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 (CCS'13), eprint 2013/813

2014/07/18

Microsoft Research

Queensland University of Technology

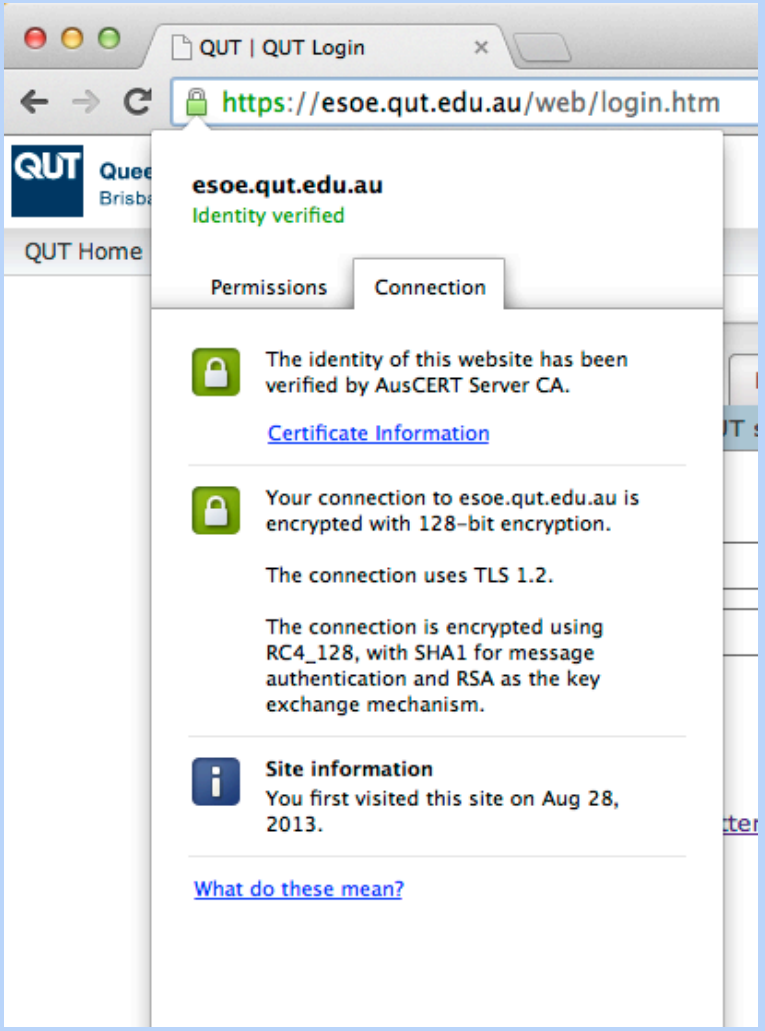
Supported by:

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

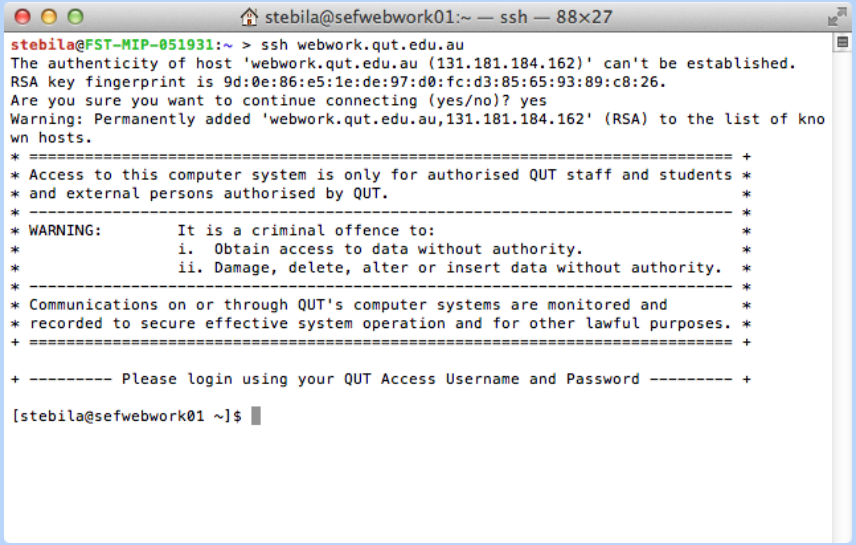
Australian Research Council Discovery Project

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

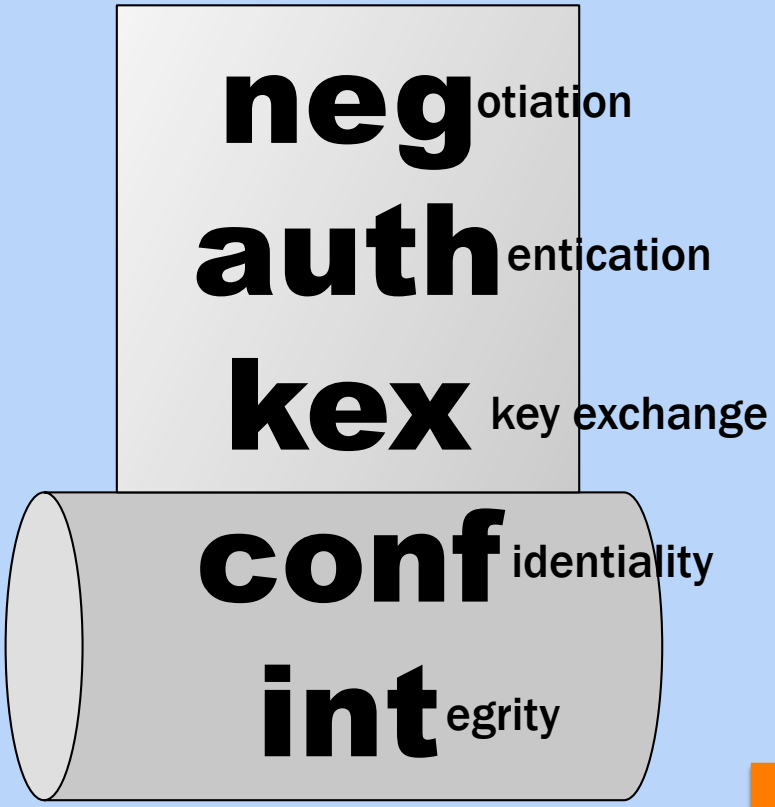


- SSH used for secure remote access (like telnet, but secure)
- Provides public key authentication of servers and clients and encrypted communication

TLS vs. SSH

TLS	SSH
provides secure transport for many applications	provides secure transport primarily for remote shell logins
entity authentication	entity authentication
confidentiality & integrity of transmissions	confidentiality & integrity of transmissions
handshake establishes secure channel	handshake establishes secure channel

Security goals of TLS and SSH



From an application perspective, TLS and SSH provide:

- negotiation of parameters
- **entity authentication**
- key exchange
- **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

