Provable security of advanced properties of TLS and SSH

Douglas Stebila

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

eprint 2012/630 (CCS'13), eprint 2013/813

2014/06/03

Secure key exchange and channels workshop
Bertinoro, Italy

Queensland University of Technology

Supported by:

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

Australian Research Council Discovery Project

Is TLS secure? – sLHAE and ACCE

2001-2008

Truncated / modified TLS handshake is secure key exchange; modified record layer is secure authenticated-encryption scheme

2011

TLS AES-GCM is a secure stateful length-hiding authenticated encryption (sLHAE) scheme

[PRS11]

2012

Signed Diffie–Hellman TLS is a secure authenticated and confidential channel establishment (ACCE) protocol

[JKSS12]

2013

Most TLS ciphersuites are ACCE-secure

[KPW13,KSS13]

Components of TLS

Crypto primitives

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

Ciphersuite details

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

Advanced functionality

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

Libraries

- OpenSSL
- GnuTLS
- SChannel
- Java JSSE

Applications

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

Components of TLS

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

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

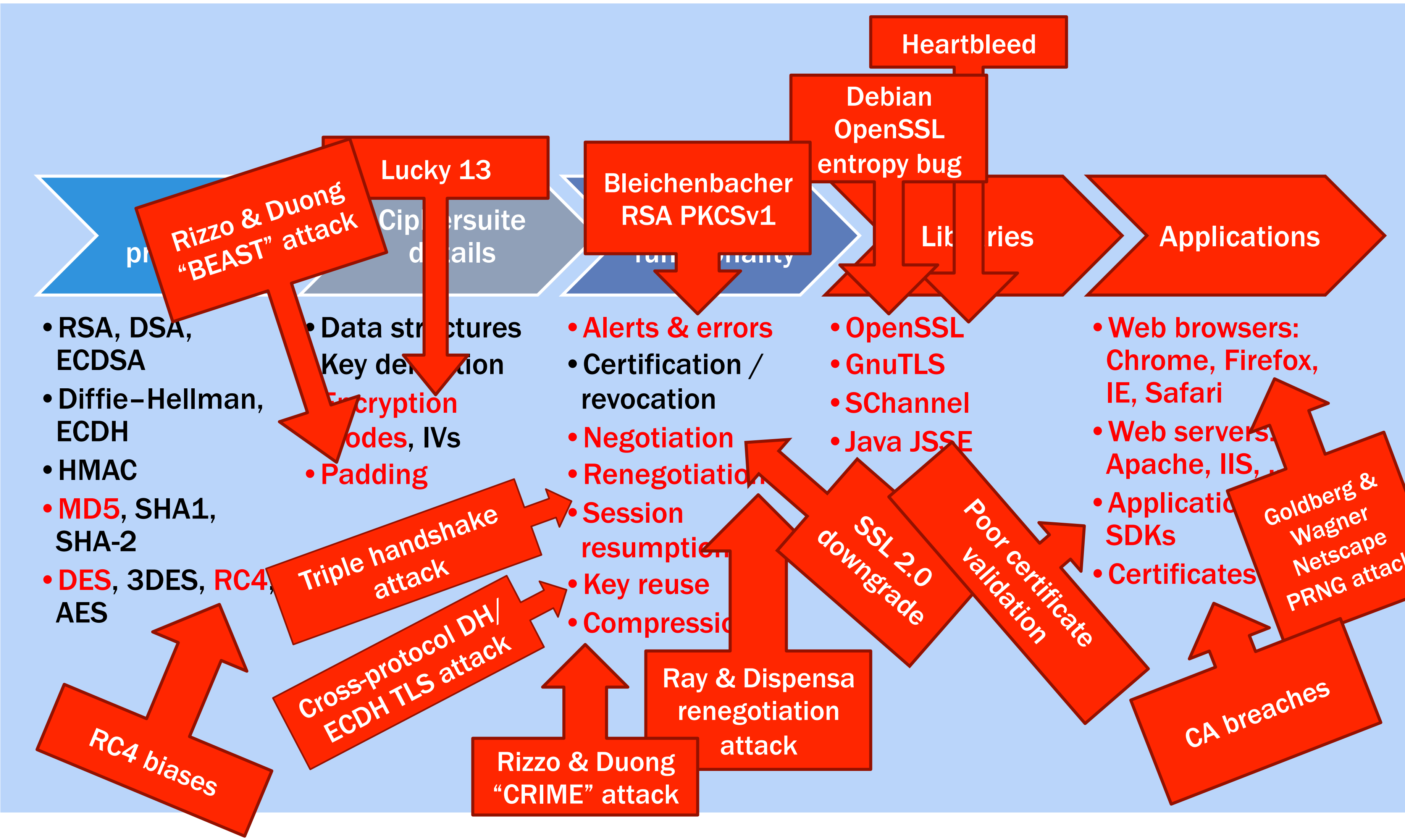
"Cryptographic core"
sLHAE: TLS AES-GCM
ACCE results: TLS-DHE, -RSA, -DH, -PSK

