Triple Handshake: can cryptography, formal methods, and applied security be friends?

http://miTLS.org

Karthik Bhargavan
Markulf Kohlweiss
with
Antoine Delignat-Lavaud, Cédric Fournet,
Alfredo Pironti, Pierre-Yves Strub,
Santiago Zanella Beguelin
Transport Layer Security (1994—)

The default secure channel protocol?
HTTPS, 802.1x, VPNs, files, mail, VoIP, ...

20 years of attacks, fixes, and extensions
1994 Netscape’s Secure Sockets Layer
1996 SSL3
1999 TLS1.0 (RFC2246)
2006 TLS1.1 (RFC4346)
2008 TLS1.2 (RFC5246)
2015 TLS1.3?

Many implementations
OpenSSL, SecureTransport, NSS, SChannel, GnuTLS, JSSE, PolarSSL, ...
many bugs, attacks, patches every year

Many papers
Well-understood, detailed specs
many security theorems...
mostly for small simplified models of TLS
Goal: a secure channel

Security Goal: As long as the adversary does not control the long-term credentials of the client and server, it cannot

- Inject forged data into the stream (authenticity)
- Distinguish the data stream from random bytes (confidentiality)

Secure channels for the Web

Security Goal: As long as the client is honest and the adversary does not know the server’s private key, it cannot
- Inject forged data into the data stream (authenticity)
- Distinguish the data stream from random bytes (confidentiality)

More formally: SACCE [Krawczyk et al. ’13]
TLS protocol overview

Hello
- Protocol negotiation
  - Agree on version
  - Agree on ciphersuite
  - Determines all crypto algos

KEM
- Authenticated Key Exchange
  - Verify server/client identity
  - Generate master secret
  - Derive connection keys

Finished
- Key, transcript confirmation
  - Completes authentication
  - Matches transcripts
  - Authenticated encryption

AppData
- Application data streams
  - Full duplex channel
  - Authenticated encryption
Common TLS configurations

- **Hello**
  - Client
  - Server
  - 4 protocol versions
  - 100s of ciphersuites
  - 10s of extensions

- **KEM**
  - Client
  - Server
  - RSA key transport
  - DHE/ECDHE with RSA/DSA/ECDSA signatures
  - PSK, SRP, ...

- **Finished**
  - Client
  - Server

- **AppData**
  - Client
  - Server
  - HMAC with AES-CBC
  - HMAC with RC4
  - AES-GCM
**TLS_RSA_WITH_AES_128_CBC_SHA**

- **Hello**
  - Client: \( cr \)
  - Server: \( sr \)
  - Client and server exchange fresh nonces
  - TLS 1.2 (mandatory cipher suite)

- **KEM**
  - Cert\(_S\)
  - \( \text{rsa-enc(pms,pk}_S\)\)
  - Server certificate Cert\(_S\) supports RSA encryption
  - Client generates pms
  - ms, keys \( (k) \) derived from pms, cr, sr

- **Finished**
  - \( \text{ae(0||tag}_C,k) \)
  - \( \text{ae(0||tag}_S,k) \)
  - tag\(_C\), tag\(_S\) derived from ms + SHA-1 hash of handshake log

- **AppData**
  - \( \text{ae(i||d}_i,k) \)
  - Authenticated encryption MAC-then-Pad-then-Encrypt HMAC with AES-CBC
TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256

**Hello**
- Client
  - cr
  - sr
- Server
  - TLS 1.2 (Google's cipher suite)
  - Client and server exchange fresh nonces

**KEM**
- Server
  - cert_s
  - ecdsa-sign((G, g^y), sk_s)
  - g^x
- Client
  - Server picks group/curve
  - signs group, key share
  - pms = g^{xy}
  - ms, keys (k) derived from pms, cr, sr

**Finished**
- Client
  - ae(0||tag_c,k)
- Server
  - ae(0||tag_s,k)
- Server
  - tag_c, tag_s derived from ms + SHA-256 hash of handshake log

