

Provable security of advanced properties of TLS and SSH

Dr Douglas Stebila

joint work with Ben Dowling (QUT),
Florian Bergsma (né Giesen), Florian Kohlar,
Jörg Schwenk (Bochum)

IACR eprint 2012/630 (ACM CCS 2013)
IACR eprint 2013/813 (ACM CCS 2014)

2014/09/22
University of
Waterloo



Supported by:

Australian Technology Network-
German Academic Exchange
Service (ATN-DAAD) Joint
Research Cooperation Scheme

Australian Research Council
Discovery Project
DP130104304

Ben, Florian K, Florian B, Jörg, Douglas

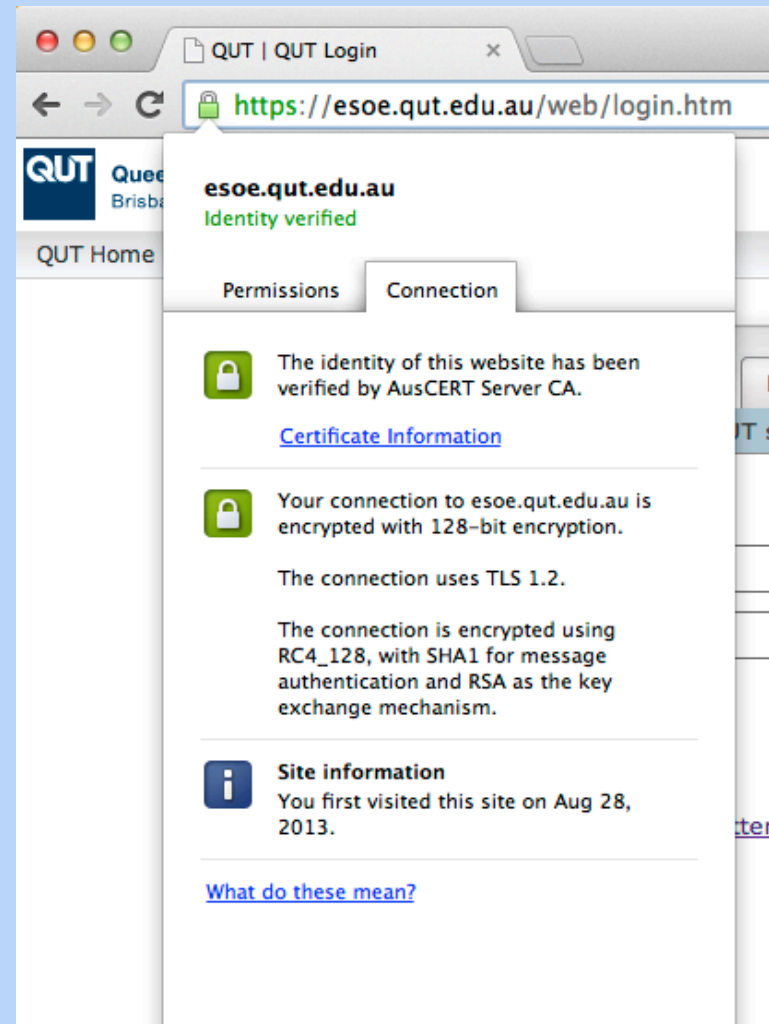


QUT

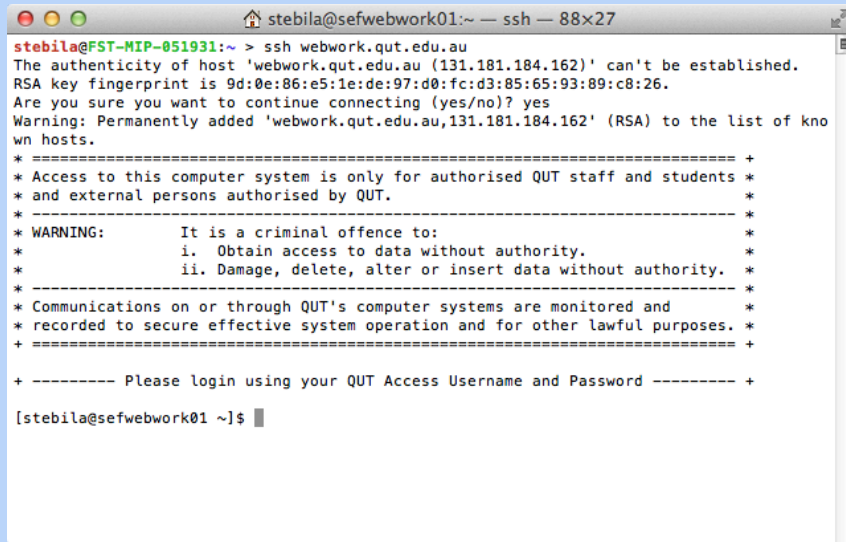


TLS (Transport Layer Security) protocol a.k.a. SSL (Secure Sockets Layer)

- The “s” in “https”
- The most important cryptographic protocol on the Internet — used to secure billions of connections every day.



SSH (Secure Shell) protocol



```
stebila@sefwebwork01:~ — ssh — 88x27
stebila@FST-MIP-051931:~ > ssh webwork.qut.edu.au
The authenticity of host 'webwork.qut.edu.au (131.181.184.162)' can't be established.
RSA key fingerprint is 9d:0e:86:e5:1e:de:97:d0:fc:d3:85:65:93:89:c8:26.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'webwork.qut.edu.au,131.181.184.162' (RSA) to the list of known hosts.
* =====+
* Access to this computer system is only for authorised QUT staff and students *
* and external persons authorised by QUT. *
* -----+
* WARNING:      It is a criminal offence to: *
*               i. Obtain access to data without authority. *
*               ii. Damage, delete, alter or insert data without authority. *
* -----+
* Communications on or through QUT's computer systems are monitored and *
* recorded to secure effective system operation and for other lawful purposes. *
* =====+
+ ----- Please login using your QUT Access Username and Password ----- +
[stebila@sefwebwork01 ~]$
```

- SSH used for secure remote access (like telnet, but secure)
- Provides public key authentication of servers and clients and encrypted communication

TLS vs. SSH

TLS

- provides secure transport for many applications
- entity authentication
- confidentiality & integrity of transmissions
- handshake establishes secure channel

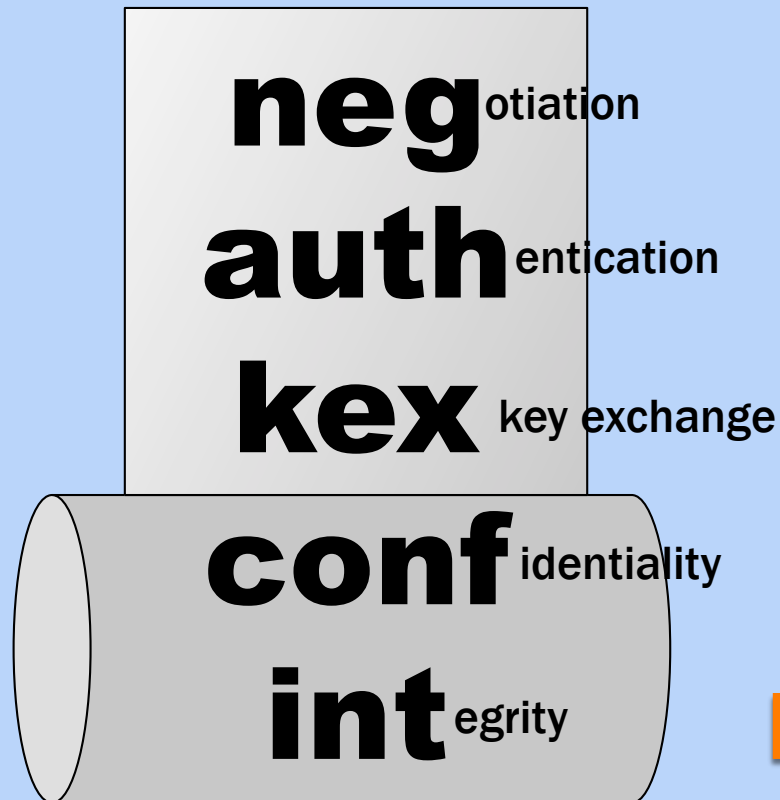
SSH

- provides secure transport primarily for remote shell logins
- entity authentication
- confidentiality & integrity of transmissions
- handshake establishes secure channel

Commonalities of TLS and SSH

Structure

Security goals



From an application perspective, TLS and SSH provide:

- **entity authentication**
- **confidentiality and integrity of messages**

+ a lot more

Outline

1. Provable security of TLS

2. TLS renegotiation

- Motivated by existing attack from 2009
- Extended security models to prove security of standardized countermeasures for TLS renegotiation

3. Multi-ciphersuite security and SSH

- Generic results on securely composing multiple protocols that share long-term keys
- First security results for full SSH protocol

4. Conclusions and opinions on secure channel definitions

Security of TLS

Structure of TLS

HANDSHAKE PROTOCOL

Negotiation of cryptographic parameters

Authentication (one-way or mutual) using public key certificates

Establishment of a master secret key

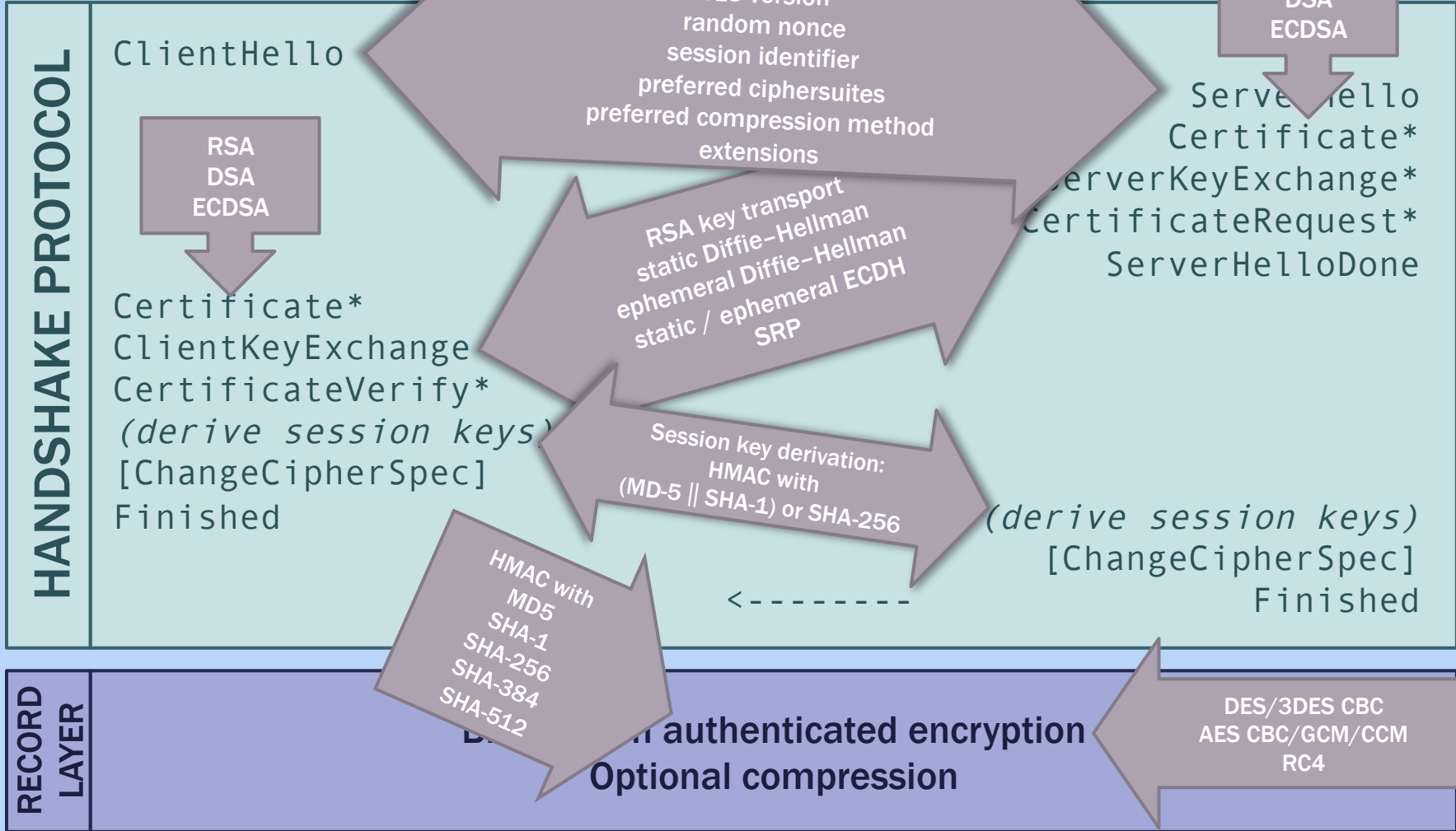
Derivation of encryption and authentication keys