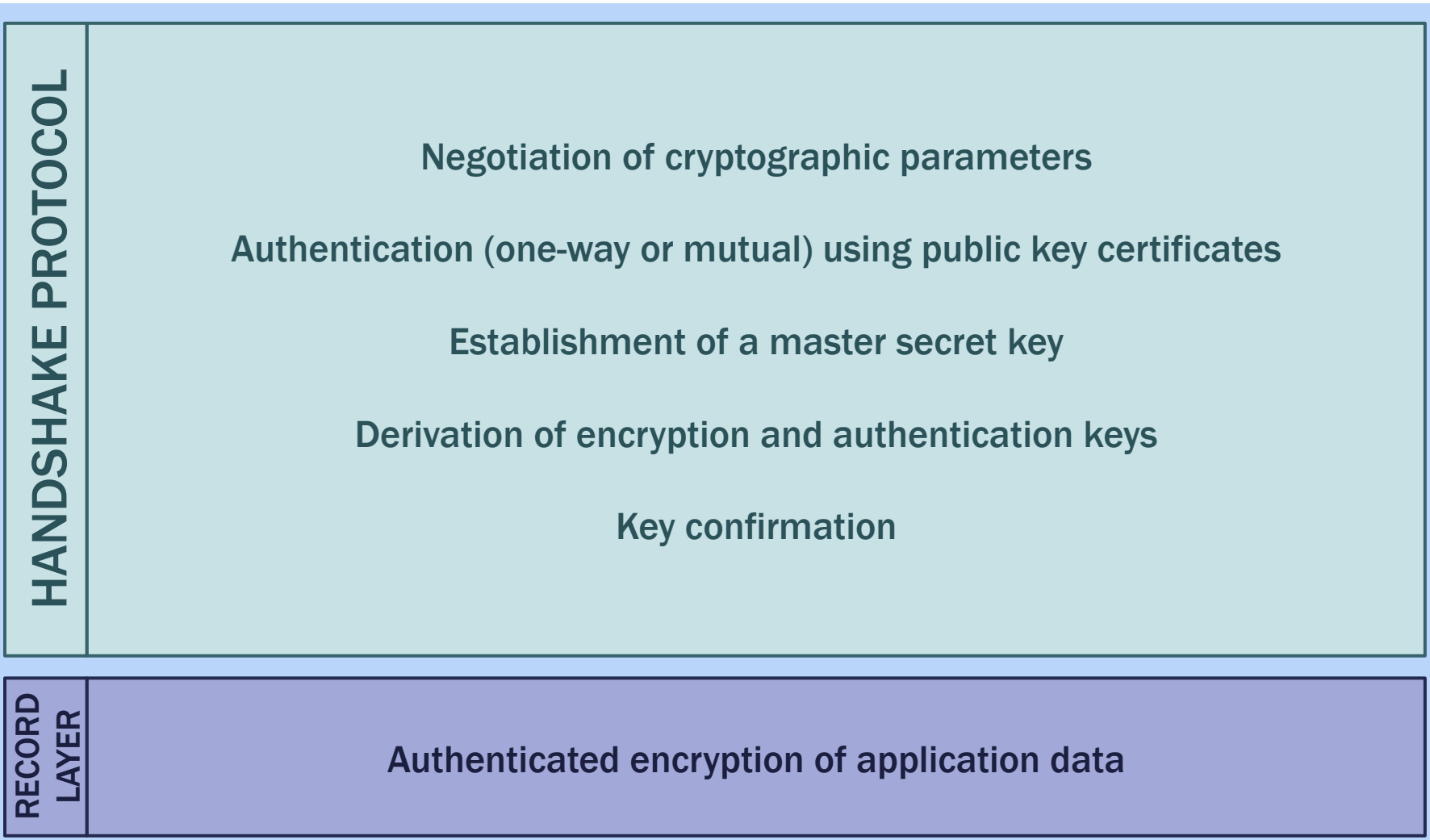
Provable security

- Define a cryptographic scheme as a set of algorithms.
- Define security as an interactive game between a challenger and an adversary
- Specify your scheme.
- Prove a theorem that any adver win the security game can be use break some hard problem ("reduction").

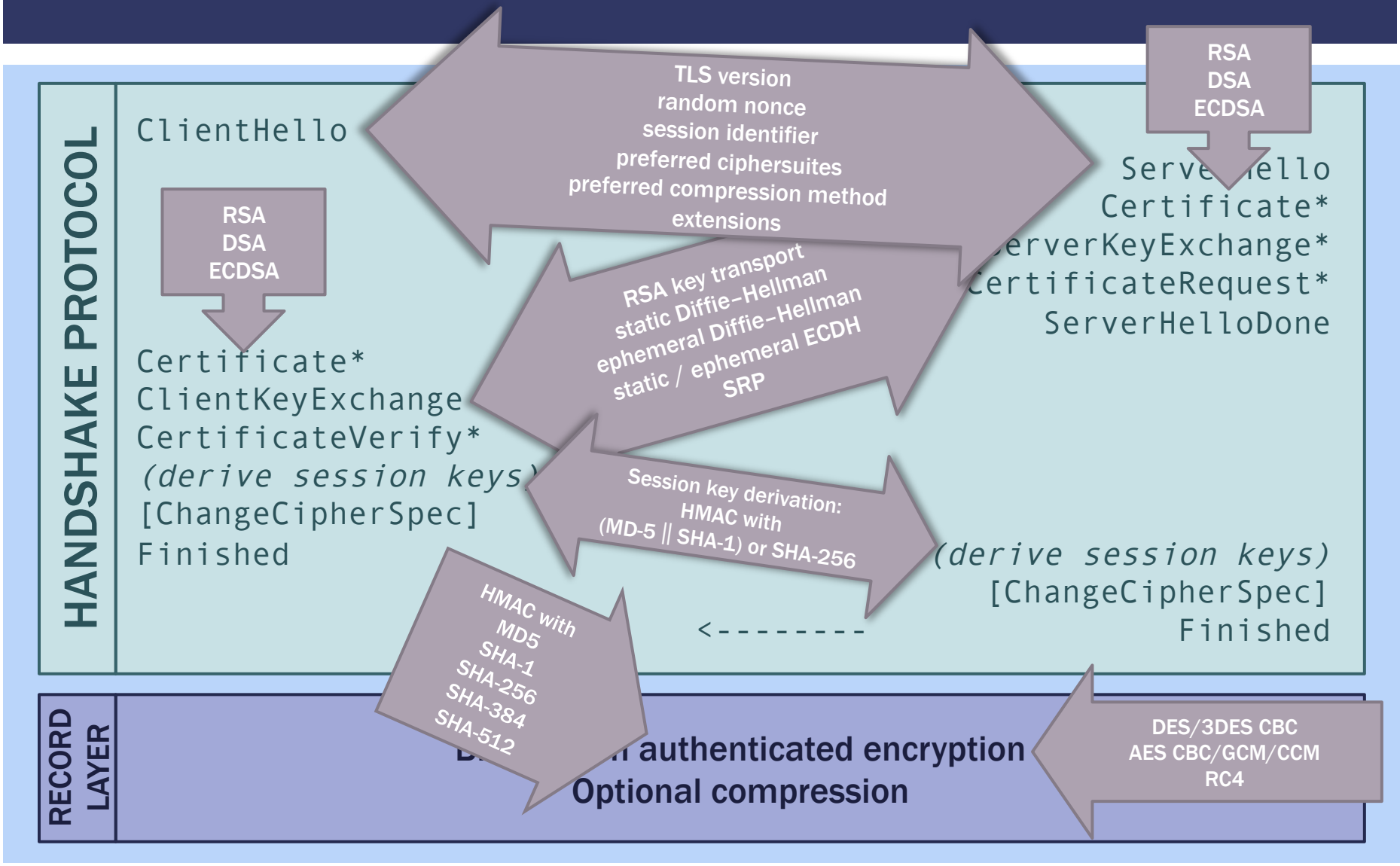
Same type of reduction as e.g. proving NP-completeness of travelling salesman problem

Security of TLS

Structure of TLS



Structure of TLS



Components of TLS

- RSA, DSA, ECDSA
- Diffie–Hellman, ECDH
- HMAC
- MD5, SHA1, SHA-2
- DES, 3DES, RC4, AES

- Data structures
- Key derivation
- Encryption modes, IVs
- Padding

- Alerts & errors
- Certification / revocation
- Negotiation
- Renegotiation
- Session resumption
- Key reuse
- Compression

- OpenSSL
- GnuTLS
- SChannel
- Java JSSE

- Web browsers: Chrome, Firefox, IE, Safari
- Web servers: Apache, IIS, …
- Application SDKs
- Certificates

Is TLS secure?