**AppData**
- Client
  - ae(i||d_i,k)
- Server
  - authenticated encryption AEAD with AES-GCM
Cryptographic weaknesses

Many obsolete crypto constructions

- RSA encryption with PKCS#1 v1.5 padding (Bleichenbacher)
- MAC-then-Pad-then-Encrypt with AES-CBC (Padding oracle)
- Compress-then-MAC-then-Pad-then-Encrypt (CRIME)
- Chained IVs in TLS 1.0 AES-CBC (BEAST)
- RC4 key biases

Countermeasures

- Disable these features: SSL3, compression, RC4
- Implement ad-hoc mitigations very very carefully:
  - empty fragment to initialize IV for TLS 1.0 AES-CBC
  - constant time mitigation for Bleichenbacher attacks
  - constant-time plaintext length-hiding HMAC to prevent Lucky 13
Other implementation challenges

Memory safety
Buffer overruns leak secrets

Missing checks
Forgetting to verify signature/MAC/certificate bypasses crypto guarantees

Certificate validation
ASN.1 parsing, wildcard certificates

State machine bugs
Most TLS implementations don’t conform to spec
Unexpected transitions break protocol (badly)

(EarlyCCS, OpenSSL, …)
Implementing TLS correctly

Use formal methods!

• Use a type-safe programming language
  • OCaml, F#, Java, C#,...
  • (No buffer overruns, no Heartbleed)

• Verify the logical correctness of your code
  • Use a software verifier: Why3, F7/F*, Boogie, Frama-C,...

• Link software invariants to cryptographic guarantees
  • Use a crypto verifier: EasyCrypt, CryptoVerif, ProVerif
  • Hire a cryptographer!
miTLS: a verified implementation

- Reference implementation of TLS 1.2 in F#
  - 7000 lines of code, 3000 lines of logical specification
  - Automated proofs by typechecking with F7
- Supports major protocol versions, ciphersuites

<table>
<thead>
<tr>
<th>Protocol Versions</th>
<th>Key exchange</th>
<th>Record encryption</th>
<th>Record HMAC</th>
<th>Extensions</th>
</tr>
</thead>
<tbody>
<tr>
<td>TLS 1.2</td>
<td>RSA</td>
<td>AES.256.GCM</td>
<td>SHA384</td>
<td>Secure renegotiation</td>
</tr>
<tr>
<td></td>
<td>DHE.DSS</td>
<td>AES.128.GCM</td>
<td>SHA256</td>
<td></td>
</tr>
<tr>
<td></td>
<td>DHE.RSA</td>
<td>AES.256.CBC</td>
<td>SHA</td>
<td></td>
</tr>
<tr>
<td></td>
<td>DH.RSA</td>
<td>AES.128.CBC</td>
<td>MD5</td>
<td></td>
</tr>
<tr>
<td></td>
<td>DH.DSA</td>
<td>3DES.EDE_CBC</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>DH.anon</td>
<td>RC4.128</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- How does this verification link to crypto assumptions and the secure channel goal?
miTLS: a verified implementation

miTLS

A verified reference TLS implementation

miTLS is a verified reference implementation of the TLS protocol. Our code fully supports its wire formats, ciphersuites, sessions and connections, re-handshakes and resumptions, alerts and errors, and data fragmentation, as prescribed in the RFCs; it interoperates with mainstream web browsers and servers. At the same time, our code is carefully structured to enable its modular, automated verification, from its main API down to computational assumptions on its cryptographic algorithms.

- How does this verification link to crypto assumptions and the secure channel goal?
For agility parameter $p^*$, parameter set $P$, $\text{Adv}^{\text{NR/OW-PCA}}_{p^*, P}(A) \overset{\text{def}}{=} \Pr[\text{NR/OW-PCA} : 1]$

**Oracle** $\text{PCO}(p, k, c) \overset{\text{def}}{=} \begin{cases} \text{if } p \notin P \text{ then return } \bot \\ k' \leftarrow \text{dec}(p, sk, c) \end{cases}$ (Plaintext Checking Oracle)

