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)
Ben, Florian K, Florian B, Jörg, Douglas
The “s” in “https”

The **most important** cryptographic protocol on the Internet — used to secure billions of connections every day.

**TLS (Transport Layer Security) protocol**

a.k.a. **SSL (Secure Sockets Layer)**
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

<table>
<thead>
<tr>
<th>TLS</th>
<th>SSH</th>
</tr>
</thead>
<tbody>
<tr>
<td>- provides secure transport for many applications</td>
<td>- provides secure transport primarily for remote shell logins</td>
</tr>
<tr>
<td>- entity authentication</td>
<td>- entity authentication</td>
</tr>
<tr>
<td>- confidentiality &amp; integrity of transmissions</td>
<td>- confidentiality &amp; integrity of transmissions</td>
</tr>
<tr>
<td>- handshake establishes secure channel</td>
<td>- handshake establishes secure channel</td>
</tr>
</tbody>
</table>
From an application perspective, TLS and SSH provide:

- entity authentication
- confidentiality and integrity of messages

a lot more
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

**HANDSHAKE PROTOCOL**

ClientHello

- Certificate*
- CertificateVerify* (derive session keys)
- ClientKeyExchange
- Finished

ServerHello

- ServerHelloDone
- ServerKeyExchange*
- CertificateRequest*
- Certificate

**RECORD LAYER**

- HMAC with MD5, SHA-1, SHA-256, SHA-384, SHA-512
- DES/3DES CBC
- AES CBC/GCM/CCM
- RC4

**Session key derivation:**

HMAC with (MD-5 || SHA-1) or SHA-256

**TLS version**

random nonce

session identifier

preferred ciphersuites

preferred compression method

extensions

**RSA key transport**

static Diffie–Hellman

ephemeral Diffie–Hellman

SRP

**Session key derivation:**

HMAC with (MD-5 || SHA-1) or SHA-256

**RSA**

DSA

ECDSA

**RSA**

DSA

ECDSA
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
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?
<table>
<thead>
<tr>
<th>Idea</th>
<th>Problem</th>
</tr>
</thead>
</table>
| 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 | TLS handshake sends messages encrypted under the session key  
  - => overlap between handshake and record layer  
  - Adversary can distinguish real session key from random |

- neg auth kex conf int
- neg auth kex conf int
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?

SSL v3.0 standardized in 1996.

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

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

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

“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

**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

Provably secure "cryptographic core"
lSHAE: TLS AES-GCM
ACCE results: TLS-DHE, -RSA, -DH, -PSK
Real-world attacks on 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

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

SSL 2.0 downgrade
- Poor certificate validation

Poor certificate
- SSL 2.0 downgrade
- Certificate validation

Libraries
- OpenSSL
- GnuTLS
- SChannel
- Java JSSE

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

Related attacks
- Rizzo & Duong "BEAST" attack
- Lucky 13
- Bleichenbacher RSA PKCSv1
- Cross-protocol DH/ECDH TLS attack
- Triple handshake attack
- SSL 2.0 downgrade
- Poor certificate validation
- Rizzo & Duong "CRIME" attack
- Ray & Dispensa renegotiation attack
- RC4 biases
- Debian OpenSSL entropy bug
- goto fail;
Extending provable security results

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, BDFPS14, BFKPSZ14,...] looks at several layers simultaneously.
TLS and renegotiation

joint work with
Florian Giesen & Florian Kohlar (Bochum)
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

I’d like to renegotiate

Server
(TLS)

Messages for renegotiated handshake are like those in original handshake, just sent in existing record layer
TLS Renegotiation “Attack”
Ray & Dispensa, November 2009

Client

TLS handshake_{AB}

Eve

Not an attack on TLS, but on how applications misuse TLS

Server (application)

Application receives concatenation of record layers

Server

m_E \parallel m_A

Client

m_A

m_E

m_E

TLS recordlayer_{AB}

TLS recordlayer_{EB}
Example: HTTP Injection

- **Attacker sends**
  - \( m_E = \text{“GET /orderPizza?deliverTo=123-Fake-St\(\uparrow\)}\)
  - X-Ignore-This: \(\uparrow\)

- **Client sends**
  - \( m_A = \text{“GET /orderPizza?deliverTo=456-Real-St\(\uparrow\)}\)
  - Cookie: Account=1A2B

- **Server’s web server receives**
  - \( m_E \parallel m_A = \text{“GET /orderPizza?deliverTo=123-Fake-St\(\uparrow\)}\)
  - X-Ignore-This: GET /orderPizza?deliverTo=456-Real-St\(\uparrow\)
  - 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.
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.
SCSV/RIE fairly quickly and widely adopted.