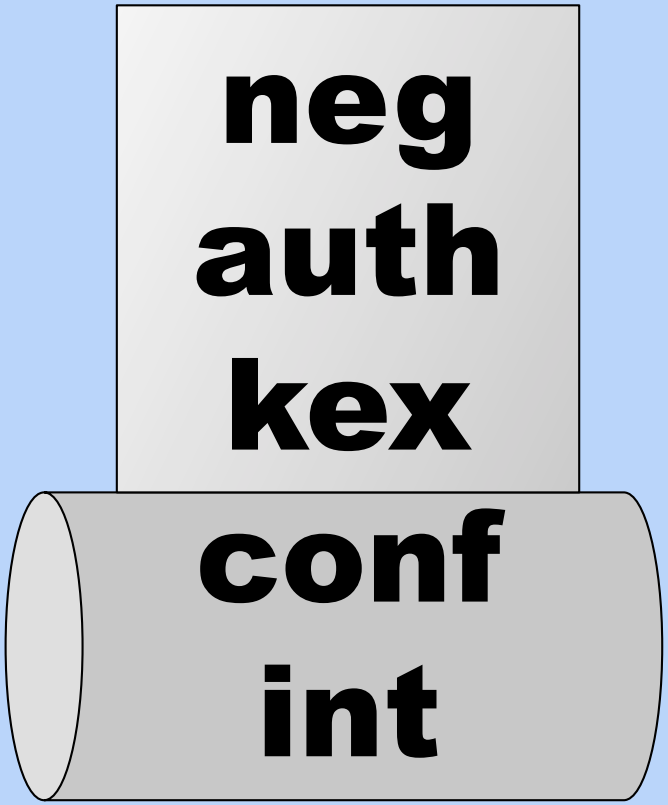
Core cryptographic components

- 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?

Security goals of TLS and SSH



From an application perspective, TLS and SSH provide:

- (negotiation of parameters)
- entity authentication
- (key exchange)
- confidentiality and integrity of messages

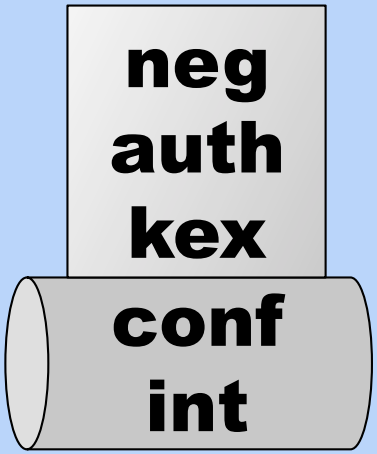
Is TLS secure?

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

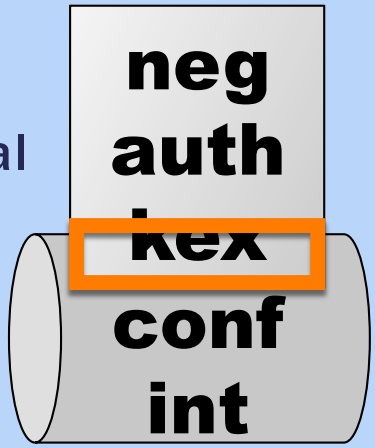
Prove the TLS record layer is a secure authenticated encryption scheme



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?

1996 SSL v3.0 standardized

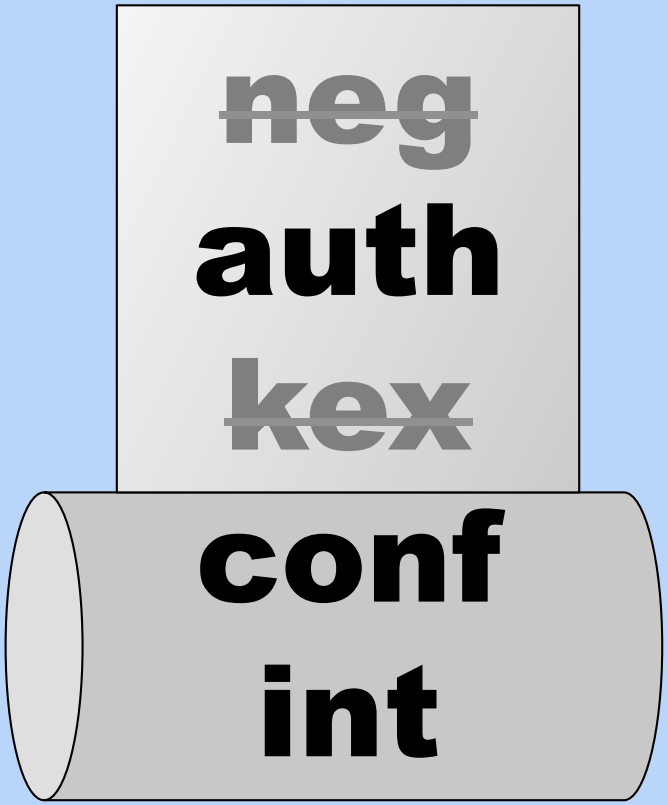
2011 Some modes of TLS record layer are secure authenticated encryption schemes [PRS11]

2012 Unaltered full signed Diffie–Hellman ciphersuite is a secure channel [JKSS12]

2013 Most unaltered full TLS ciphersuites are a secure channel [KSS13, KPW13, BFKPS13]

"unaltered"… "full"… "most ciphersuites"