There exist adversaries $B$ and $C$ running in time $t + O(q_{\text{DEC}} \cdot q_{\text{KEF}})$ such that

$$\text{Adv}^{\text{RCCA}}_{p^*, P}(A) \leq 2 \left( \text{Adv}^{\text{NR-PCA}}_{pv^*, P'}(B) + \text{Adv}^{\text{OW-PCA}}_{pv^*, P'}(C) + 2^{|pv| - |pms|} (q_{\text{KEF}} + q_{\text{DEC}}) \right)$$

where $P'$ includes all $pv$ such that $(pv, h) \in P$

Proof formalized and checked in EasyCrypt (3,000 lines)
State of the art

Monolithic proof (ACCE model), does not cover TLS-RSA

[Krawczyk, Paterson, Wee ’13] Security for TLS-DHE + TLS-RSA + authenticated encryption
KEM abstraction (SACCE model), single ciphersuite, does not cover resumption, renegotiation

[Bhargavan’14 et al.] Comprehensive modular treatment of a TLS handshake implementation
Multi-ciphersuite, multi-handshake security
Cryptographic core of TLS

Client

\(\ell_C \leftarrow \$;\)

\(\text{ClientHello}[\ell_C] \rightarrow \ell_S \leftarrow \$; \ell := \ell_C \parallel \ell_S;\)

\(\ell := \ell_C \parallel \ell_S;\)

\(pk := pk(\text{cert}_S)\)

\(c, ms \leftarrow \text{Enc}(pk, \ell)\)

\(k := \text{KDF}(ms, \ell)\)

\(\text{ServerHello}[\ell_S]\)

\(\text{ServerCertificate}[\text{cert}_S]\)

\(\text{ServerHelloDone}\)

\(\text{ClientKeyExchange}[c] \rightarrow ms \leftarrow \text{Dec}(sk, \ell, c)\)

\(\text{ClientFinished}[\text{tag}_C] \rightarrow \) log\(_C := \{\text{all prior messages}\}\)

\(\text{tag}_C := \text{MAC}(ms, "C", \text{log}_C)\)

\(\text{ServerFinished}[\text{tag}_S] \leftarrow \) log\(_S := \{\text{all prior messages}\}\)

\(\text{tag}_S := \text{MAC}(ms, "S", \text{log}_S)\)

AppData
Cryptography of TLS ‘as it is’

Client:
\[ l_C \leftarrow \$; \ a_C \leftarrow cf g_C .a_C \]

\[ \ell := \ell_C \parallel \ell_S; \ a := \text{alg}_C(cf g_C, as) \]

\[ \ell \leftarrow \ell_C \parallel \ell_S; \ s_i d \leftarrow \$; \ c e r t S := cf g_S .c e r t; \ c e r t C := \perp \]

\[ p k := pk(c e r t S); \]

\[ c, m s \leftarrow \text{Enc}_S(p_E, p k, \ell) \]

\[ k := \text{KDF}(p_D, m s, \ell, r) \]

\[ \ell \leftarrow \ell_C \parallel \ell_S; \ s_i d \leftarrow \$; \]

\[ a, a_s := \text{alg}_S(cf g_S, a_C) \]

Server:

\[ \text{ClientHello} [\ell_C, a_C, t a g_C] \rightarrow \]

\[ \text{ServerCertificate}[c e r t S] \rightarrow \text{ServerHelloDone} \]

\[ \ell \leftarrow \ell_C \parallel \ell_S; \ s_i d \leftarrow \$; \]

\[ \text{ClientKeyExchange}[c] \rightarrow \text{ms} \leftarrow \text{Dec}_C(p_E, s k, \ell, c) \]

\[ \log_C := \{\text{all prior epoch messages}\} \]

\[ \text{ClientFinished}[t a g_C] \rightarrow \]

\[ \log_S := \{\text{all prior epoch messages}\} \]

\[ \text{ServerFinished}[t a g_S] \rightarrow \]

\[ \text{complete} := 1; \text{store} (\ell, s_i d, m s) \]
Cryptographic security goals

If a client completes with an honest server’s certificate and (all) strong algorithms, then

