lecture on 0-RTT protocols:

- 0-RTT mode of TLS 1.3
- Forward-secure 0-RTT protocols
  - Seemingly impossible?
  - Basic constructions
  - Forward security for TLS 1.3 0-RTT
- Survey of recent results from EUROCRYPT 2017/2018/2019