Key confirmation

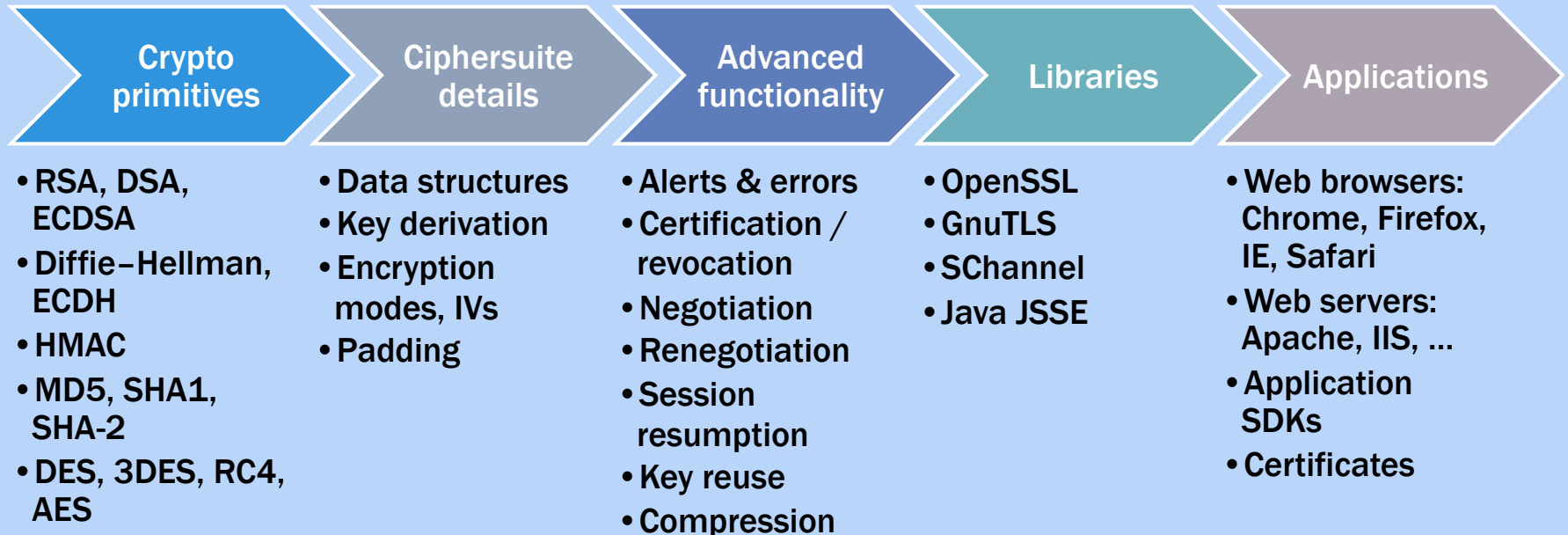
RECORD LAYER

Authenticated encryption of application data

Structure of TLS



Components of TLS



Is TLS secure?

Core cryptographic components

- Handshake protocol
 - secure authenticated key exchange protocol?
- Record layer
 - secure authenticated encryption channel?

Additional protocol functionality

- Alerts & errors?
- Certification?
- Renegotiation?
- Session resumption?
- Long-term key re-use?

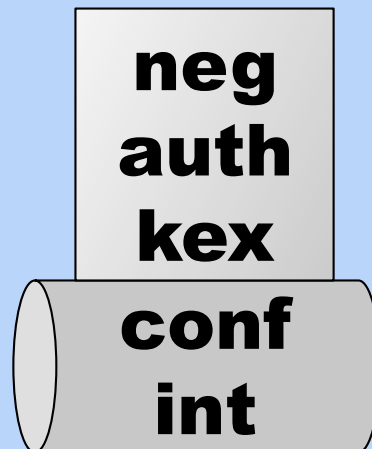
Is TLS secure?

Idea

Prove the TLS handshake is a secure authenticated key exchange protocol

- BR or CK or eCK model: adversary can't distinguish real session key from random session key

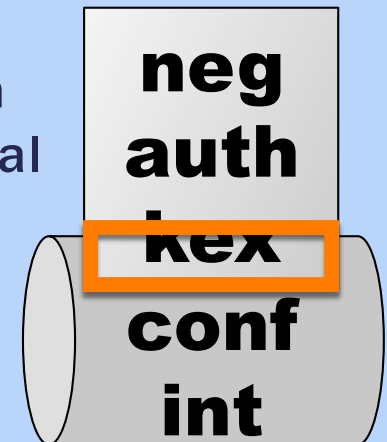
Prove the TLS record layer is a secure authenticated encryption scheme



Problem


TLS handshake sends messages encrypted under the session key

- => **overlap** between handshake and record layer
- Adversary can distinguish real session key from random



Is TLS secure?

1996
 SSL v3.0
standardized

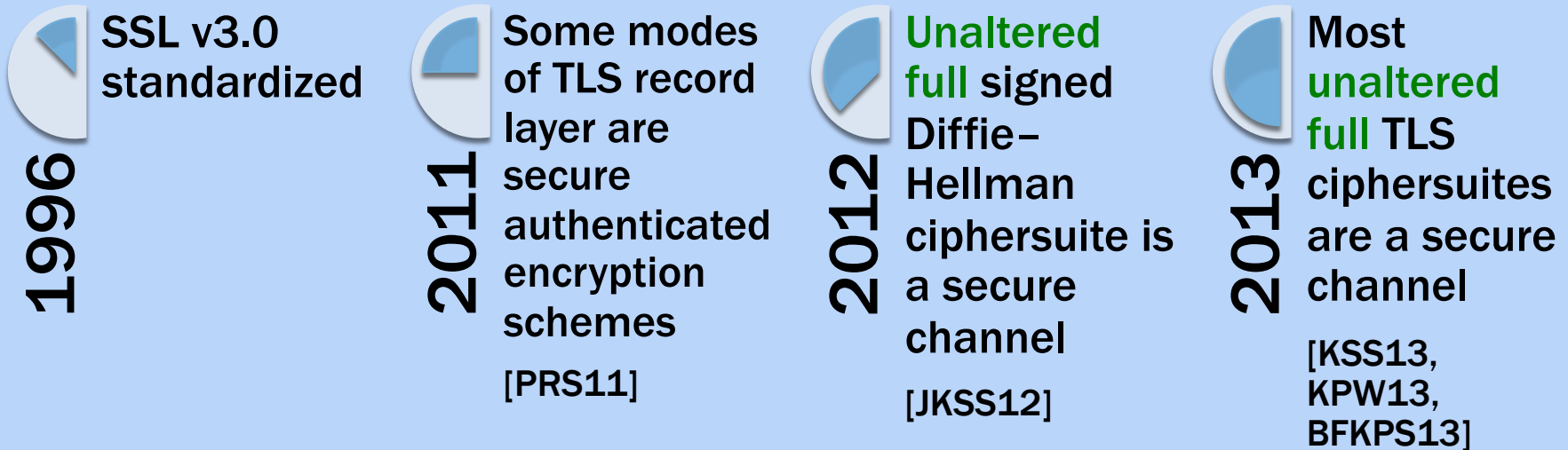
2001
 **Some
variant** of
one
ciphersuite
of the TLS
record layer
is a secure
encryption
scheme
[Kra01]

2002
 **Truncated**
TLS
handshake
using RSA
key transport
is a secure
authenticated
key exchange
protocol
[JK02]

2008
 **Truncated**
TLS
handshake
using RSA
key transport
or signed
Diffie-
Hellman is a
secure AKE
[MSW08]

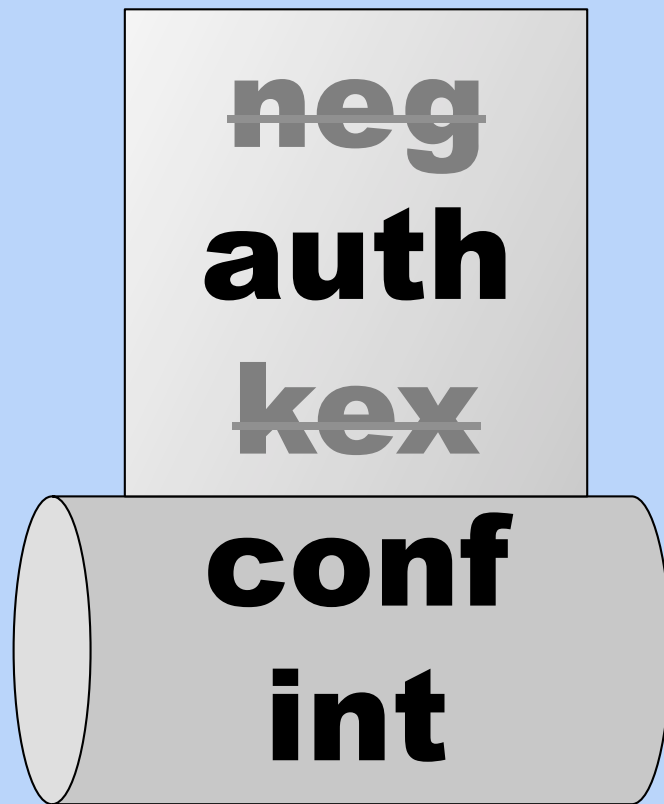
“some variant” ... “truncated TLS” ...
limited ciphersuites

Is TLS secure?