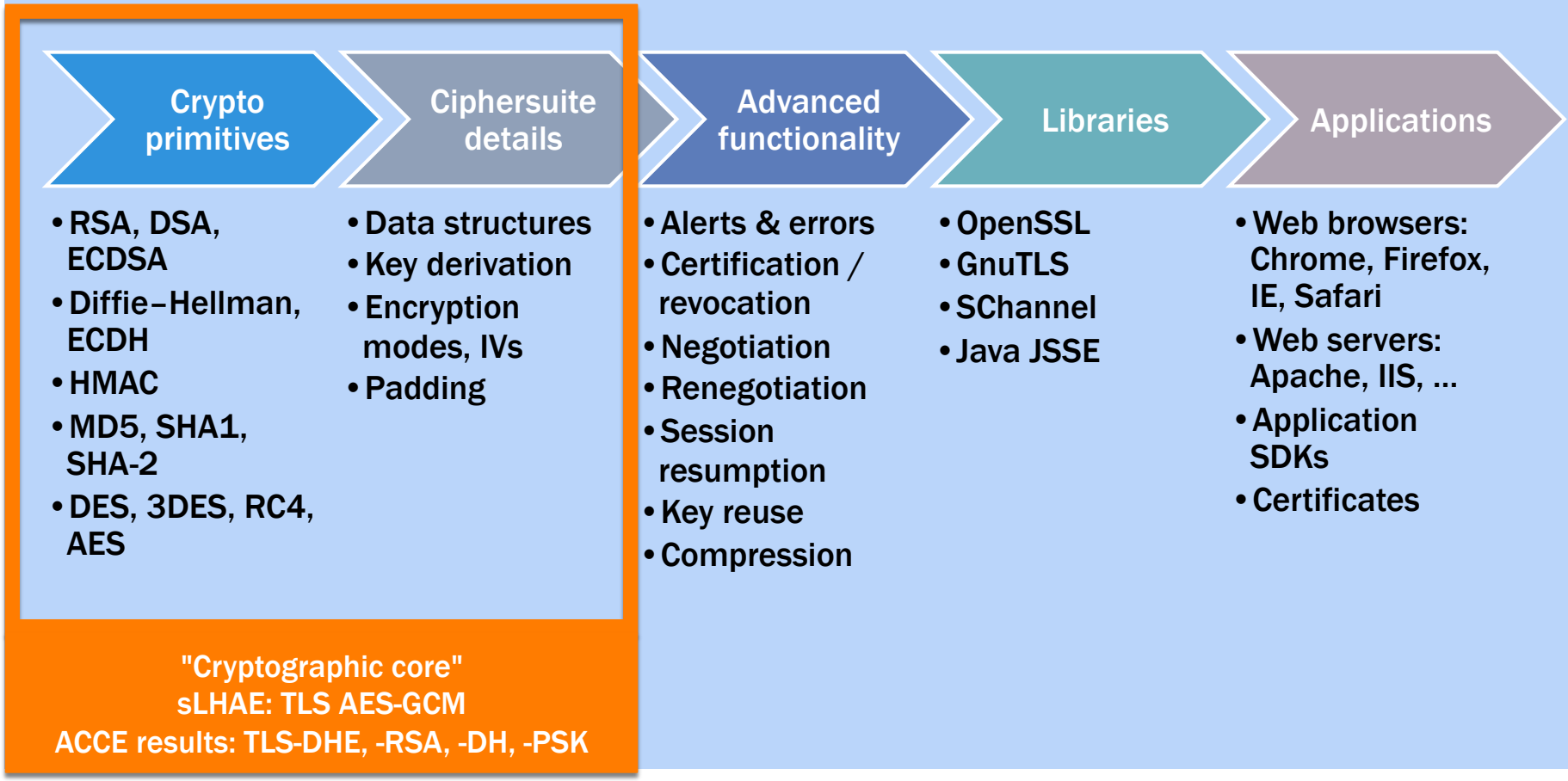
Security goals of TLS and SSH



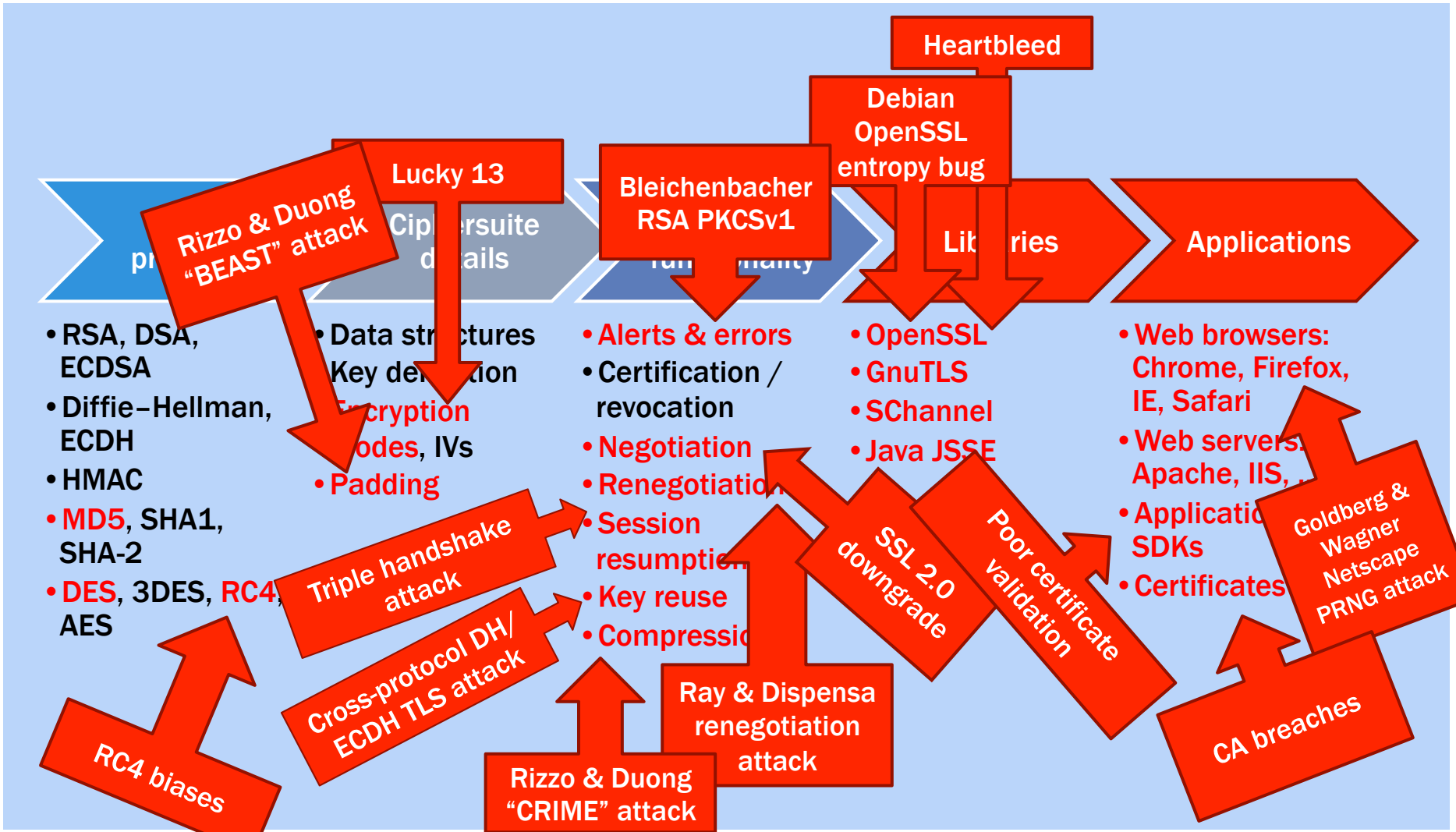
Authenticated and Confidential Channel Establishment (ACCE) security definition [JKSS12] captures:

- entity authentication
- confidentiality and integrity of messages

Results on the provable security of TLS

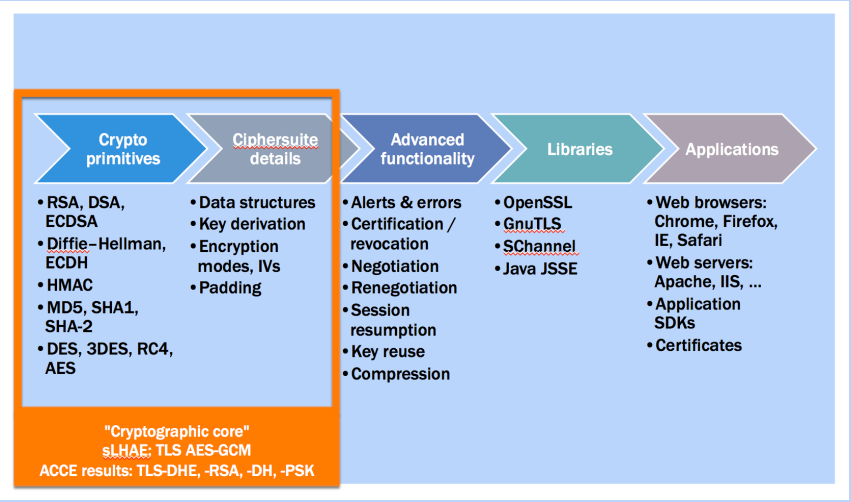


Real-world attacks on TLS

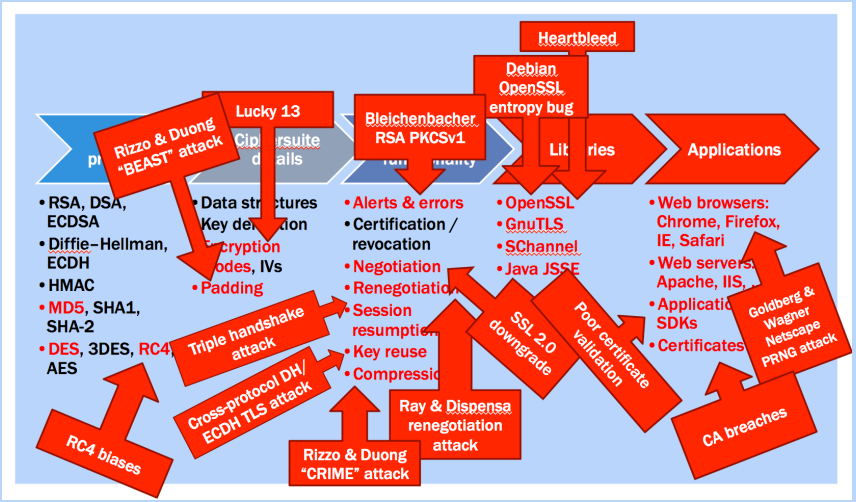


The gap between theory and practice

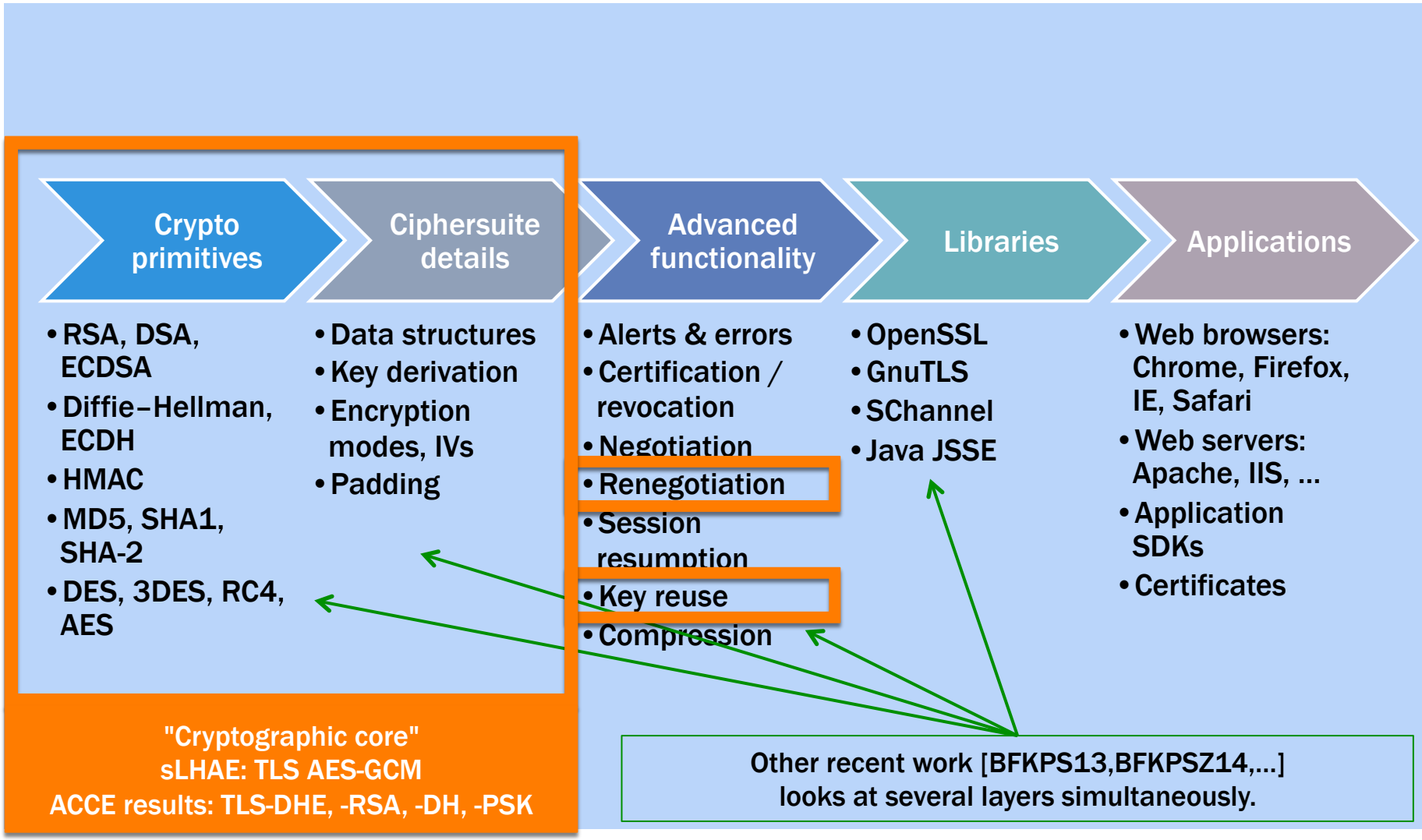
Provable security results



Real-world attacks



Components of TLS



TLS and renegotiation

joint work with
Florian Giesen & Florian Kohlar (Bochum)

ACM CCS 2013

IACR eprint 2012/630

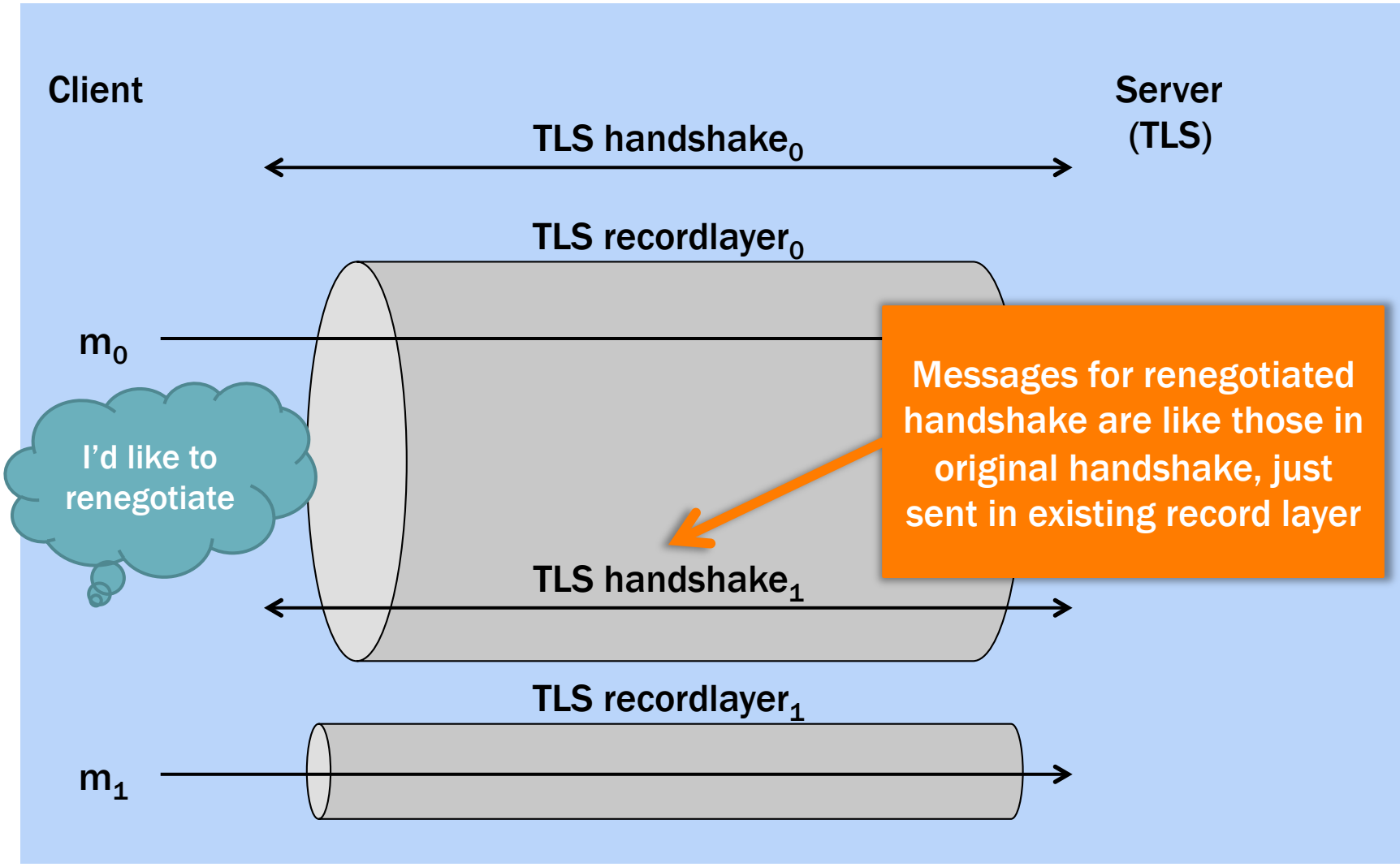
Why renegotiate?