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

Libraries
- OpenSSL
- GnuTLS
- SChannel
- Java JSSE

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

Real-world attacks on TLS



Components of TLS

Crypto primitives

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

Ciphersuite details

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

Advanced functionality

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

Libraries

- OpenSSL
- GnuTLS
- SChannel
- Java JSSE

Applications

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

"Cryptographic core"
sLHAE: TLS AES-GCM
ACCE results: TLS-DHE, -RSA, -DH, -PSK

Other recent work [BFKPS13,BFKPSZ14,…]
looks at several layers simultaneously.

TLS and renegotiation

ACM CCS 2013
eprint 2012/630

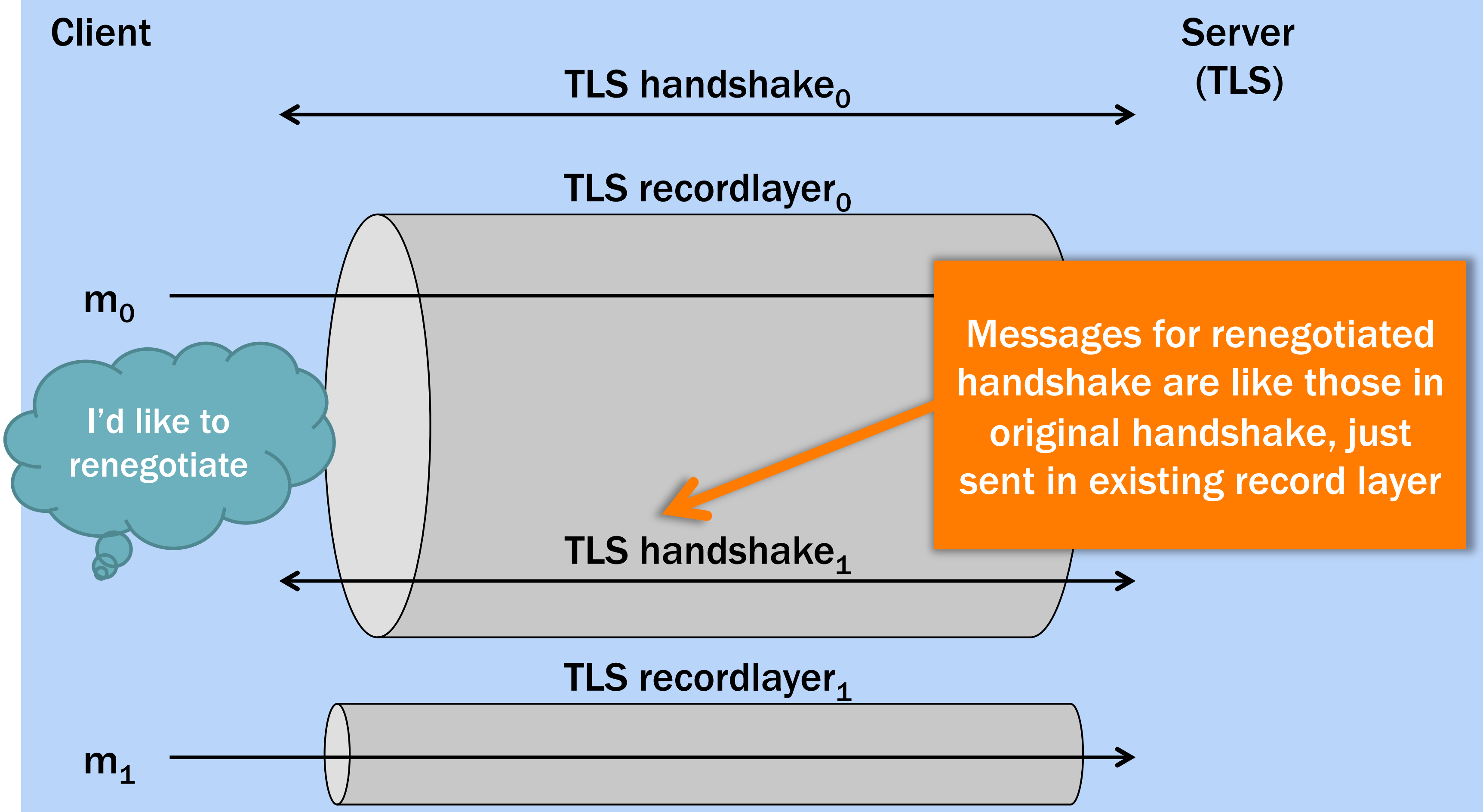
Why renegotiate?