“unaltered” ... “full” ... “most ciphersuites”

Security goals of TLS and SSH



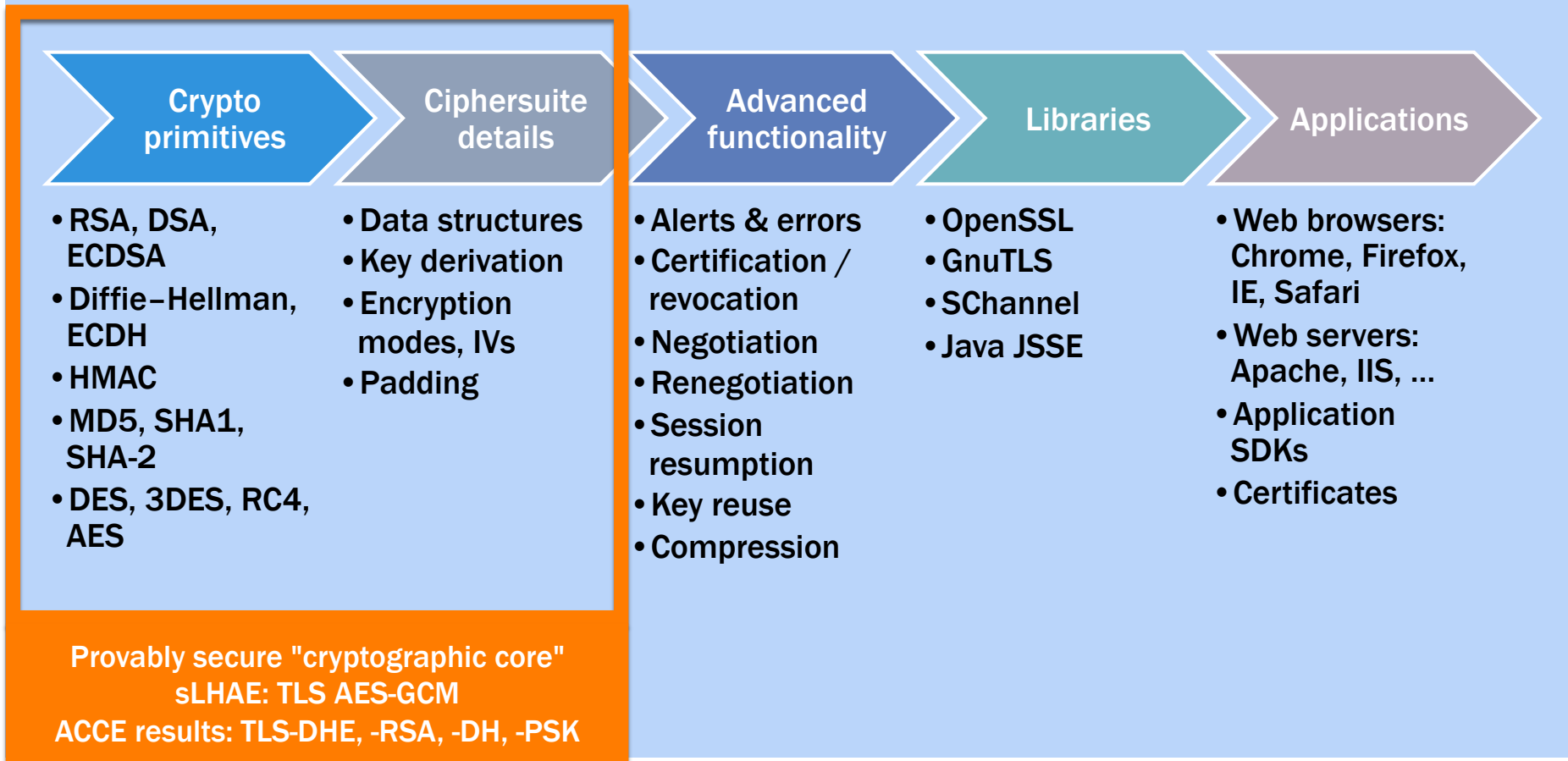
Authenticated and Confidential Channel Establishment (ACCE)

security definition

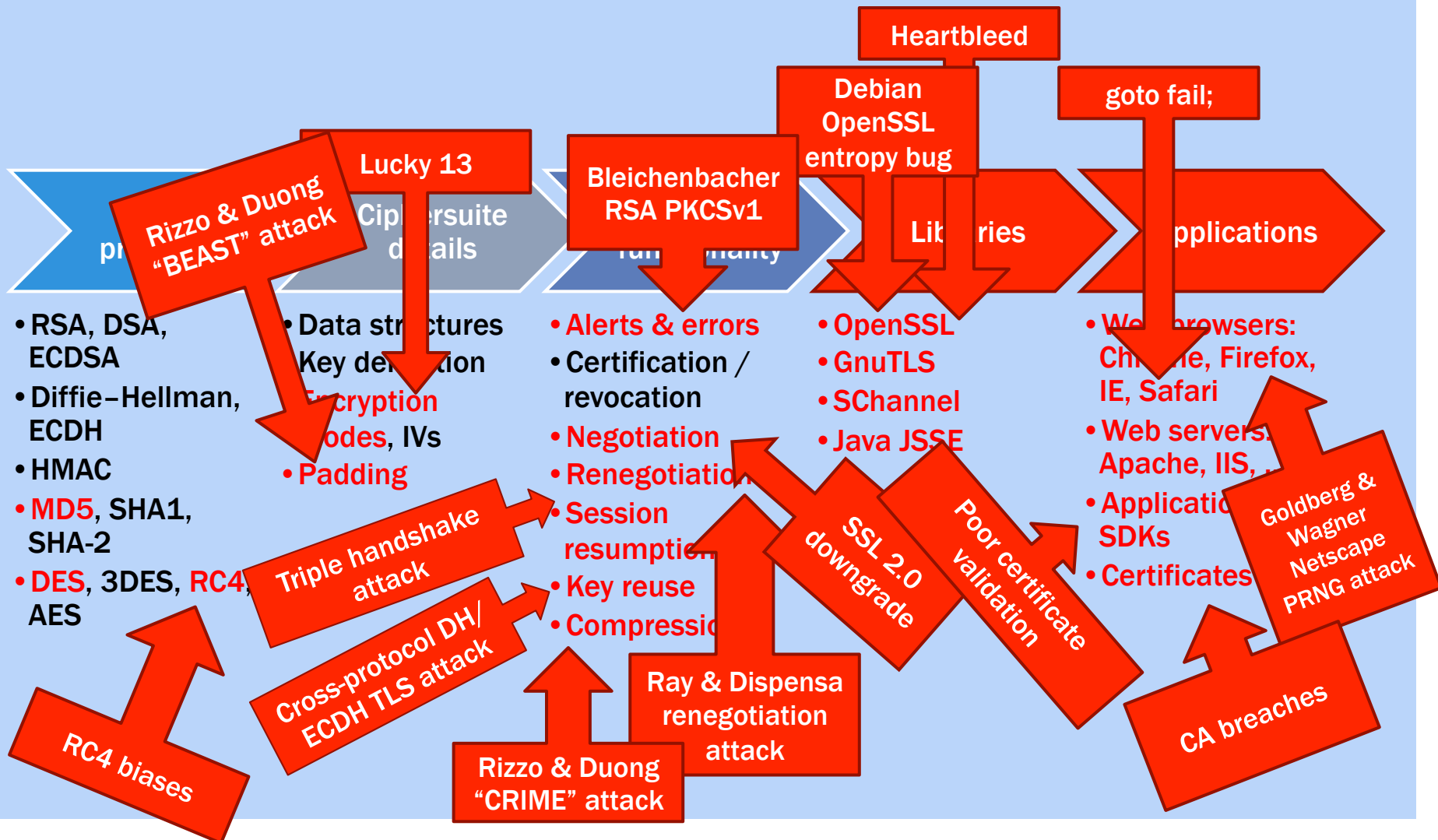
[JKSS12] captures:

- entity authentication
- confidentiality and integrity of messages

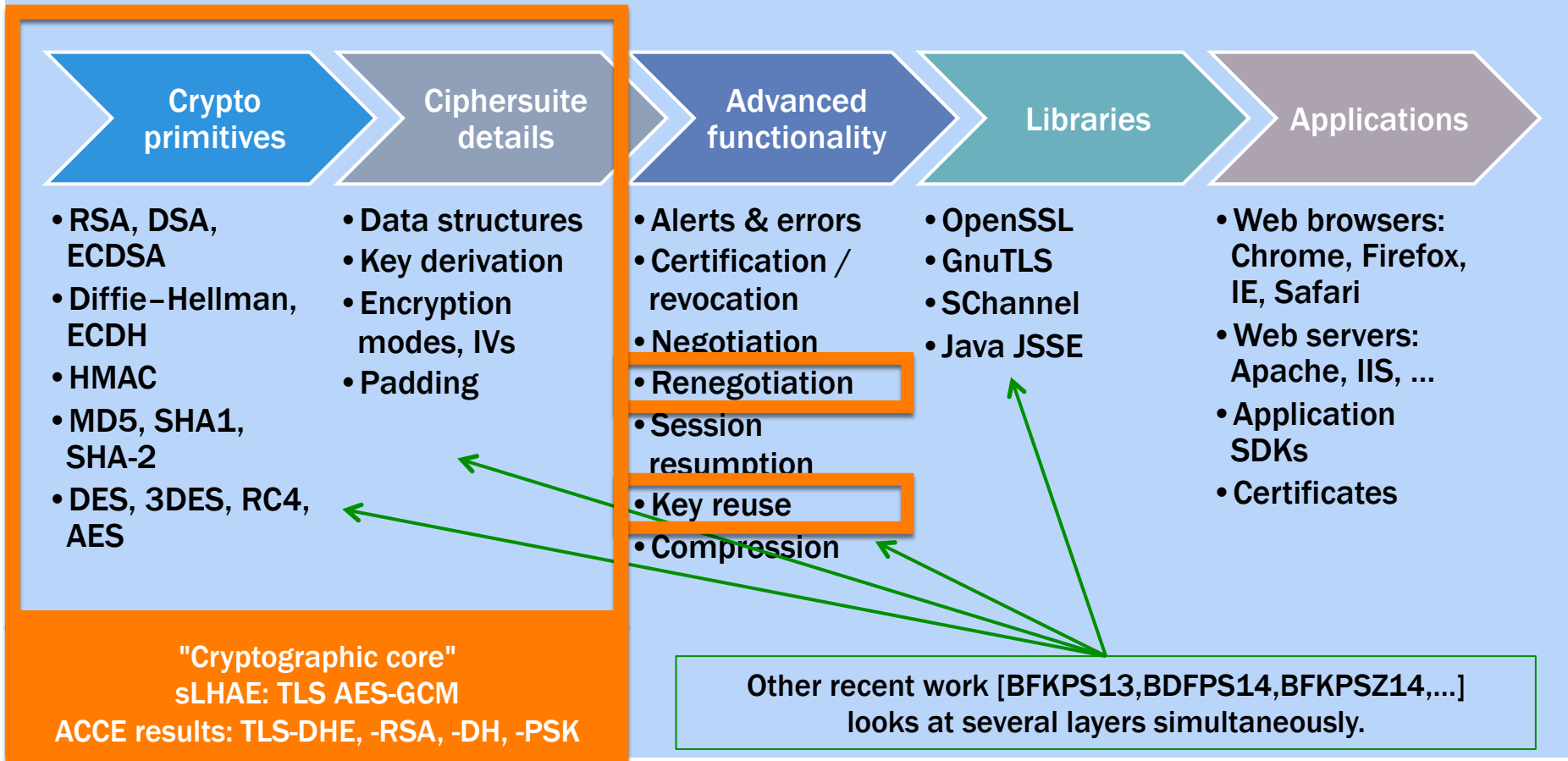
Components of TLS



Real-world attacks on TLS



Extending provable security results



TLS and renegotiation

joint work with
Florian Giesen & Florian Kohlar (Bochum)

ACM CCS
2013

IACR eprint
2012/630

Why renegotiate?

