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
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

<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:

- Negotiation of parameters
- Entity authentication
- Key exchange
- Confidentiality and integrity of messages

Security goals of TLS and SSH
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 adversary that can win the security game can be used to 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

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*
ClientKeyExchange
CertificateVerify*
(derive session keys)
[ChangeCipherSpec]
Finished

ServerHello
Certificate*
ServerKeyExchange*
CertificateRequest*
ServerHelloDone

TLS version
random nonce
session identifier
preferred ciphersuites
preferred compression method
extensions

TLS version
random nonce
session identifier
preferred ciphersuites
preferred compression method
extensions

RSA key transport
static Diffie–Hellman
ephemeral Diffie–Hellman
static / ephemeral ECDH
SRP

Session key derivation:
HMAC with
(MD-5 || SHA-1) or SHA-256

HMAC with
MD5
SHA-1
SHA-256
SHA-384
SHA-512

RECORD LAYER

Optional compression

DES/3DES CBC
AES CBC/GCM/CCM
RC4

in authenticated encryption
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?

<table>
<thead>
<tr>
<th>Core cryptographic components</th>
<th>Additional protocol functionality</th>
</tr>
</thead>
<tbody>
<tr>
<td>Handshake protocol</td>
<td>Alerts &amp; errors?</td>
</tr>
<tr>
<td>- secure authenticated key exchange protocol?</td>
<td>- Certification?</td>
</tr>
<tr>
<td>Record layer</td>
<td>- Renegotiation?</td>
</tr>
<tr>
<td>- secure authenticated encryption channel?</td>
<td>- Session resumption?</td>
</tr>
<tr>
<td></td>
<td>- Long-term key re-use?</td>
</tr>
</tbody>
</table>
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
SSL v3.0 standardized

1996

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

2001

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

2002

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

2008

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

Is TLS secure?
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”
Authenticated and Confidential Channel Establishment (ACCE) security definition [JKSS12] captures:  
- entity authentication  
- confidentiality and integrity of messages
Results on the provable security 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
Real-world attacks on TLS

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

**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**
- **Goldberg & Wagner Netscape PRNG attack**
- **CA breaches**

**Heartbleed**
- **Debian OpenSSL entropy bug**

**Ray & Dispensa renegotiation attack**
- **SSL 2.0 downgrade**
- **Poor certificate validation**

**Lucky 13**
- **Triple handshake attack**

**Bleichenbacher RSA PKCSv1**
- **RC4 biases**

**Rizzo & Duong “BEAST” attack**
- **Cross-protocol DH/ECDH TLS attack**

**Rizzo & Duong “CRIME” attack**
The gap between theory and practice

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

Real-world attacks

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

- Debian OpenSSL entropy bug
- "BEAST" attack
- Lucky 13
- Bleichenbacher
- RSA PKCS#1
- Heartbleed
- "CRIME" attack
- RC4 biases
- Triple handshake attack
- Cross-protocol DROWN
- ECDH attack
- Rizov & Duong
- Rizov & Duong
- SSL 2.0 downgrade
- Ray & Dispensa
- renegotiation attack
- CA breaches
- "Golden and Wigan" attack
- "Monskypiro" attack

- Web browsers: Chrome, Firefox, IE, Safari
- Web servers: Apache, IIS
- Application SDKs
- Certificates
- Goldman & Wigan attack
- "Monskypiro" attack
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

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
I'd like to renegotiate TLS handshake

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

Client

\(\text{TLS handshake}_{AB}\)

Eve

\(\text{TLS recordlayer}_{EB}\)

\(m_E\)

\(m_E\)

\(m_E \parallel m_A\)

Server (application)

\(\text{Application receives concatenation of record layers}\)

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

\(\text{Server (application)}\)

\(m_A\)

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

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

Server’s web server receives
   \( m_E \parallel m_A = \text{"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.
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 87% deployment (SSL Pulse, July 2, 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.
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. Transformation Thm: tagged-ACCE-fin + renegotiation countermeasure, => secure renegotiable ACCE.

5. Prove TLS-DHE satisfies tagged-ACCE-fin
<table>
<thead>
<tr>
<th>Secure renegotiable ACCE</th>
</tr>
</thead>
</table>

### 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 RFC 5746 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>
<tbody>
<tr>
<td>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.</td>
<td>- TLS without fixes is <strong>not</strong> a weakly secure renegotiable ACCE.</td>
</tr>
<tr>
<td></td>
<td>- TLS with RFC 5746 fixes is a weakly secure renegotiable ACCE.</td>
</tr>
<tr>
<td></td>
<td>- (This is probably good enough.)</td>
</tr>
</tbody>
</table>
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.
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)
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
(neg) auth\textsubscript{1}
(kex\textsubscript{1})
conf\textsubscript{1}
int\textsubscript{1}

ciphersuite 2
(neg) auth\textsubscript{2}
(kex\textsubscript{2})
conf\textsubscript{2}
int\textsubscript{2}

ciphersuite 3
(neg) auth\textsubscript{3}
(kex\textsubscript{3})
conf\textsubscript{3}
int\textsubscript{3}
In practice, TLS and SSH servers use the same long-term key for all ciphersuites.

The reality of multi-ciphersuite usage

In practice, TLS and SSH servers use the same long-term key for all 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 ServerDHPParams 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.
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”
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?
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
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.
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
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
Queensland University of Technology

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