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?

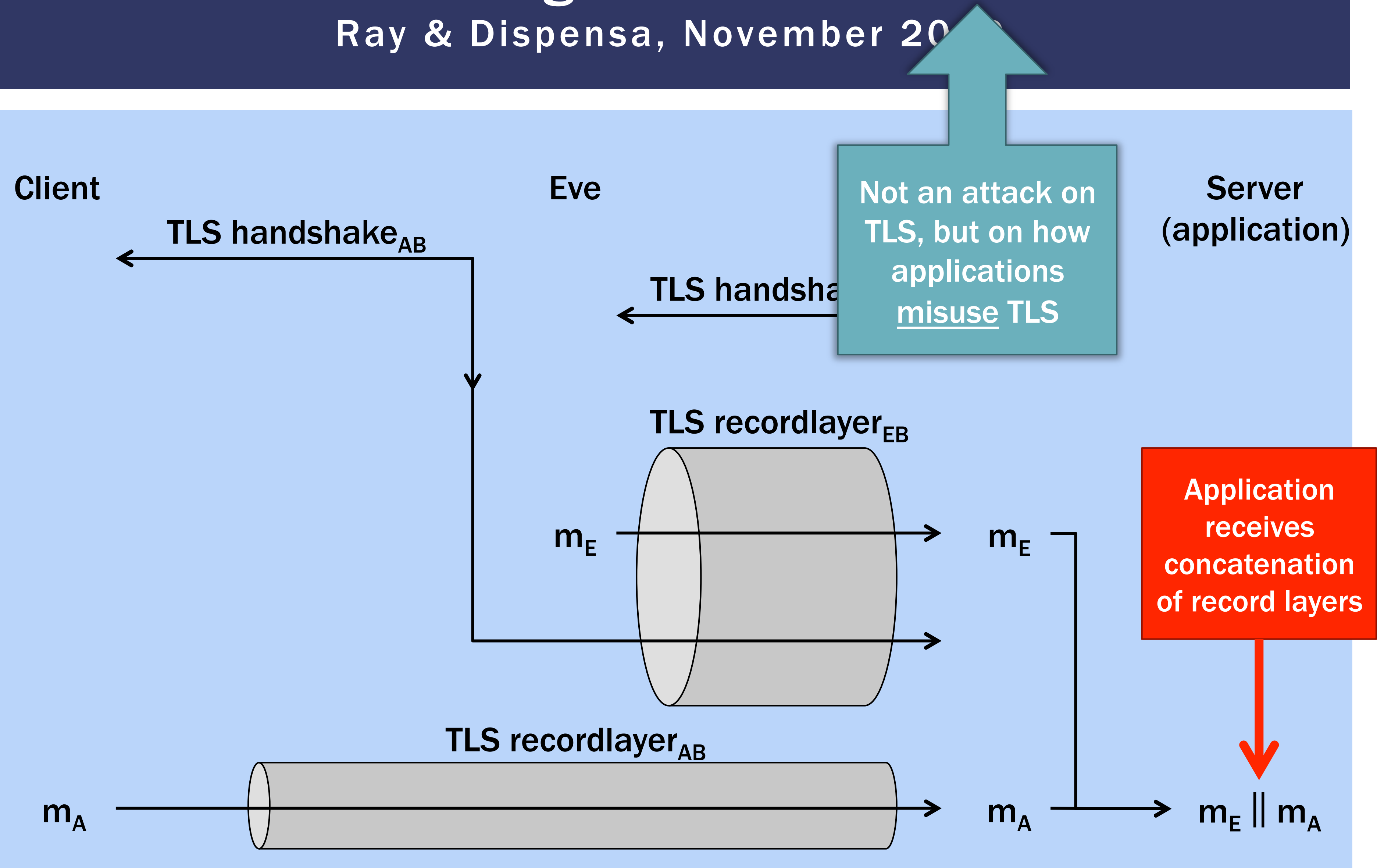
- Change cryptographic parameters
- Change authentication credentials
- Identity hiding for client
  - second handshake messages sent encrypted under first record layer
- Refresh encryption keys
  - more forward secrecy
  - record layer has maximum number of encryptions per session key