Renegotiation allows parties in an established TLS channel to create a new TLS channel that continues from the existing one.

Once you've established a TLS channel, why would you ever want to renegotiate it?

- Change cryptographic parameters
- Change authentication credentials
- Identity hiding for client
 - second handshake messages sent encrypted under first record layer
- Refresh encryption keys
 - more forward secrecy
 - record layer has maximum number of encryptions per session key

Renegotiation in TLS

(pre-November 2009)

Client

Server
(TLS)

TLS handshake₀

TLS recordlayer₀

m₀

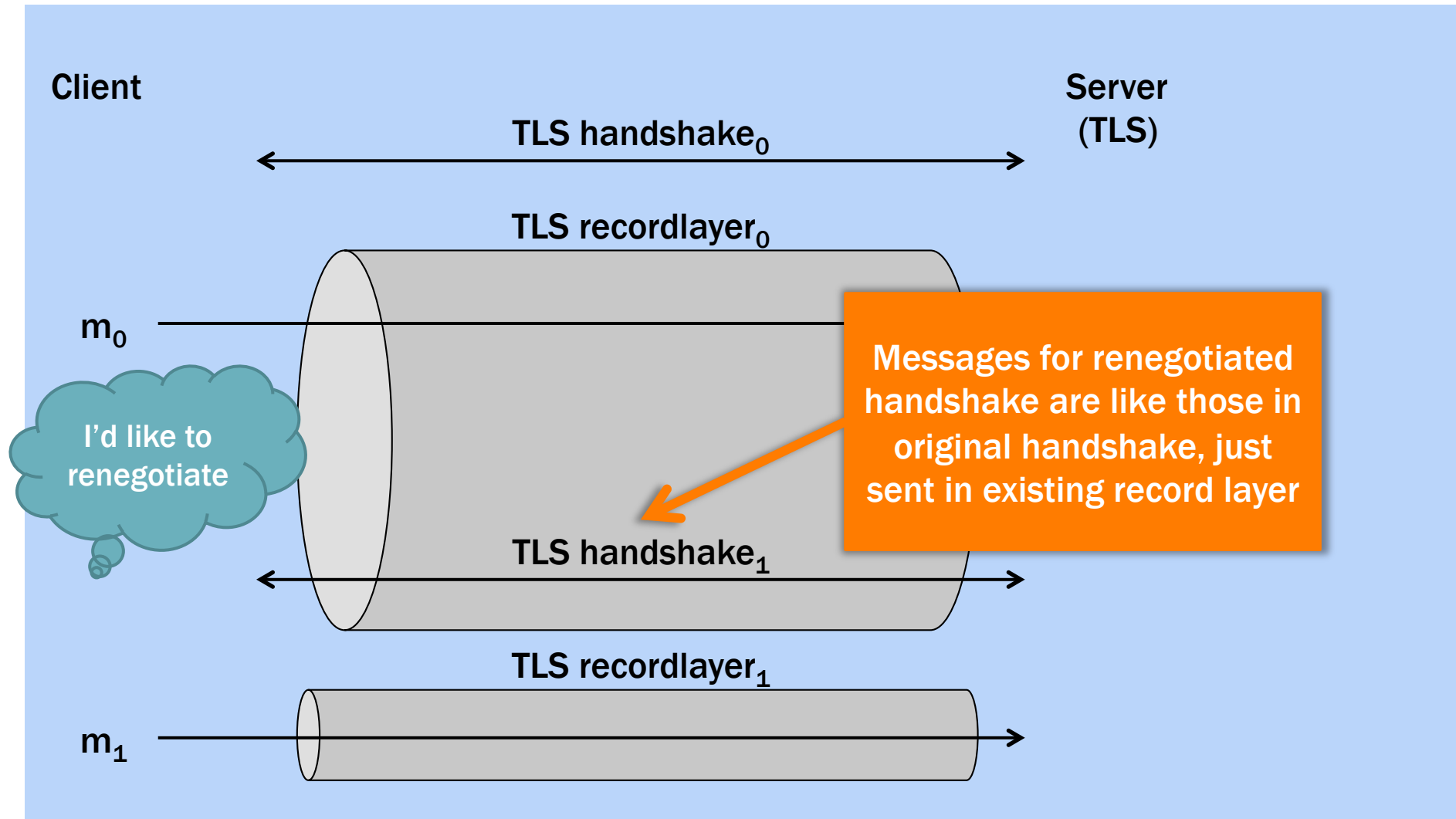
I'd like to renegotiate

Messages for renegotiated handshake are like those in original handshake, just sent in existing record layer

TLS handshake₁

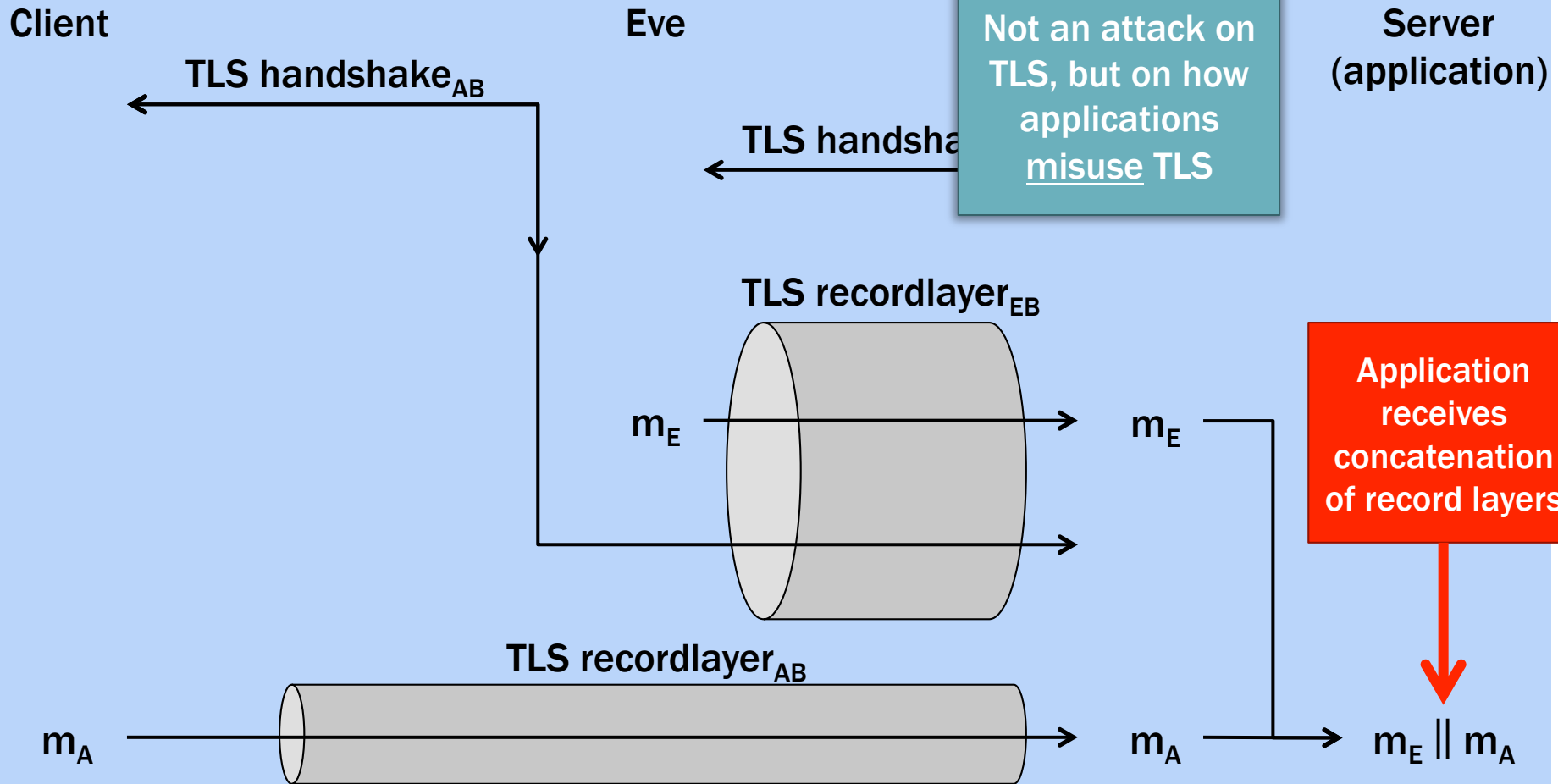
TLS recordlayer₁

m₁



TLS Renegotiation “Attack”

Ray & Dispensa, November 20



Example: HTTP Injection

- Attacker sends

- $m_E = \text{"GET /orderPizza?deliverTo=123-Fake-St}\leftarrow\rightleftharpoons\text{X-Ignore-This: "}$

- Client sends

- $m_A = \text{"GET /orderPizza?deliverTo=456-Real-St}\leftarrow\rightleftharpoons\text{Cookie: Account=1A2B"}$

- Server's web server receives

- $m_E \parallel m_A = \text{"GET /orderPizza?deliverTo=123-Fake-St}\leftarrow\rightleftharpoons\text{X-Ignore-This: GET /orderPizza?deliverTo=456-Real-St}\leftarrow\rightleftharpoons\text{Cookie: Account=1A2B"}$

X-Ignore-This: is an invalid header, so the rest of that line gets ignored.

The server's GET request is processed with the cookie supplied by the client.

Renegotiation security

Q: What property should a secure renegotiable protocol have?

A: Whenever two parties successfully renegotiate, they are assured they have the exact same view of everything that happened previously.