**Agreement**: there must be a server that agrees on all handshake variables \((a, cert, ms, k, tag, \ldots)\)

**Authenticity**: each endpoint only accepts a prefix of the data sequence send by its peer

If connection is gracefully closed, then all sent data has been accepted

**Confidentiality**: the data sequences in both directions are indistinguishable from random

(vice versa for server if client is authenticated)
The concrete implementation

Handshake/CCS

RSAKey

Sig

Cert

Extensions

Nonce

DH

SessionDB

Handshake (and CCS)

KEM

KEF

KDF/MAC

Alert Protocol

Alert

Datastream

AppData

Enc

Encode

TLS Fragment

TLS Record

TLS API

TLS

dispatch

Application

Auth

RPC

Adversary

Untyped API

Untyped Adversary
miTLS API & ideal functionality (outline)

Standard socket API with embedded security specification

- Abstract types for confidentiality (a la information flow)
- Refinements for authenticity (a la contracts/pre-/post-conditions)

```haskell
module SecuritySpec where

type Connection // for each local instance of the protocol
  type (;c:Connection) AppData

// creating new client and server instances
val connect: TcpStream -> Params -> Connection
val accept: TcpStream -> Params -> Connection

// reading data
val read: c:Connection -> (;c) IOResult_i

// writing data
val write: c:Connection -> data:(;c) AppData -> (;c,data) IOResult_o

// triggering new handshakes, and closing connections
val rehandshake: c:Connection -> Connection Result
val request: c:Connection -> Connection Result
val shutdown: c:Connection -> TcpStream Result
```
Security of master secret KEM

We prove Handshake security assuming the master secret KEM is secure under agile Replayable Chosen-Ciphertext Attacks (IND-RCCA)

\[
\begin{align*}
\text{Client} & \quad pk := pk(\text{cert}s) \\
& \quad c, ms \leftarrow \text{Enc}(p^*, pk, \ell) \\
& \quad k := D.KDF(p_d, ms, \ell, r) \\
\text{Server} & \quad \text{ClientKeyExchange}[c] \quad \rightarrow \quad ms' \leftarrow \text{Dec}(p \in P, sk, \ell, c)
\end{align*}
\]

For \( p^* \) an agility parameter, \( P \) a set of parameters \( \text{Adv}^{\text{RCCA}}_{p^*, P}(\mathcal{A}) \) is defined as

\[
2 \Pr[\text{RCCA} : 1] - 1
\]

**Game RCCA**

\[
\begin{align*}
pk, sk & \leftarrow \text{KeyGen}() \\
K, L & := \emptyset \\
b & \leftarrow \{0, 1\} \\
b' & \leftarrow \mathcal{A}^{\text{ENC}, \text{DEC}}(pk) \\
\text{return} & \ (b' = b)
\end{align*}
\]

**Oracle ENC(\ell)**

\[
\begin{align*}
\text{if} \ \ell \in L & \ \text{then return} \ \bot \\
k_0, c & \leftarrow \text{Enc}(p^*, pk, \ell) \\
k_1 & \leftarrow $ \\
K(\ell) & := K(\ell) \cup \{k_0, k_1\} \\
\text{return} & \ k_b, c
\end{align*}
\]

**Oracle DEC(p, \ell, c)**

\[
\begin{align*}
\text{if} \ \ell \in L \lor p \notin P & \ \text{then return} \ \bot \\
L & := L \cup \{\ell\} \\
k & \leftarrow \text{Dec}(p, sk, \ell, c) \\
\text{if} \ k & \in K(\ell) \ \text{then return} \ \bot \\
\text{return} & \ k
\end{align*}
\]
Security of master secret KEM

We prove Handshake security assuming the master secret KEM is secure under agile Replayable Chosen-Ciphertext Attacks (IND-RCCA)

\[
\begin{align*}
\text{Client} & \quad \text{Server} \\
\text{pk} & := \text{pk(\text{cert}_S)} \\
c, ms & \leftarrow \text{Enc}(p^*, pk, \ell) \\
k & := \text{D.KDF}(p_D, ms, \ell, r) \\
\text{ClientKeyExchange}[c] & \quad ms' \leftarrow \text{Dec}(p \in P, sk, \ell, c)
\end{align*}
\]