Renegotiation in TLS
(pre-November 2009)



TLS Renegotiation "Attack"

Ray & Dispensa, November 20…



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:

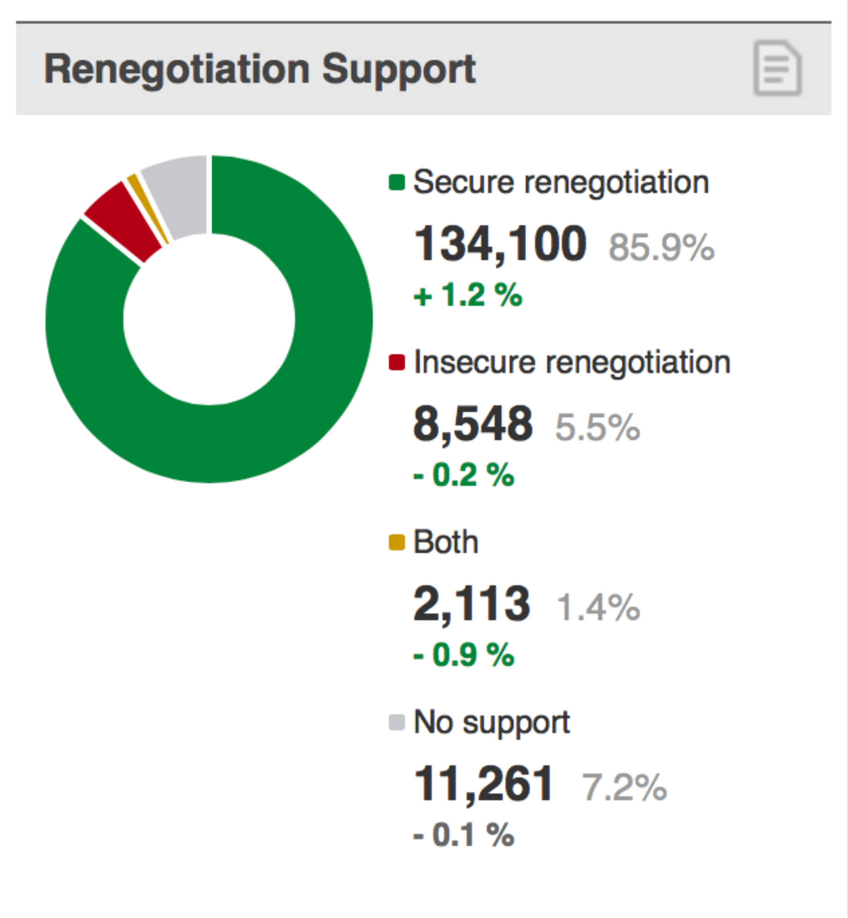
1. Signalling Ciphersuite Value
2. Renegotiation Indication Extension

Basic idea: include fingerprint of previous handshake when renegotiating.

TLS Renegotiation Countermeasures

SCSV/RIE fairly quickly and widely adopted.

Currently 86% deployment
(SSL Pulse, May 2, 2014)



Does this really fix the problem?

Does this really fix the problem?

ACCE security isn't enough: these ciphersuites have been proven ACCE security yet are vulnerable to renegotiation attack.

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"

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

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

Multi-phase ACCE

Definition

Each instance $\prod_i^s$ can have multiple **phases** each of which consists of a handshake and record layer.

- Separately keep track of handshake and record layer transcript for each phase.

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.)

Proving security of TLS renegotiation countermeasure

TLS Renegotiation Indication Extension

- Include Finished messages from previous handshake in renegotiated handshake
  - Finished message includes a hash of the handshake transcript
  - Authenticates previous handshake

- A "white box" modification of TLS
  - reveals an intermediate (encrypted) value
  - modifies messages
- New ACCE-based definition: tagged-ACCE-fin
  - "tagged": can include arbitrary tag data in handshake
  - "fin": Finished messages