- **Every time we accept, we have a matching conversation of previous handshakes and record layers.**

TLS Renegotiation Countermeasures

Two related countermeasures standardized by IETF in RFC 5746:

1. Signalling Ciphersuite Value
2. Renegotiation Indication Extension

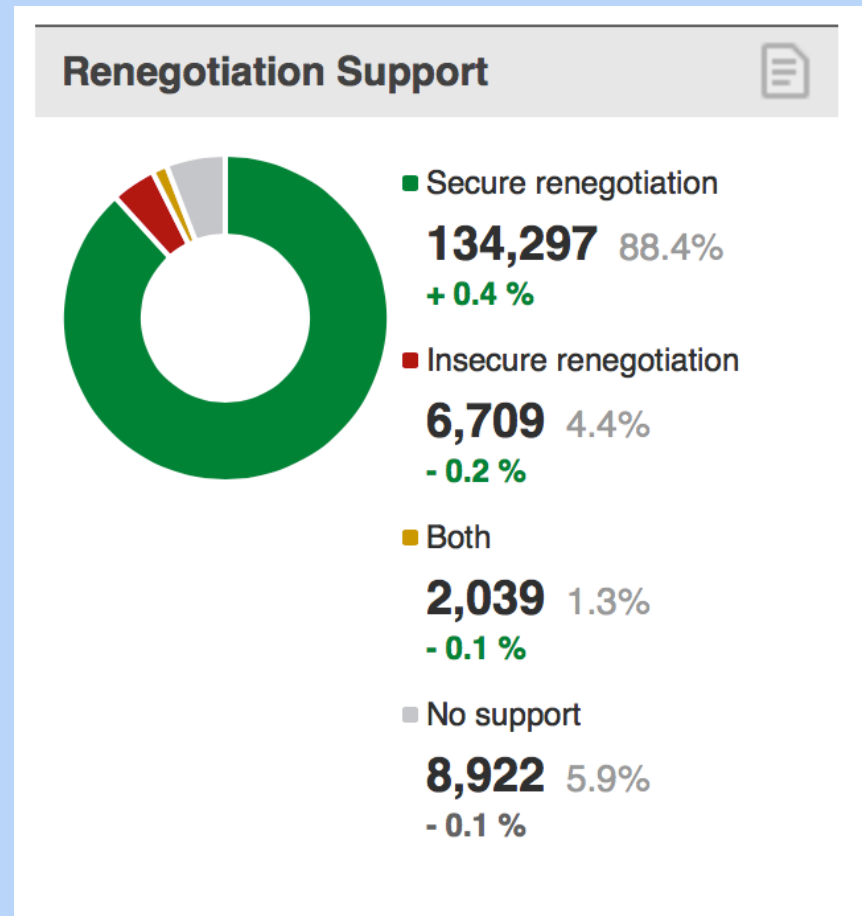
Basic idea: include fingerprint of previous handshake when renegotiating.

- Note: This is a "white-box" modification of TLS.

TLS Renegotiation Countermeasures

SCSV/RIE fairly quickly and widely adopted.

Currently 88% deployment
(SSL Pulse, Sept 3, 2014)



**Does this really fix the
problem?**

Does this really fix the problem?

Existing security definition (ACCE) isn't enough: these ciphersuites have been proven ACCE-secure yet are vulnerable to renegotiation attack.

To answer the question, need a security definition that includes renegotiation.

Secure renegotiable ACCE

Definition

When a party successfully renegotiates a new phase, its partner has a phase with a matching handshake and record layer transcript

- allowing maximal reveal of secrets

TLS

- TLS without fixes is not a secure renegotiable ACCE.
- **TLS with RFC 5746 fixes is not a secure renegotiable ACCE.**

Weakly secure renegotiable ACCE

Definition

When a party successfully renegotiate a new phase, its partner has a phase with a matching handshake and record layer transcript, *provided no previous phase's session key was revealed.*

TLS

- TLS without fixes is not a weakly secure renegotiable ACCE.
- **TLS with RFC 5746 fixes is a weakly secure renegotiable ACCE.**
 - (This is probably good enough.)

TLS renegotiation conclusions

- Renegotiation not previously included in AKE/channel security definitions.
 - Different levels of renegotiation security
- Security of a protocol in isolation doesn't imply security with renegotiation.
- Need to “open up” ACCE security definitions in order to generically transform protocols.
- Confidence in standardized TLS renegotiation fixes.

Triple handshake attack

[BDFPS14]

- Man-in-the-middle attack on three consecutive handshakes
- Relies on session resumption and renegotiation
 - works even with RIE countermeasure
- Works due to lack of binding between sessions during session resumption

Multi-ciphersuite security, TLS and SSH

joint work with Ben Dowling (QUT),
Florian Bergsma, Florian Kohlar, Jörg Schwenk (Bochum)