For \( p^* \) an agility parameter, \( P \) a set of parameters \( \text{Adv}^{\text{RCCA}}_{p^*, P}(A) \) defined as

\[
\text{Game } \text{RCCA} \overset{\text{def}}{=} \\
\begin{align*}
\text{pk, sk} & \leftarrow \text{KeyGen}() \\
K, L & := \emptyset \\
b & \leftarrow \{0, 1\} \\
b' & \leftarrow A^{\text{ENC,DEC}}(pk) \\
\text{return } (b' = b)
\end{align*}
\]

\[
\text{Oracle } \text{ENC}(\ell) \overset{\text{def}}{=} \\
\begin{align*}
\text{if } \ell \in L & \text{ then return } \bot \\
k_0, c & \leftarrow \text{Enc}(p^*, pk, \ell) \\
k_1 & \leftarrow \$ \\
K(\ell) & := K(\ell) \cup \{k_0, k_1\} \\
\text{return } k_b, c
\end{align*}
\]

\[
\text{Oracle } \text{DEC}(p, \ell, c) \overset{\text{def}}{=} \\
\begin{align*}
\text{if } \ell \in L \lor p \notin P & \text{ then return } \bot \\
L & := L \cup \{\ell\} \\
k & \leftarrow \text{Dec}(p, sk, \ell, c) \\
\text{if } k \in K(\ell) & \text{ then return } \bot \\
\text{return } k
\end{align*}
\]
Sufficient assumptions on the pms-KEM

For agility parameter $p^*$, parameter set $P$, $\text{Adv}_{p^*,P}^{\text{NR/OW-PCA}}(\mathcal{A}) \overset{\text{def}}{=} \Pr[\text{NR/OW-PCA}_P : 1]$

**Oracle** $\text{PCO}(p,k,c) \overset{\text{def}}{=} \begin{cases} \bot & \text{if } p \notin P \\ k' \leftarrow \text{dec}(p,sk,c) \\ \text{return } (k' = k) \end{cases}$

(Plaintext Checking Oracle)

(One-Wayness)

**Game** $\text{OW-PCA} \overset{\text{def}}{=} \begin{align*} & pk, sk \leftarrow \text{keygen()} \\ & k^*, c^* \leftarrow \text{enc}(p^*,pk) \\ & k \leftarrow A^{\text{PCO}}(pk,c^*) \\ & \text{return } (k = k^*) \end{align*}$

**Game** $\text{NR-PCA} \overset{\text{def}}{=} \begin{align*} & pk, sk \leftarrow \text{keygen()} \\ & k^*, c^* \leftarrow \text{enc}(p^*,pk) \\ & c \leftarrow A^{\text{PCO}}(pk,c^*) \\ & \text{return } (c \neq c^* \land k^* = \text{dec}(p^*,sk,c)) \end{align*}$

(Non-Randomizability)
Sufficient assumptions on the pms-KEM

For agility parameter $p^*$, parameter set $P$, \( \text{Adv}^{\text{NR/OW-PCA}}_{p^*, P}(\mathcal{A}) \stackrel{\text{def}}{=} \Pr[\text{NR/OW-PCA} : 1] \)

**Oracle** \( \text{PCO}(p, k, c) \) \( \stackrel{\text{def}}{=} \)
  
  if \( p \notin P \) then return \( \perp \)
  
  \( k' \leftarrow \text{dec}(p, sk, c) \)
  
  return \( (k' = k) \)

(Plaintext Checking Oracle)

**One-Wayness**

**Game** \( \text{OW-PCA} \) \( \stackrel{\text{def}}{=} \)

\( pk, sk \leftarrow \text{keygen}() \)

\( k^*, c^* \leftarrow \text{enc}(p^*, pk) \)

\( k \leftarrow A^{\text{PCO}}(pk, c^*) \)

return \( (k = k^*) \)

(Non-Randomizability)

**Game** \( \text{NR-PCA} \) \( \stackrel{\text{def}}{=} \)

\( pk, sk \leftarrow \text{keygen}() \)

\( k^*, c^* \leftarrow \text{enc}(p^*, pk) \)

\( c \leftarrow A^{\text{PCO}}(pk, c^*) \)

return \( c \neq c^* \land \)