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?

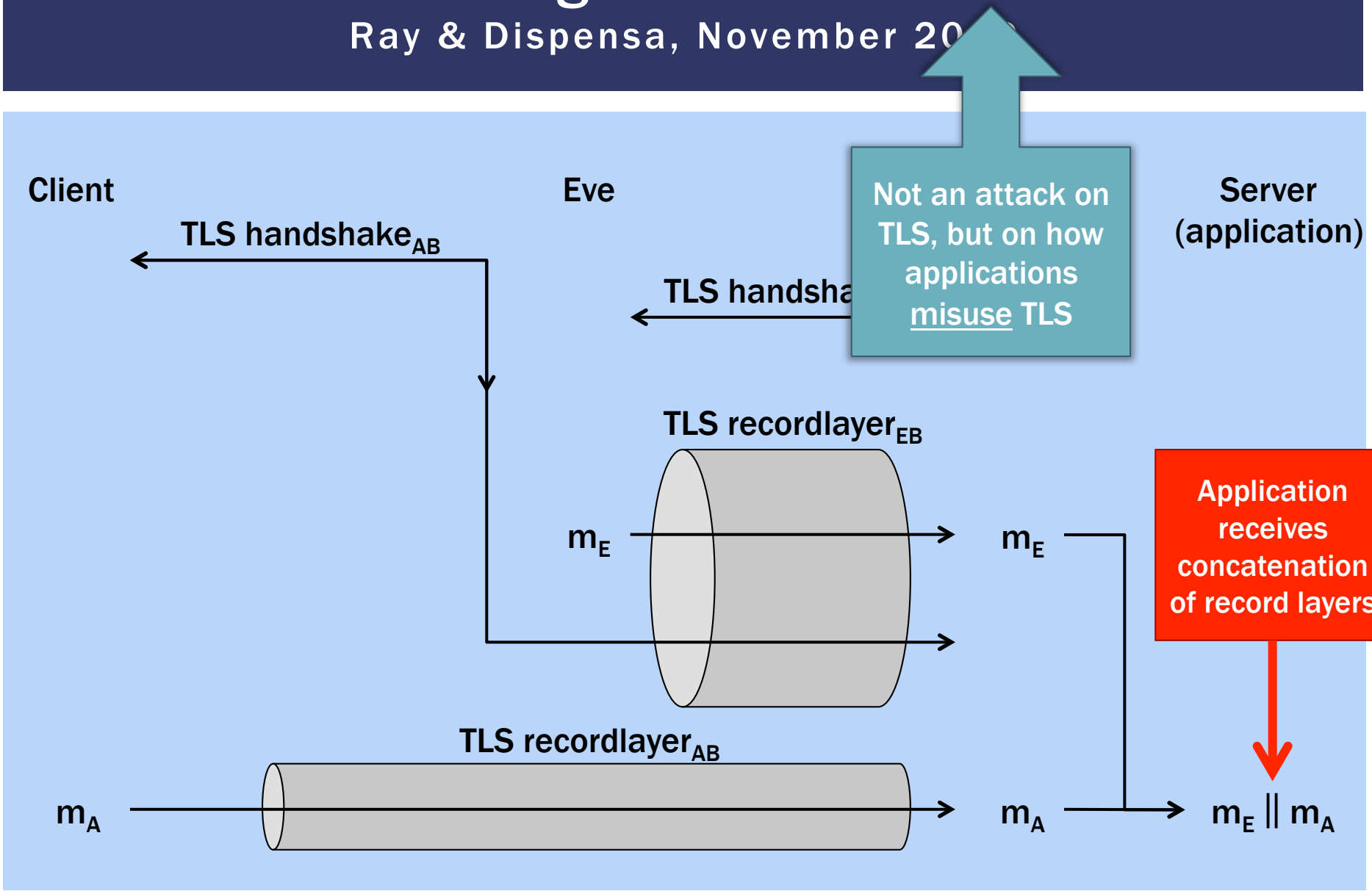
- Change cryptographic parameters
- Change authentication credentials
- Identity hiding for client
  - second handshake messages sent encrypted under first record layer
- Refresh encryption keys
  - more forward secrecy
  - record layer has maximum number of encryptions per session key

Renegotiation in TLS
(pre-November 2009)



TLS Renegotiation "Attack"

Ray & Dispensa, November 20



Example: HTTP Injection

- Attacker sends
  - m_E = "GET /orderPizza?deliverTo=123-Fake-St↩
    X-Ignore-This: "
- Client sends
  - m_A = "GET /orderPizza?deliverTo=456-Real-St↩
    Cookie: Account=1A2B"
- Server's web server receives
  - $m_E \| m_A$ = "GET /orderPizza?deliverTo=123-Fake-St↩
    X-Ignore-This: GET /orderPizza?deliverTo=456-Real-St↩
    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

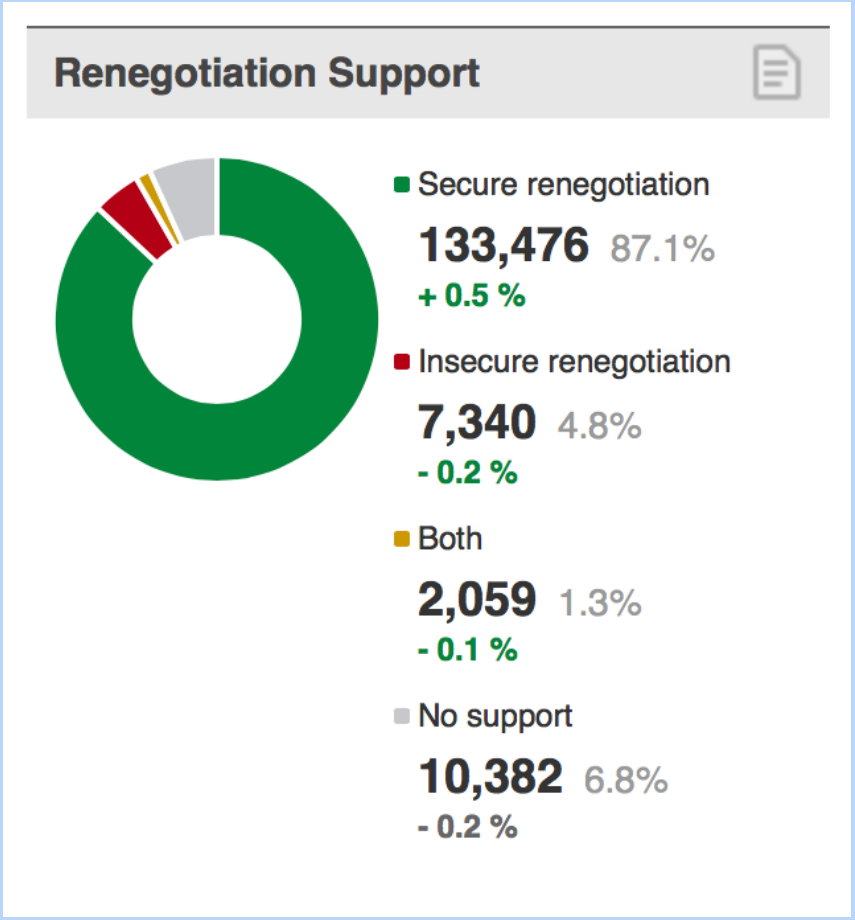
Basic idea: include fingerprint of previous handshake when renegotiating.