ACM CCS
2014

IACR eprint
2013/813

TL_NULL_WITH_NULL_NULL_TLS_RSA_WITH_NULL_MD5_TLS_RSA_WITH_NULL_SHA_TLS_RSA_EXPORT_WITH_RC4_40_MD5_TLS_RSA_WITH_RC4_128_MD5_TLS_RSA_WITH_RC4_128_SHA
TLS_RSA_EXPORT_WITH_RC2_CBC_40_MD5_TLS_RSA_WITH_IDEA_CBC_SHA_TLS_RSA_EXPORT_WITH_DES40_CBC_SHA_TLS_RSA_WITH_DES_CBC_SHA_TLS_RSA_WITH_3DES_EDE_CBC_SHA
TLS_DH_DSS_EXPORT_WITH_DES40_CBC_SHA_TLS_DH_DSS_EXPORT_WITH_DES_CBC_SHA_TLS_DH_DSS_EXPORT_WITH_3DES_EDE_CBC_SHA_TLS_DH_RSA_EXPORT_WITH_DES40_CBC_SHA
TLS_DH_RSA_EXPORT_WITH_DES40_CBC_SHA_TLS_DHE_DSS_EXPORT_WITH_DES40_CBC_SHA_TLS_DHE_DSS_EXPORT_WITH_DES_CBC_SHA_TLS_DHE_DSS_EXPORT_WITH_3DES_EDE_CBC_SHA_TLS_DHE_RSA_EXPORT_WITH_DES40_CBC_SHA
TLS_DHE_RSA_EXPORT_WITH_DES40_CBC_SHA_TLS_DHE_RSA_EXPORT_WITH_3DES_EDE_CBC_SHA_TLS_DH_anon_EXPORT_WITH_RC4_40_MD5_TLS_DH_anon_EXPORT_WITH_DES40_CBC_SHA
TLS_DH_anon_EXPORT_WITH_DES_CBC_SHA_TLS_DH_anon_EXPORT_WITH_3DES_EDE_CBC_SHA_TLS_KRB5_EXPORT_WITH_DES_CBC_SHA_TLS_KRB5_EXPORT_WITH_3DES_EDE_CBC_SHA_TLS_KRB5_EXPORT_WITH_IDEA_CBC_SHA
TLS_KRB5_EXPORT_WITH_DES_CBC_MD5_TLS_KRB5_EXPORT_WITH_3DES_EDE_CBC_MD5_TLS_KRB5_EXPORT_WITH_RC4_128_MD5_TLS_KRB5_EXPORT_WITH_IDEA_CBC_MD5_TLS_KRB5_EXPORT_WITH_DES_CBC_40_SHA
TLS_KRB5_EXPORT_WITH_RC2_CBC_40_SHA_TLS_KRB5_EXPORT_WITH_RC4_40_SHA_TLS_KRB5_EXPORT_WITH_DES_CBC_40_MD5_TLS_KRB5_EXPORT_WITH_RC2_CBC_40_MD5_TLS_KRB5_EXPORT_WITH_RC4_40_MD5
TLS_PSK_WITH_NULL_SHA_TLS_DHE_PSK_WITH_NULL_SHA_TLS_RSA_PSK_WITH_NULL_SHA_TLS_RSA_WITH_AES_128_CBC_SHA_TLS_DH_DSS_WITH_AES_128_CBC_SHA_TLS_DH_RSA_WITH_AES_128_CBC_SHA
TLS_DHE_DSS_WITH_AES_128_CBC_SHA_TLS_DHE_RSA_WITH_AES_128_CBC_SHA_TLS_DH_anon_EXPORT_WITH_AES_128_CBC_SHA_TLS_RSA_WITH_AES_256_CBC_SHA_TLS_DH_DSS_WITH_AES_256_CBC_SHA
TLS_DH_RSA_WITH_AES_256_CBC_SHA_TLS_DHE_DSS_WITH_AES_256_CBC_SHA_TLS_DHE_RSA_WITH_AES_256_CBC_SHA_TLS_DH_anon_EXPORT_WITH_AES_256_CBC_SHA_TLS_RSA_WITH_NULL_SHA256
TLS_RSA_WITH_AES_128_CBC_SHA256_TLS_RSA_WITH_AES_256_CBC_SHA256_TLS_DH_DSS_WITH_AES_128_CBC_SHA256_TLS_DH_RSA_WITH_AES_128_CBC_SHA256_TLS_DHE_DSS_WITH_AES_128_CBC_SHA256
TLS_RSA_WITH_CAMELLIA_128_CBC_SHA_TLS_DH_DSS_WITH_CAMELLIA_128_CBC_SHA_TLS_DH_RSA_WITH_CAMELLIA_128_CBC_SHA_TLS_DHE_DSS_WITH_CAMELLIA_128_CBC_SHA
TLS_DHE_RSA_WITH_CAMELLIA_128_CBC_SHA_TLS_DH_anon_EXPORT_WITH_CAMELLIA_128_CBC_SHA_TLS_DHE_RSA_WITH_AES_128_CBC_SHA256_TLS_DH_DSS_WITH_AES_256_CBC_SHA256
TLS_DH_RSA_WITH_AES_256_CBC_SHA256_TLS_DHE_DSS_WITH_AES_256_CBC_SHA256_TLS_DHE_RSA_WITH_AES_256_CBC_SHA256_TLS_DH_anon_EXPORT_WITH_AES_128_CBC_SHA256
TLS_DH_anon_EXPORT_WITH_AES_256_CBC_SHA256_TLS_RSA_WITH_CAMELLIA_256_CBC_SHA_TLS_DH_DSS_WITH_CAMELLIA_256_CBC_SHA_TLS_DH_RSA_WITH_CAMELLIA_256_CBC_SHA
TLS_DHE_DSS_WITH_CAMELLIA_256_CBC_SHA_TLS_DHE_RSA_WITH_CAMELLIA_256_CBC_SHA_TLS_DH_anon_EXPORT_WITH_CAMELLIA_256_CBC_SHA_TLS_PSK_WITH_RC4_128_SHA_TLS_PSK_WITH_3DES_EDE_CBC_SHA
TLS_PSK_WITH_AES_128_CBC_SHA_TLS_PSK_WITH_AES_256_CBC_SHA_TLS_DHE_PSK_WITH_RC4_128_SHA_TLS_DHE_PSK_WITH_3DES_EDE_CBC_SHA_TLS_DHE_PSK_WITH_AES_128_CBC_SHA
TLS_DHE_PSK_WITH_AES_256_CBC_SHA_TLS_RSA_PSK_WITH_RC4_128_SHA_TLS_RSA_PSK_WITH_3DES_EDE_CBC_SHA_TLS_RSA_PSK_WITH_AES_128_CBC_SHA_TLS_RSA_PSK_WITH_AES_256_CBC_SHA
TLS_RSA_WITH_SEED_CBC_SHA_TLS_DH_DSS_WITH_SEED_CBC_SHA_TLS_DH_RSA_WITH_SEED_CBC_SHA_TLS_DHE_DSS_WITH_SEED_CBC_SHA_TLS_DHE_RSA_WITH_SEED_CBC_SHA_TLS_DH_anon_EXPORT_WITH_SEED_CBC_SHA
TLS_RSA_WITH_AES_128_GCM_SHA256_TLS_RSA_WITH_AES_256_GCM_SHA384_TLS_DHE_RSA_WITH_AES_128_GCM_SHA256_TLS_DHE_RSA_WITH_AES_256_GCM_SHA384_TLS_DH_DSS_WITH_AES_128_GCM_SHA256
TLS_DH_RSA_WITH_AES_256_GCM_SHA384_TLS_DHE_DSS_WITH_AES_128_GCM_SHA256_TLS_DHE_DSS_WITH_AES_256_GCM_SHA384_TLS_DH_DSS_WITH_AES_128_GCM_SHA256
TLS_DH_DSS_WITH_AES_256_GCM_SHA384_TLS_DH_anon_EXPORT_WITH_AES_128_GCM_SHA256_TLS_DH_anon_EXPORT_WITH_AES_256_GCM_SHA384_TLS_PSK_WITH_AES_128_GCM_SHA256
TLS_PSK_WITH_AES_128_GCM_SHA256_TLS_PSK_WITH_AES_256_GCM_SHA384_TLS_PSK_WITH_NULL_SHA256_TLS_PSK_WITH_NULL_SHA384_TLS_DHE_PSK_WITH_AES_128_CBC_SHA256
TLS_DHE_PSK_WITH_AES_256_CBC_SHA384_TLS_DHE_PSK_WITH_NULL_SHA256_TLS_DHE_PSK_WITH_NULL_SHA384_TLS_RSA_PSK_WITH_AES_128_CBC_SHA256_TLS_RSA_PSK_WITH_AES_256_CBC_SHA384
TLS_RSA_PSK_WITH_NULL_SHA256_TLS_RSA_PSK_WITH_NULL_SHA384_TLS_RSA_WITH_CAMELLIA_128_CBC_SHA256_TLS_DH_DSS_WITH_CAMELLIA_128_CBC_SHA256_TLS_DH_RSA_WITH_CAMELLIA_128_CBC_SHA256
TLS_DHE_DSS_WITH_CAMELLIA_128_CBC_SHA256_TLS_DHE_RSA_WITH_CAMELLIA_128_CBC_SHA256_TLS_DH_anon_EXPORT_WITH_CAMELLIA_128_CBC_SHA256_TLS_RSA_WITH_CAMELLIA_256_CBC_SHA256
TLS_DH_DSS_WITH_CAMELLIA_256_CBC_SHA256_TLS_DH_RSA_WITH_CAMELLIA_256_CBC_SHA256_TLS_DHE_DSS_WITH_CAMELLIA_256_CBC_SHA256_TLS_DHE_RSA_WITH_CAMELLIA_256_CBC_SHA256
TLS_DH_anon_EXPORT_WITH_CAMELLIA_256_CBC_SHA256_TLS_EMPTY_RENEGOTIATION_INFO_SCSV_TLS_ECDH_ECDSA_WITH_NULL_SHA_TLS_ECDH_ECDSA_WITH_RC4_128_SHA_TLS_ECDH_ECDSA_WITH_3DES_EDE_CBC_SHA
TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA_TLS_ECDH_ECDSA_WITH_AES_256_CBC_SHA_TLS_ECDH_ECDSA_WITH_NULL_SHA_TLS_ECDH_ECDSA_WITH_RC4_128_SHA_TLS_ECDH_ECDSA_WITH_3DES_EDE_CBC_SHA
TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA_TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA_TLS_ECDH_RSA_WITH_NULL_SHA_TLS_ECDH_RSA_WITH_RC4_128_SHA_TLS_ECDH_RSA_WITH_3DES_EDE_CBC_SHA
TLS_ECDH_RSA_WITH_AES_128_CBC_SHA_TLS_ECDH_RSA_WITH_AES_256_CBC_SHA_TLS_ECDHE_RSA_WITH_NULL_SHA_TLS_ECDHE_RSA_WITH_RC4_128_SHA_TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA_TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA_TLS_ECDH_anon_EXPORT_WITH_NULL_SHA_TLS_ECDH_anon_EXPORT_WITH_RC4_128_SHA_TLS_ECDH_anon_EXPORT_WITH_3DES_EDE_CBC_SHA
TLS_ECDH_anon_EXPORT_WITH_AES_128_CBC_SHA_TLS_ECDH_anon_EXPORT_WITH_AES_256_CBC_SHA_TLS_SRP_SHA_WITH_3DES_EDE_CBC_SHA_TLS_SRP_SHA_RSA_WITH_3DES_EDE_CBC_SHA
TLS_SRP_SHA_DSS_WITH_3DES_EDE_CBC_SHA_TLS_SRP_SHA_WITH_AES_128_CBC_SHA_TLS_SRP_SHA_WITH_AES_256_CBC_SHA_TLS_SRP_SHA_DSS_WITH_AES_128_CBC_SHA
TLS_SRP_SHA_WITH_AES_256_CBC_SHA_TLS_SRP_SHA_RSA_WITH_AES_256_CBC_SHA_TLS_SRP_SHA_DSS_WITH_AES_256_CBC_SHA_TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256
TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384_TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA256_TLS_ECDH_ECDSA_WITH_AES_256_CBC_SHA384_TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384_TLS_ECDH_RSA_WITH_AES_128_CBC_SHA256_TLS_ECDH_RSA_WITH_AES_256_CBC_SHA384_TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256
TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384_TLS_ECDH_ECDSA_WITH_AES_128_GCM_SHA256_TLS_ECDH_ECDSA_WITH_AES_256_GCM_SHA384_TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384_TLS_ECDH_RSA_WITH_AES_128_GCM_SHA256_TLS_ECDH_RSA_WITH_AES_256_GCM_SHA384_TLS_ECDHE_PSK_WITH_RC4_128_SHA
TLS_ECDHE_PSK_WITH_3DES_EDE_CBC_SHA_TLS_ECDHE_PSK_WITH_AES_128_CBC_SHA_TLS_ECDHE_PSK_WITH_AES_256_CBC_SHA_TLS_ECDHE_PSK_WITH_AES_128_CBC_SHA256
TLS_ECDHE_PSK_WITH_AES_256_CBC_SHA384_TLS_ECDHE_PSK_WITH_NULL_SHA_TLS_ECDHE_PSK_WITH_NULL_SHA256_TLS_ECDHE_PSK_WITH_NULL_SHA384_TLS_RSA_WITH_ARIA_128_CBC_SHA256
TLS_RSA_WITH_ARIA_256_CBC_SHA384_TLS_DH_DSS_WITH_ARIA_128_CBC_SHA256_TLS_DH_DSS_WITH_ARIA_256_CBC_SHA384_TLS_DH_RSA_WITH_ARIA_128_CBC_SHA256_TLS_DH_RSA_WITH_ARIA_256_CBC_SHA384
TLS_DHE_DSS_WITH_ARIA_128_CBC_SHA256_TLS_DHE_DSS_WITH_ARIA_256_CBC_SHA384_TLS_DHE_RSA_WITH_ARIA_128_CBC_SHA256_TLS_DHE_RSA_WITH_ARIA_256_CBC_SHA384
TLS_DH_anon_EXPORT_WITH_ARIA_128_CBC_SHA256_TLS_DH_anon_EXPORT_WITH_ARIA_256_CBC_SHA384_TLS_ECDHE_ECDSA_WITH_ARIA_128_CBC_SHA256_TLS_ECDHE_ECDSA_WITH_ARIA_256_CBC_SHA384
TLS_ECDH_ECDSA_WITH_ARIA_128_CBC_SHA256_TLS_ECDH_ECDSA_WITH_ARIA_256_CBC_SHA384_TLS_ECDHE_RSA_WITH_ARIA_128_CBC_SHA256_TLS_ECDHE_RSA_WITH_ARIA_256_CBC_SHA384
TLS_ECDH_RSA_WITH_ARIA_128_CBC_SHA256_TLS_ECDH_RSA_WITH_ARIA_256_CBC_SHA384_TLS_RSA_WITH_ARIA_128_GCM_SHA256_TLS_RSA_WITH_ARIA_256_GCM_SHA384
TLS_DHE_RSA_WITH_ARIA_128_GCM_SHA256_TLS_DHE_RSA_WITH_ARIA_256_GCM_SHA384_TLS_DH_RSA_WITH_ARIA_128_GCM_SHA256_TLS_DH_RSA_WITH_ARIA_256_GCM_SHA384
TLS_DHE_DSS_WITH_ARIA_128_GCM_SHA256_TLS_DHE_DSS_WITH_ARIA_256_GCM_SHA384_TLS_DH_DSS_WITH_ARIA_128_GCM_SHA256_TLS_DH_DSS_WITH_ARIA_256_GCM_SHA384
TLS_DH_anon_EXPORT_WITH_ARIA_128_GCM_SHA256_TLS_DH_anon_EXPORT_WITH_ARIA_256_GCM_SHA384_TLS_ECDHE_ECDSA_WITH_ARIA_128_GCM_SHA256_TLS_ECDHE_ECDSA_WITH_ARIA_256_GCM_SHA384
TLS_ECDH_ECDSA_WITH_ARIA_128_GCM_SHA256_TLS_ECDH_ECDSA_WITH_ARIA_256_GCM_SHA384_TLS_ECDHE_RSA_WITH_ARIA_128_GCM_SHA256_TLS_ECDHE_RSA_WITH_ARIA_256_GCM_SHA384
TLS_ECDH_RSA_WITH_ARIA_128_GCM_SHA256_TLS_ECDH_RSA_WITH_ARIA_256_GCM_SHA384_TLS_PSK_WITH_ARIA_128_CBC_SHA256_TLS_PSK_WITH_ARIA_256_CBC_SHA384
TLS_DHE_PSK_WITH_ARIA_128_CBC_SHA256_TLS_DHE_PSK_WITH_ARIA_256_CBC_SHA384_TLS_RSA_PSK_WITH_ARIA_128_CBC_SHA256_TLS_RSA_PSK_WITH_ARIA_256_CBC_SHA384
TLS_PSK_WITH_ARIA_128_GCM_SHA256_TLS_PSK_WITH_ARIA_256_GCM_SHA384_TLS_DHE_PSK_WITH_ARIA_128_GCM_SHA256_TLS_DHE_PSK_WITH_ARIA_256_GCM_SHA384
TLS_RSA_PSK_WITH_ARIA_128_GCM_SHA256_TLS_RSA_PSK_WITH_ARIA_256_GCM_SHA384_TLS_ECDHE_PSK_WITH_ARIA_128_CBC_SHA256_TLS_ECDHE_PSK_WITH_ARIA_256_CBC_SHA384
TLS_ECDHE_ECDSA_WITH_CAMELLIA_128_CBC_SHA256_TLS_ECDHE_ECDSA_WITH_CAMELLIA_256_CBC_SHA384_TLS_ECDH_ECDSA_WITH_CAMELLIA_128_CBC_SHA256
TLS_ECDH_ECDSA_WITH_CAMELLIA_256_CBC_SHA384_TLS_ECDHE_RSA_WITH_CAMELLIA_128_CBC_SHA256_TLS_ECDHE_RSA_WITH_CAMELLIA_256_CBC_SHA384_TLS_ECDH_RSA_WITH_CAMELLIA_128_CBC_SHA256
TLS_ECDH_RSA_WITH_CAMELLIA_256_CBC_SHA384_TLS_RSA_WITH_CAMELLIA_128_GCM_SHA256_TLS_RSA_WITH_CAMELLIA_256_GCM_SHA384_TLS_DHE_RSA_WITH_CAMELLIA_128_GCM_SHA256
TLS_DHE_RSA_WITH_CAMELLIA_256_GCM_SHA384_TLS_DH_RSA_WITH_CAMELLIA_128_GCM_SHA256_TLS_DH_RSA_WITH_CAMELLIA_256_GCM_SHA384_TLS_DHE_DSS_WITH_CAMELLIA_128_GCM_SHA256
TLS_DHE_DSS_WITH_CAMELLIA_256_GCM_SHA384_TLS_DH_DSS_WITH_CAMELLIA_128_GCM_SHA256_TLS_DH_DSS_WITH_CAMELLIA_256_GCM_SHA384_TLS_DH_anon_EXPORT_WITH_CAMELLIA_128_GCM_SHA256
TLS_DH_anon_EXPORT_WITH_CAMELLIA_256_GCM_SHA384_TLS_ECDHE_ECDSA_WITH_CAMELLIA_128_GCM_SHA256_TLS_ECDHE_ECDSA_WITH_CAMELLIA_256_GCM_SHA384
TLS_ECDH_ECDSA_WITH_CAMELLIA_128_GCM_SHA256_TLS_ECDH_ECDSA_WITH_CAMELLIA_256_GCM_SHA384_TLS_ECDHE_RSA_WITH_CAMELLIA_128_GCM_SHA256_TLS_ECDHE_RSA_WITH_CAMELLIA_256_GCM_SHA384
TLS_ECDH_RSA_WITH_CAMELLIA_128_GCM_SHA256_TLS_ECDH_RSA_WITH_CAMELLIA_256_GCM_SHA384_TLS_PSK_WITH_CAMELLIA_128_GCM_SHA256_TLS_PSK_WITH_CAMELLIA_256_GCM_SHA384
TLS_DHE_PSK_WITH_CAMELLIA_128_GCM_SHA256_TLS_DHE_PSK_WITH_CAMELLIA_256_GCM_SHA384_TLS_RSA_PSK_WITH_CAMELLIA_128_GCM_SHA256_TLS_RSA_PSK_WITH_CAMELLIA_256_GCM_SHA384
TLS_PSK_WITH_CAMELLIA_128_CBC_SHA256_TLS_PSK_WITH_CAMELLIA_256_CBC_SHA384_TLS_DHE_PSK_WITH_CAMELLIA_128_CBC_SHA256_TLS_DHE_PSK_WITH_CAMELLIA_256_CBC_SHA384
TLS_RSA_PSK_WITH_CAMELLIA_128_CBC_SHA256_TLS_RSA_PSK_WITH_CAMELLIA_256_CBC_SHA384_TLS_ECDHE_PSK_WITH_CAMELLIA_128_CBC_SHA256_TLS_ECDHE_PSK_WITH_CAMELLIA_256_CBC_SHA384
TLS_RSA_WITH_AES_128_CCM_TLS_RSA_WITH_AES_256_CCM_TLS_DHE_RSA_WITH_AES_128_CCM_TLS_DHE_RSA_WITH_AES_256_CCM_TLS_RSA_WITH_AES_128_GCM_SHA256_TLS_RSA_WITH_AES_256_CCM_8
TLS_DHE_RSA_WITH_AES_128_CCM_8_TLS_DHE_RSA_WITH_AES_256_CCM_8_TLS_PSK_WITH_AES_128_CCM_TLS_PSK_WITH_AES_256_CCM_TLS_DHE_PSK_WITH_AES_128_CCM_TLS_DHE_PSK_WITH_AES_256_CCM_8
TLS_PSK_WITH_AES_128_CCM_8_TLS_PSK_WITH_AES_256_CCM_8_TLS_PSK_DHE_WITH_AES_128_CCM_8_TLS_PSK_DHE_WITH_AES_256_CCM_8