\( k^* = \text{dec}(p^*, sk, c) \)
Sufficient assumptions on the pms-KEM

For agility parameter $p^*$, parameter set $P$, $\text{Adv}^{\text{NR/OW-PCA}}_{p^*, P}(A) \overset{\text{def}}{=} \Pr[\text{NR/OW-PCA} : 1]$

**Oracle** $\text{PCO}(p, k, c) \overset{\text{def}}{=} \text{(Plaintext Checking Oracle)}$

if $p \not\in P$ then return $\bot$

$k' \leftarrow \text{dec}(p, sk, c)$

There exist adversaries $B$ and $C$ running in time $t + O(q_{\text{DEC}} \cdot q_{\text{KEF}})$ such that

$$\text{Adv}^{\text{RCCA}}_{p^*, P}(A) \leq 2 \left( \text{Adv}^{\text{NR-PCA}}_{pv^*, P'}(B) + \text{Adv}^{\text{OW-PCA}}_{pv^*, P'}(C) + 2^{1|pv| - |pms|} (q_{\text{KEF}} + q_{\text{DEC}}) \right)$$

where $P'$ includes all $pv$ such that $(pv, h) \in P$

Proof formalized and checked in EasyCrypt (3,000 lines)
Main crypto result: concrete TLS & ideal TLS are computationally indistinguishable.

We prove that ideal miTLS meets its secure channel specification using standard program verification (typing).

Safe, except for a negligible probability $\approx \epsilon$

Safe by typing (info-theoretically) $\approx \epsilon$
Security theorem

Proof automation
7,000 lines of F#
checked against
3,000 lines of F7
+ 3,000 lines of EasyCrypt
for the core key exchange

Ongoing work
ECDHE, GCM, Certificates, Side-channels
Mission accomplished?
## Reusing established sessions

<table>
<thead>
<tr>
<th>Client</th>
<th>Server</th>
</tr>
</thead>
</table>
| **Hello** | Protocol negotiation  
• Agree on version  
• Agree on ciphersuite  
Determines all crypto algos |
| **KEM** | Authenticated Key Exchange  
• Verify server/client identity  
• **Generate master secret**  
• Derive connection keys |
| **Finished** | Key, transcript confirmation  
• Completes authentication  
• Matches transcripts  
• Authenticated encryption |
| **AppData** | Application data streams  
• Full duplex channel  
• Authenticated encryption |
Efficiency: One round-trip before client sends data

Security?
Security of session resumption

If a client completes an abbreviated handshake and the server in the original handshake was honest, and the master secret has not been leaked, then

**Agreement:** there must be a unique server that agrees on the variables in both the abbreviated handshake and the original handshake

**Authenticity and Confidentiality:** (as usual)

(vice versa for server if client was authenticated)
User authentication over TLS

- Common Pattern
  - **Outer**: server-authenticated TLS
  - **Inner**: user authentication protocol

- Many examples
  - SASL, GSSAPI, EAP, ...
  - TLS Renegotiation with client certificate

- Inner authentication *blesses* outer unauthenticated channel

*Need to strongly bind the two protocol layers together!*
Generic credential forwarding attack
Simplified version of [Asokan, Niemi, Nyberg’02]

- Suppose $u$ uses same authentication credential at both $M$ and $S$
- $M$ forwards $S$'s authentication challenge to $C$
- $M$ forwards $C$'s response to $S$

- $M$ can log in as $u$ at $S$!
TLS renegotiation attack [2009]
Martin Rex’s Version

- Suppose $u$ uses same client cert to log in to both $M$ and $S$
- $M$ forwards $S$’s renegotiation request to $C$
- $M$ forwards renego handshake between $C$ and $S$

- $S$ concatenates data sent by $M$ to data sent by $u$!
Binding user auth to TLS channels

Computing a channel identifier (cid):
- \( f(\text{master secret}) \) (EAP)
- \( f(\text{handshake log}) \) (Renegotiation Indication,SASL)

Does not work if \( M \) can ensure that \( \text{cid} = \text{cid}' \)