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

TLS Renegotiation Countermeasures

SCSV/RIE fairly quickly and widely adopted.

Currently 87% deployment
(SSL Pulse, July 2, 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.

Technical approach

1. Define a secure renegotiable ACCE

2. See that unpatched TLS not a secure renegotiable ACCE

3. Slightly open up ACCE definition: "tagged-ACCE-fin"

5. Prove TLS-DHE satisfies tagged-ACCE-fin

4. Transformation Thm: tagged-ACCE-fin + renegotiation countermeasure, => secure renegotiable ACCE.

Secure renegotiable ACCE

Definition	TLS

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

Weakly secure renegotiable ACCE

Definition	TLS

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 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)

IACR eprint
2013/813

TLS_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_WITH_DES_CBC_SHA TLS_DH_DSS_WITH_3DES_EDE_CBC_SHA TLS_DH_RSA_EXPORT_WITH_DES40_CBC_SHA TLS_DH_RSA_WITH_DES_CBC_SHA
TLS_DH_RSA_WITH_3DES_EDE_CBC_SHA TLS_DHE_DSS_EXPORT_WITH_DES40_CBC_SHA TLS_DHE_DSS_WITH_DES_CBC_SHA TLS_DHE_DSS_WITH_3DES_EDE_CBC_SHA TLS_DHE_RSA_EXPORT_WITH_DES40_CBC_SHA
TLS_DHE_RSA_WITH_DES_CBC_SHA TLS_DHE_RSA_WITH_3DES_EDE_CBC_SHA TLS_DH_anon_EXPORT_WITH_RC4_40_MD5 TLS_DH_anon_WITH_RC4_128_MD5 TLS_DH_anon_EXPORT_WITH_DES40_CBC_SHA
TLS_DH_anon_WITH_DES_CBC_SHA TLS_DH_anon_WITH_3DES_EDE_CBC_SHA TLS_KRB5_WITH_DES_CBC_SHA TLS_KRB5_WITH_3DES_EDE_CBC_SHA TLS_KRB5_WITH_RC4_128_SHA TLS_KRB5_WITH_IDEA_CBC_SHA
TLS_KRB5_WITH_DES_CBC_MD5 TLS_KRB5_WITH_3DES_EDE_CBC_MD5 TLS_KRB5_WITH_RC4_128_MD5 TLS_KRB5_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_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_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_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_WITH_AES_128_CBC_SHA256
TLS_DH_anon_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_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_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_RSA_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_WITH_AES_128_GCM_SHA256 TLS_DH_anon_WITH_AES_256_GCM_SHA384 TLS_PSK_WITH_AES_128_GCM_SHA256 TLS_PSK_WITH_AES_256_GCM_SHA384
TLS_DHE_PSK_WITH_AES_128_GCM_SHA256 TLS_DHE_PSK_WITH_AES_256_GCM_SHA384 TLS_RSA_PSK_WITH_AES_128_GCM_SHA256 TLS_RSA_PSK_WITH_AES_256_GCM_SHA384
TLS_PSK_WITH_AES_128_CBC_SHA256 TLS_PSK_WITH_AES_256_CBC_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_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_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_ECDHE_ECDSA_WITH_NULL_SHA TLS_ECDHE_ECDSA_WITH_RC4_128_SHA TLS_ECDHE_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_WITH_NULL_SHA TLS_ECDH_anon_WITH_RC4_128_SHA TLS_ECDH_anon_WITH_3DES_EDE_CBC_SHA
TLS_ECDH_anon_WITH_AES_128_CBC_SHA TLS_ECDH_anon_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_RSA_WITH_AES_128_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_WITH_ARIA_128_CBC_SHA256 TLS_DH_anon_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_WITH_ARIA_128_GCM_SHA256 TLS_DH_anon_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_WITH_CAMELLIA_128_GCM_SHA256
TLS_DH_anon_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_CCM_8 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
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

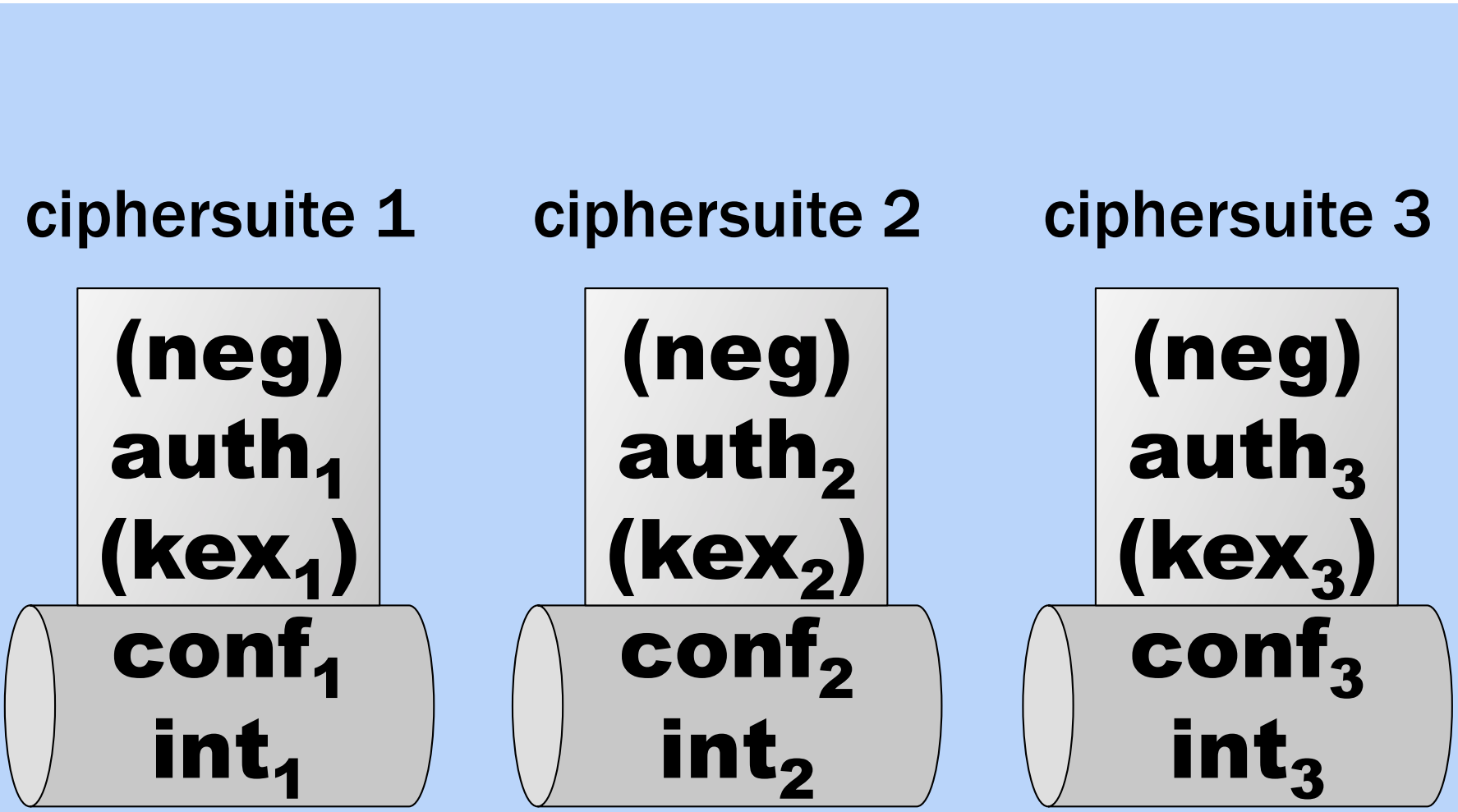
Encryption:

- 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

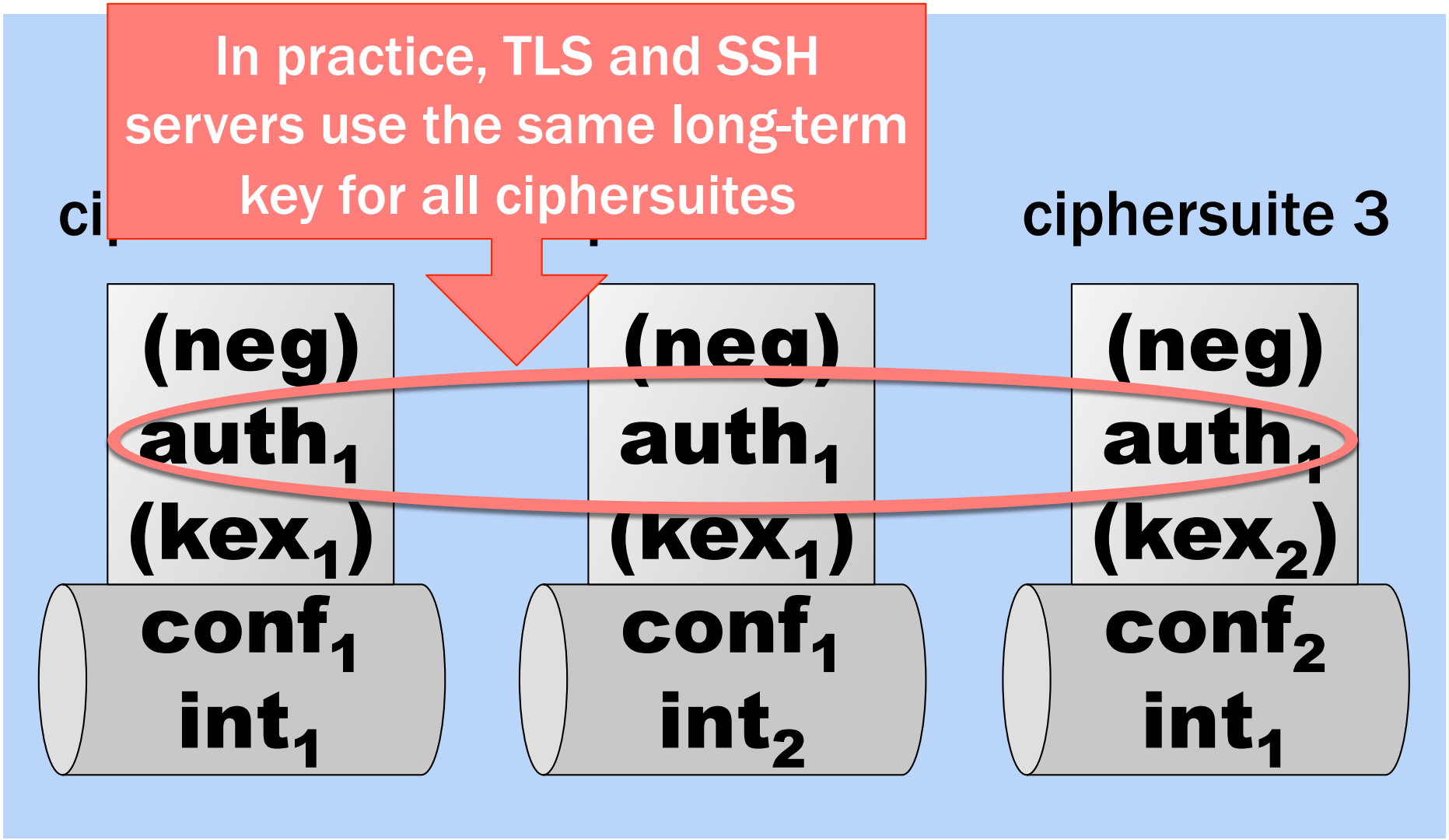
MACs:

- 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



The reality of multi-ciphersuite usage



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..2^16-1>;
  opaque dh_g<1..2^16-1>;
  opaque dh_Ys<1..2^16-1>;
} ServerDHParams;

struct {
  ECCurveType curve_type = explicit_prime(1);
  opaque      prime_p <1..2^8-1>;
  ECCurve     curve;
  ECPoint     base;
  opaque      order <1..2^8-1>;
  opaque      cofactor <1..2^8-1>;
  opaque      point <1..2^8-1>;
} 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?

Provable security of long-term key reuse

Goal: Generic composition theorem:
If an individual ciphersuite is secure, then it is secure even if long-term keys are reused across ciphersuites.

- Impossible: TLS cross-ciphersuite attack.

Proof approach:

- Guess the target ciphersuite
- Use ACCE challenger for target ciphersuite
- 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 an individual ciphersuite is secure under additional conditions, then it is 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

SSH multi-ciphersuite conclusions

Theory

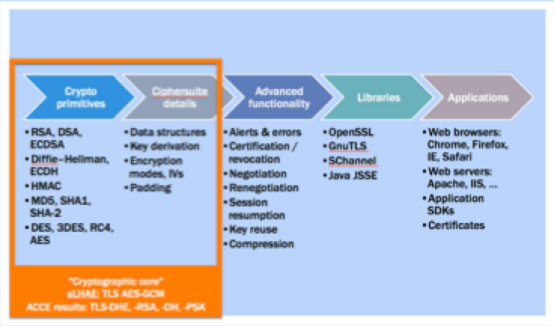
- Definition for security of multi-ciphersuite protocols.
- Generic theorem on when it is safe to reuse long-term keys across individually secure ciphersuites.

Practice

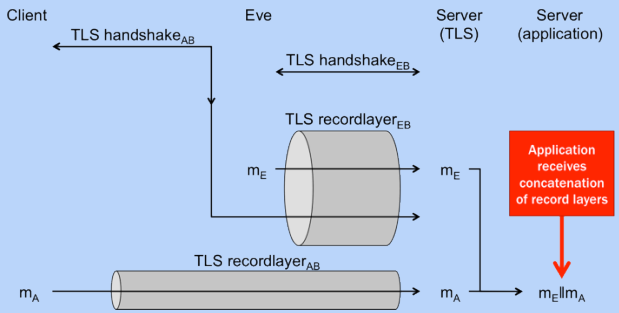
- Confidence in signed-DH SSH ciphersuites, even if the same long-term keys are reused across ciphersuites.
  - … and even when reused with unambiguously independent protocols.

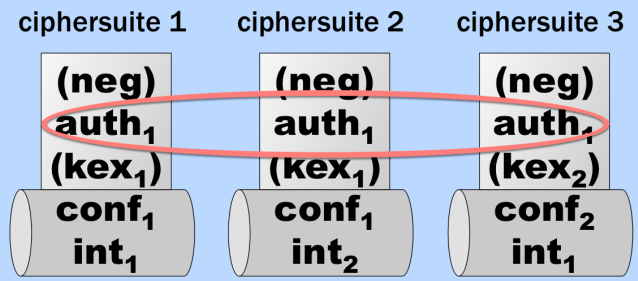
Conclusions

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



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





Provable security of advanced properties of TLS and SSH

Douglas Stebila

QUT Queensland University of Technology

Slides and papers

http://www.douglas.stebila.ca/research/presentations/