List of all 314 TLS ciphersuites

List of SSH ciphersuites

■ Authentication:

- RSA signatures
- DSA-SHA1
- ECDSA-SHA2
- X509-RSA signatures
- X509-DSA-SHA1
- X509-ECDSA-SHA2

■ Key exchange:

- DH explicit group SHA1
- DH explicit group SHA2
- DH group 1 SHA1
- DH group 14 SHA1
- ECDH-nistp256-SHA2
- ECDH-nistp384-SHA2
- ECDH-nistp521-SHA2
- ECDH-*-SHA2
- GSS-group1-SHA1-*
- GSS-group14-SHA1-*
- GSS explicit group SHA1
- RSA1024-SHA1
- RSA2048-SHA2
- ECMQV-*-SHA2

■ Encryption:

- 3des-cbc
- blowfish-cbc
- twofish256-cbc
- twofish-cbc
- twofish192-cbc
- twofish128-cbc
- aes256-cbc
- aes192-cbc
- aes128-cbc

- serpent256-cbc
- serpent192-cbc
- serpent128-cbc
- arcfour
- idea-cbc
- cast128-cbc
- des-cbc
- arcfour128
- arcfour256
- aes128-ctr
- aes192-ctr
- aes256-ctr
- 3des-ctr
- blowfish-ctr
- twofish128-ctr
- twofish192-ctr
- twofish256-ctr
- serpent128-ctr
- serpent192-ctr
- serpent256-ctr
- idea-ctr
- cast128-ctr
- AEAD_AES_128_GCM
- AEAD_AES_256_GCM

■ MACs:

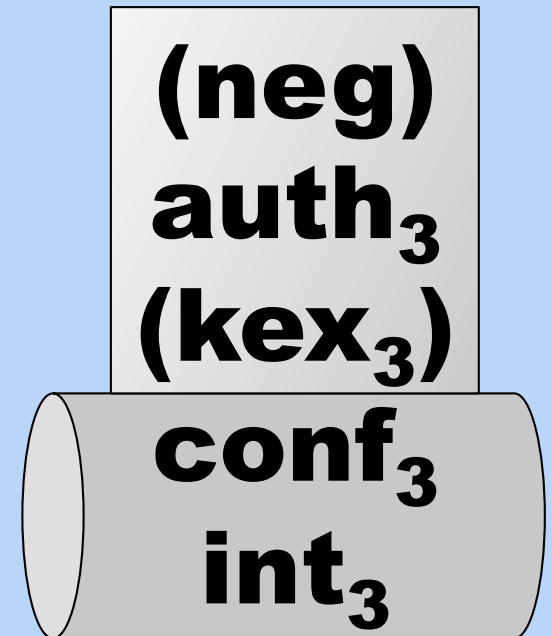
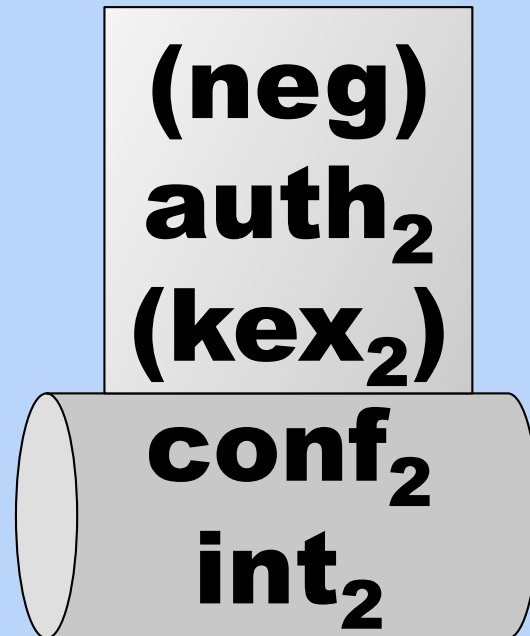
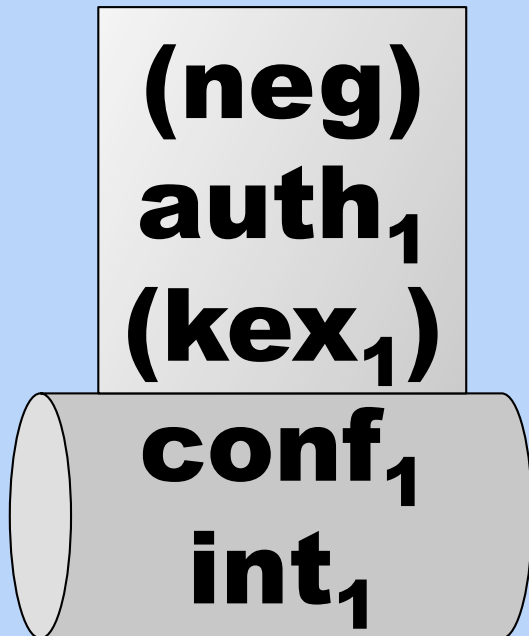
- hmac-sha1
- hmac-sha1-96
- hmac-md5
- hmac-md5-96
- AEAD_AES_128_GCM
- AEAD_AES_256_GCM
- hmac-sha2-256
- hmac-sha2-512

How we'd like to analyze ciphersuites

ciphersuite 1

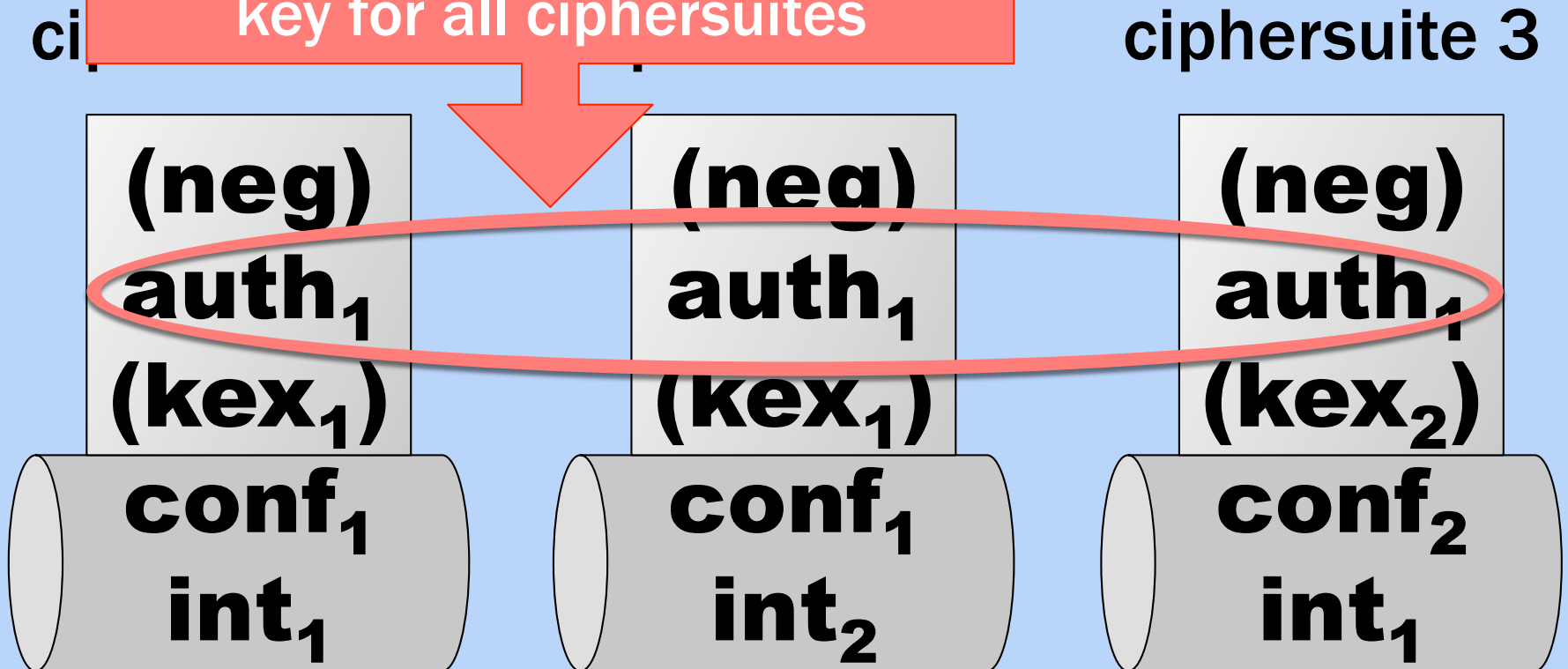
ciphersuite 2

ciphersuite 3



The reality of multi-ciphersuite usage

In practice, TLS and SSH servers use the same long-term key for all ciphersuites



Long-term key reuse across ciphersuites

Is this secure?