Currently 88% deployment (SSL Pulse, Sept 3, 2014)
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.
<table>
<thead>
<tr>
<th>Definition</th>
<th>TLS</th>
</tr>
</thead>
</table>
| 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 fixes is **not** a secure renegotiable ACCE.  
- TLS with RFC 5746 fixes is **not** a secure renegotiable ACCE. |
<table>
<thead>
<tr>
<th>Definition</th>
<th>TLS</th>
</tr>
</thead>
</table>
| 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.) |
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)
<table>
<thead>
<tr>
<th>Ciphersuites</th>
</tr>
</thead>
<tbody>
<tr>
<td>TLS NULL WITH NULL SHA</td>
</tr>
<tr>
<td>TLS RSA WITH NULL SHA</td>
</tr>
<tr>
<td>TLS RSA WITH RC4 40 MD5 TLS RSA WITH RC4 128 MD5 TLS RSA WITH RC4 128 SHA</td>
</tr>
</tbody>
</table>

**List of all 314 TLS 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
- arcfour128
- idea-cbc
- cast128-cbc
- des-cbc
- arcfour128
- rsa1024-sha1
- rsa2048-sha2
- ecmqv-*-sha2

**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
- (neg)
- auth$_1$
- (kex$_1$)
- conf$_1$
- int$_1$

(ciphersuite 2)
- (neg)
- auth$_2$
- (kex$_2$)
- conf$_2$
- int$_2$

(ciphersuite 3)
- (neg)
- auth$_3$
- (kex$_3$)
- conf$_3$
- int$_3$
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.
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)

1. No "type" information.

2. Some valid ServerECDHParams binary strings are also valid WEAK ServerDHParams binary strings.
=> TLS not secure with long-term key reuse.

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

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.
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
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.
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

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

- $\text{CS}_1$ secure with $\text{aux}_1$ and $\text{pred}_1$
- $\text{CS}_2$ secure with $\text{aux}_2$ and $\text{pred}_2$

Two ciphersuites are "compatible" if
- $\text{CS}_1$ can be simulated using $\text{aux}_2$ without violating $\text{pred}_2$
- 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 $\text{pred}$ 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**

<table>
<thead>
<tr>
<th>Approach</th>
<th>Multi-ciphersuite</th>
<th>Our approach</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Proving the TLS handshake secure (as it is)&quot; [BFKPSZ14]</td>
<td>( \text{Multi-ciphersuite} = {\text{KEMs}} \times {\text{signature algs}} \times {\text{PRFs}} \times \ldots )</td>
<td>( \text{Multi-ciphersuite} = \text{CS}_1 \text{ (ACCE with aux}_1 \text{ &amp; pred}_1 \text{)} + \text{CS}_2 \text{ (ACCE with aux}_2 \text{ &amp; pred}_2 \text{)} + \text{CS}_3 \text{ (ACCE with aux}_3 \text{ &amp; pred}_3 \text{)} + \ldots )</td>
</tr>
</tbody>
</table>

**Our approach**

\[ \text{Multi-ciphersuite} = \text{CS}_1 \text{ (ACCE with aux}_1 \text{ & pred}_1 \text{)} + \text{CS}_2 \text{ (ACCE with aux}_2 \text{ & pred}_2 \text{)} + \text{CS}_3 \text{ (ACCE with aux}_3 \text{ & pred}_3 \text{)} + \ldots \]
Conclusions
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.
Should we be trying to cryptographically analyze these more complex properties?

Is the monolithic ACCE framework the right approach?
<table>
<thead>
<tr>
<th>Is ACCE the right approach?</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="No" alt="No" /></td>
</tr>
</tbody>
</table>

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

- **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?

- 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
<table>
<thead>
<tr>
<th>But...</th>
<th>ACCE / secure channel is the &quot;interface&quot; that cryptography presents to the security world</th>
</tr>
</thead>
<tbody>
<tr>
<td>It allowed us to break through a decade of barriers in proving security of full TLS protocol.</td>
<td>&quot;Send it over a secure channel&quot;</td>
</tr>
<tr>
<td>Adapted for proving many real-world protocols</td>
<td></td>
</tr>
<tr>
<td>- TLS-DHE, TLS-RSA, TLS-DH, TLS-PSK, EMV, SSH, QUIC</td>
<td></td>
</tr>
<tr>
<td>Used by ≥ 5 independent research teams</td>
<td></td>
</tr>
<tr>
<td>Unlikely to be simplifiable</td>
<td></td>
</tr>
<tr>
<td>&quot;Surely we can simplify key exchange models&quot;</td>
<td></td>
</tr>
</tbody>
</table>
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.