Compiler: TLS countermeasure achieves weakly secure renegotiation

1. If a generic TLS ciphersuite P is tagged-ACCE-fin, then P +RIE is multi-phase secure.

2. If P+RIE is multi-phase secure and the PRF is secure, then P+RIE is weakly secure renegotiable ACCE.

3. TLS-DHE is tagged-ACCE-fin.

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

- 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

eprint 2013/813

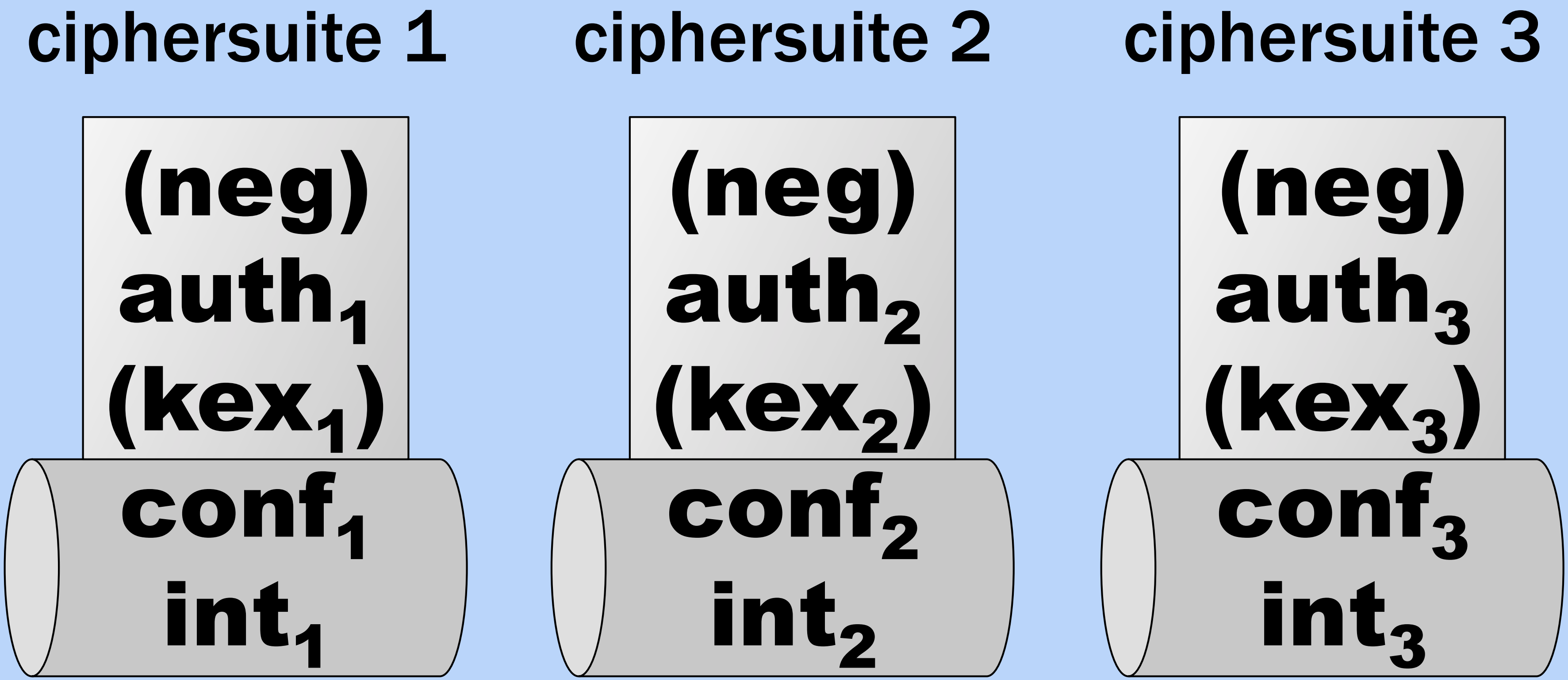
List of all 314 TLS ciphersuites

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 SSH ciphersuites

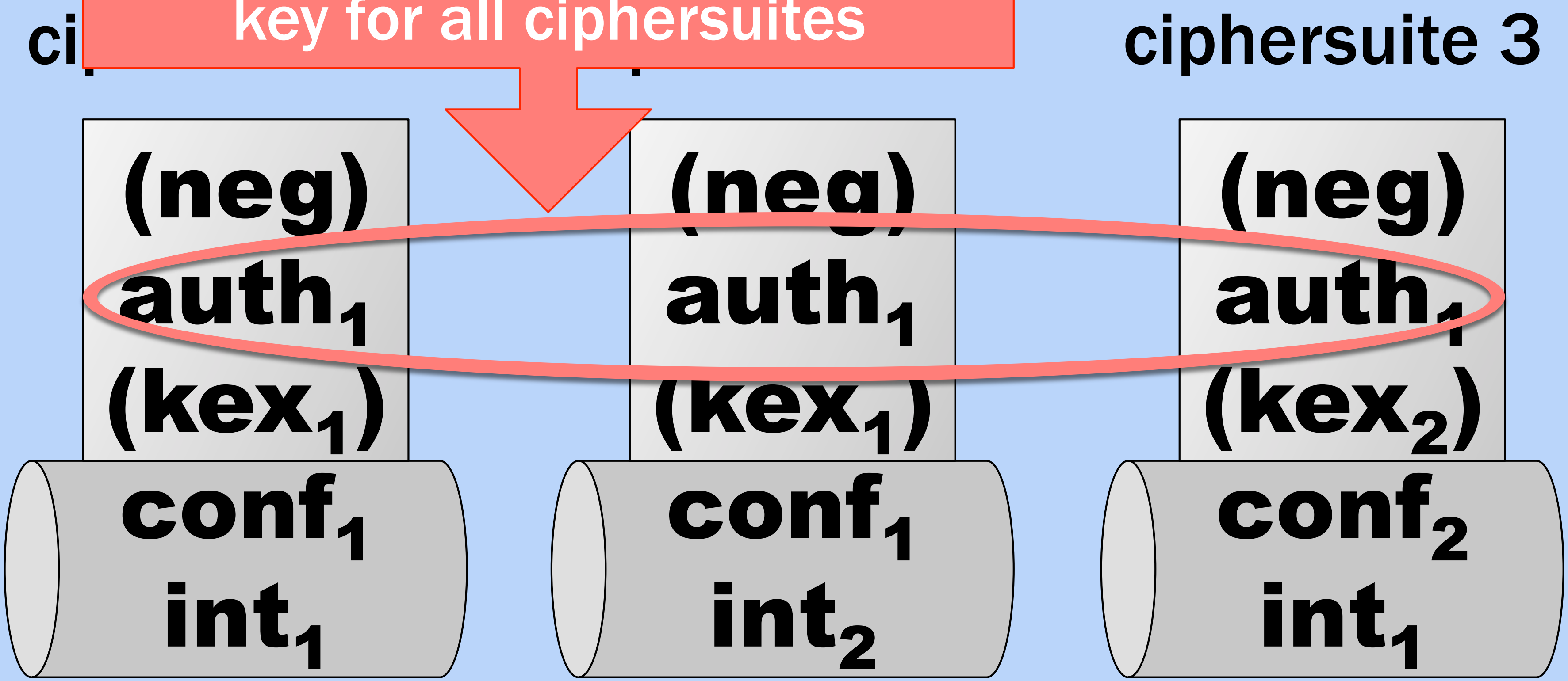
- **Authentication:**
  - RSA signatures
  - DSA-SHA1
  - ECDSA-SHA2
  - X509-RSA signatures
  - X509-DSA-SHA1
  - X509-ECDSA-SHA2
- **Key exchange:**
  - DH explicit group SHA1
  - DH explicit group SHA2
  - DH group 1 SHA1
  - DH group 14 SHA1
  - ECDH-nistp256-SHA2
  - ECDH-nistp384-SHA2
  - ECDH-nistp521-SHA2
  - 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

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..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]

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 <u>under additional conditions</u>, 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.

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 aux = "get signatures"
- Predicate pred = "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:

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

Lessons learned: multi-ciphersuite

Theory	Practice

Theory

- Definition for security of multi-ciphersuite protocols.
- Generic theorem on when it is safe to reuse long-term keys across individually secure ciphersuites.
  - Main idea: adding an auxiliary "signing oracle" to individual security to enable reduction, parameterize freshness condition.
  - Lots of other applications of this main idea…

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)"	Our approach
Multi-ciphersuite	Multi-ciphersuite
=	=
{KEMs}	CS_1 (ACCE with aux_1 & $pred_1$)
x	+
{signature algs}	CS_2 (ACCE with aux_2 & $pred_2$)
x	+
{PRFs}	CS_3 (ACCE with aux_3 & $pred_3$)
x	+
…	…

Summary

Theory	Practice
▪ 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	▪ 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	No

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

- Secure channel [CK01] a bit cleaner
  - Is ACCE equivalent (in any sense) to secure channel?

Is ACCE the right approach?

No	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

- 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"