Even if a ciphersuite is provably secure on its own, it may not be secure if the long-term key is shared between two ciphersuites.

Long-term keys in TLS

Most TLS ciphersuites are provably secure channels (ACCE).

But this assumes that each ciphersuite uses its own distinct long-term key.

[MVVP12] Cross-ciphersuite attack

(built on observation of Wagner & Schneier 1996)

```
struct {
  select (KeyExchangeAlgorithm):
    case dhe_dss:
    case dhe_rsa:
      ServerDHParams params;
      digitally-signed struct {
        opaque client_random[32];
        opaque server_random[32];
        ServerDHParams params;
      } signed_params;
    case ec_diffie_hellman:
      ServerECDHParams params;
      digitally-signed struct {
        opaque client_random[32];
        opaque server_random[32];
        ServerECDHParams params;
      } signed_params;
  } ServerKeyExchange
```

1. No "type" information.

```
struct {
  opaque dh_p<1..216-1>;
  opaque dh_g<1..216-1>;
  opaque dh_Ys<1..216-1>;
} ServerDHParams;
```

```
struct {
  ECCurveType curve_type = explicit_prime(1);
  opaque prime_p <1..28-1>;
  ECCurve curve;
  ECPPoint base;
  opaque order <1..28-1>;
  opaque cofactor <1..28-1>;
  opaque point <1..28-1>;
} ServerECDHParams;
```

2. Some valid ServerECDHParams binary strings are also valid WEAK ServerDHParams binary strings.

[MVVP12] Cross-ciphersuite attack

(built on observation of Wagner & Schneier 1996)

=> TLS not secure with long-term key reuse.

=> ACCE security of a ciphersuite in isolation does not imply security with long-term key reuse.

Long-term keys in SSH

In SSH, the thing that is signed contains an unambiguous identification of the intended ciphersuite.

We might hope to be able to prove SSH secure even with key reuse across ciphersuites.

Is SSH secure?

2006
 SSH v2
standardized

2004
 **Some
variant** of
SSH
encryption is
secure
[BKN04]

2009-10
 Attack on
SSH
encryption,
fixed version
is secure
[APW09, PW10]

2011
 **Truncated**
SSH
handshake
using signed
Diffie-
Hellman is a
secure AKE
[Wil11]

“some variant” ... “truncated SSH”

Signed-DH SSH is a secure ACCE

Theorem: Assuming

- the signature scheme is secure,
- the CDH problem is hard,
- the hash function is random,
- and the encryption scheme is a secure buffered stateful authenticated encryption scheme,

then signed-DH SSH is a secure ACCE protocol.

How can we prove it secure even with long-term key reuse across ciphersuites?

- different CDH groups, different encryption schemes, etc.

Provable security of long-term key reuse

Goal: Generic composition theorem:

If 2 individual ciphersuites are separately secure, then they are collectively secure even if long-term keys are reused across ciphersuites.

- Impossible: TLS cross-ciphersuite attack.

Proof approach:

1. Guess the target ciphersuite
2. Use ACCE challenger for target ciphersuite
3. Simulate all other ciphersuites

Main problem: how to correctly simulate private key operations of other ciphersuites that re-use long terms key

Provable security of long-term key reuse

Revised goal: Generic composition theorem:

If 2 individual ciphersuite are separately secure under additional conditions, then they are collectively secure even if long-term keys are reused across ciphersuites.

Technical approach

1. Define multi-ciphersuite ACCE security

2. Slightly open up individual ACCE definition: "ACCE with auxiliary oracle"

3. Thm: collection of ciphersuites that are individually ACCE-secure with compatible auxiliary oracles

=> multi-ciphersuite security.

Idea: adversary shouldn't be helped if he gets signatures on "unrelated" messages

4. Prove SSH signed-DH satisfies ACCE with auxiliary oracle

ACCE with auxiliary oracle

Idea: adversary shouldn't be helped if he gets signatures on "unrelated" messages

- Auxiliary oracle $\text{aux} = \text{"get signatures"}$
- Predicate $\text{pred} = \text{"unrelated messages"}$
 - e.g. unambiguous ciphersuite description part of signed data structure

Multi-ciphersuite composition theorem

- CS_1 secure with aux_1 and $pred_1$
- CS_2 secure with aux_2 and $pred_2$

Two ciphersuites are "compatible" if

- CS_1 can be simulated using aux_2 without violating $pred_2$
- vice versa

Thm: Suite of mutually compatible individually secure ciphersuites is multi-ciphersuite secure.

Proof approach:

1. Guess the target ciphersuite
2. Use ACCE-aux challenger for target ciphersuite
3. Simulate all other ciphersuites, using aux oracle when needed for private key operations
 - Underlying challenger remains "fresh" since pred not violated

SSH multi-ciphersuite conclusions

Theory

- Definition for security of multi-ciphersuite protocols.
- Generic theorem on when it is safe to reuse long-term keys across individually secure ciphersuites.

Practice

- Confidence in signed-DH SSH ciphersuites, even if the same long-term keys are reused across ciphersuites.
 - ... and even when reused with unambiguously independent protocols.

Two approaches to multi-ciphersuite security

"Proving the TLS handshake secure (as it is)" [BFKPSZ14]

Our approach

Multi-ciphersuite

=

{KEMs}

x

{signature algs}

x

{PRFs}

x

...

Multi-ciphersuite

=

CS_1 (ACCE with aux_1 & $pred_1$)

+

CS_2 (ACCE with aux_2 & $pred_2$)

+

CS_3 (ACCE with aux_3 & $pred_3$)

+

...

Conclusions

Summary

Theory

- Provable security of single ciphersuites in isolation doesn't imply security in complex settings:
 - TLS renegotiation attack
 - multi-ciphersuite security
- Can extend ACCE security models for more complex functionality
- By opening up ACCE security models, can prove more generic composition theorems

Practice

- Confidence in TLS standardized renegotiation fixes.
- Confidence in SSH signed-DH ciphersuites in isolation or with long-term key reuse.

Questions

- **Should we be trying to cryptographically analyze these more complex properties?**
- **Is the monolithic ACCE framework the right approach?**

Is ACCE the right approach?

No

- Big definition
- Monolithic security notion
- Most proofs haven't been very modular

No

- Secure channel [CK01] a bit cleaner
- Is ACCE equivalent (in any sense) to CK01 secure channel?
 - Preliminary investigations suggest not: authenticated encryption property weaker in CK01 secure channel than ACCE

Is ACCE the right approach?

No

- **Advanced functionality (renegotiation, multi-ciphersuite) doesn't follow from standalone ACCE**
 - Need variants that "open up" ACCE definition
 - Need to re-prove security of individual ciphersuites
 - often quite easy given original ACCE proof
 - still undesirable

No

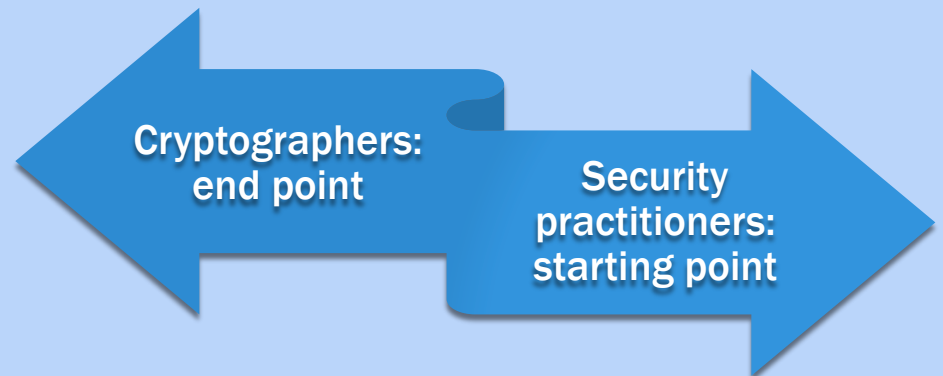
- **Many different variants of ACCE**
 - sLHAE (TLS) vs BSAE (SSH)
 - forward secrecy
 - mutual vs one-way auth.
 - public key vs. pre-shared key vs. password

Is ACCE the right approach?

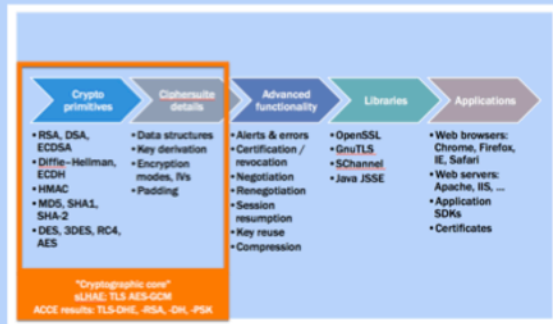
But...

- It allowed us to break through a decade of barriers in proving security of full TLS protocol.
- Adapted for proving many real-world protocols
 - TLS-DHE, TLS-RSA, TLS-DH, TLS-PSK, EMV, SSH, QUIC
 - Used by ≥ 5 independent research teams
- Unlikely to be simplifiable
 - "Surely we can simplify key exchange models"

- ACCE / secure channel is the "interface" that cryptography presents to the security world
- "Send it over a secure channel"



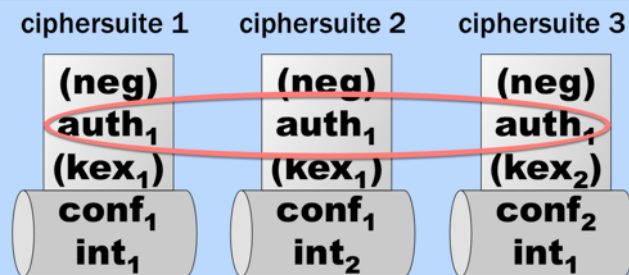
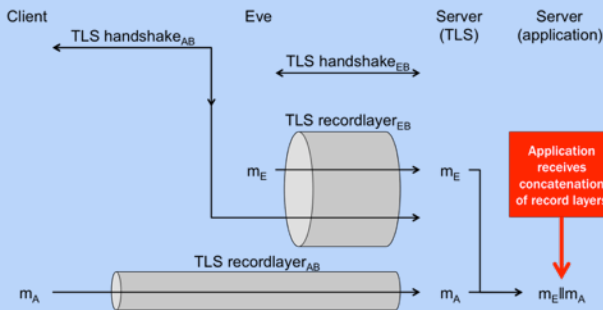
1. Gap between theory (provable security results) and practice (attacks).



Provable security of advanced properties of TLS and SSH

Douglas Stebila
 Queensland University of Technology

2. Extend provable security models and results to address TLS renegotiation and SSH multi-ciphersuite security.



Slides and papers