Extract TLS-level channel identifier cid

Bind cid to User authentication
Security of (fixed) renegotiation

[Giesen et al ’13]

If an endpoint completes renegotiation with an honest peer and (all) strong algorithms, then

Agreement: there must be a peer endpoint that agrees on all variables in the new and the old handshake (even if the peer in the old handshake was compromised or unauthenticated)

More generally, in a sequence of handshakes, the last handshake guarantees agreement on all previous ones
Triple Handshakes and Cookie Cutters: Breaking and Fixing Authentication over TLS

Karthikeyan Bhargavan*, Antoine Delignat-Lavaud*, Cédric Fournet†, Alfredo Pironti* and Pierre-Yves Strub‡

*INRIA Paris-Rocquencourt †Microsoft Research ‡IMDEA Software Institute

Details, demos at:
http://secure-resumption.com
Key Synchronization Attack
A malicious server $M$ can ensure that the master secrets in two different connections from $C$-$M$ and $M$-$S$ are the same

RSA Key Synchronization
$M$ re-encrypts $C$’s premaster secret under $S$’s public key
$M$ forces same ciphersuite and nonces on the two handshakes

DHE Key Synchronization
$M$ chooses a “bad” (non-prime) Diffie-Hellman group

Does not break Markulf’s theorem
“If a client completes with an honest server...”

Breaks EAP compound authentication (reenables 2002 attack)
The master secret is not a good channel identifier (it isn’t contributive)
Renegotiation indication channel identifier (handshake log) still works.
Transcript Synchronization Attack
After resumption, a malicious server $M$ can ensure that the master secrets, keys, and handshake logs on two different connections from $C$-$M$ and $M$-$S$ are the same.

Abbreviated agreement
Transcript depends only on master secret, ciphersuite, session ID (no certificates)

Does not break Markulf’s theorem
“If the server in the original handshake was honest...”

Breaks transcript-based channel identifiers
After resumption, handshake log is not a good channel identifier
Breaks tls-unique (SASL), renegotiation indication
User Impersonation Attack (reenables 2009 attack)

\[ \text{cid} = \text{hash(abbreviated handshake log)} \text{ same on both connections} \]

So \( M \) can forward renegotiation between \( C \) and \( S \) unchanged.

Surely this must break Markulf’s multi-handshake theorem?

Renegotiation with honest peer implies agreement on abbreviated handshake, but not on original handshake.

Theorem needs honest peer in original handshake for agreement on all three.

Impact

A malicious website can impersonate any user who uses client certificates on any other website that requires client certificate auth, and supports resumption and renegotiation.
Fix the implementations

Disallow server certificate change during renegotiation
Preferred fix for web browsers (same origin policy)

Use only well-known DH groups and validate DH keys
Preferred fix for TLS libraries (good idea anyway)

Disallow client authentication after resumption
Difficult to enforce. How else can we fix SASL, EAP?

Root problem: master secret is not context bound
Master secret does not depend on server certificate
If we make the master secret a good session identifier, EAP, SASL, and renegotiation indication will all be fixed!
Fix the standard: session hash

• Compute a session hash for every full handshake
  `session_hash = Hash(handshake log)`
  (All messages up to and including ClientKeyExchange)

• Add session hash to master secret derivation:
  `master_secret = PRF(pre_master_secret,
  "extended master secret",
  session_hash)[0..47];`

• Extension draft: `draft-ietf-tls-session-hash-02.txt`
  • Implemented in miTLS, OpenSSL, NSS, PolarSSL, ...
  • Construction built in to TLS 1.3
  • Verification ongoing:
    • changes ms-KEM structure, new stronger security spec
Let’s be friends?

- TLS and its applications pose interesting (!) challenges for academic cryptographers

- To scale cryptographic proofs up to full implementations, we use many formal tools
  - F7, F*, EasyCrypt, Coq, ProVerif, Frama-C
  - We can verify (small) fragments of OpenSSL too

- Still many open verification problems
  - session hash, PKI (ASN.1 parsing), side channels
  - (TLS 1.3, here we come!)
Questions?

- More details, demos, research papers:
  http://secure-resumption.com
  http://mitls.org