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

Inria
INVENTORS FOR THE DIGITAL WORLD
Microsoft
Microsoft Research - Inria
JOINT CENTRE
institute IMDEA software

Transport Layer Security (1994—)

The default secure channel protocol?

HTTPS, 802.1x, VPNs, files, mail, VoIP, …

20 years of attacks, fixes, and extensions

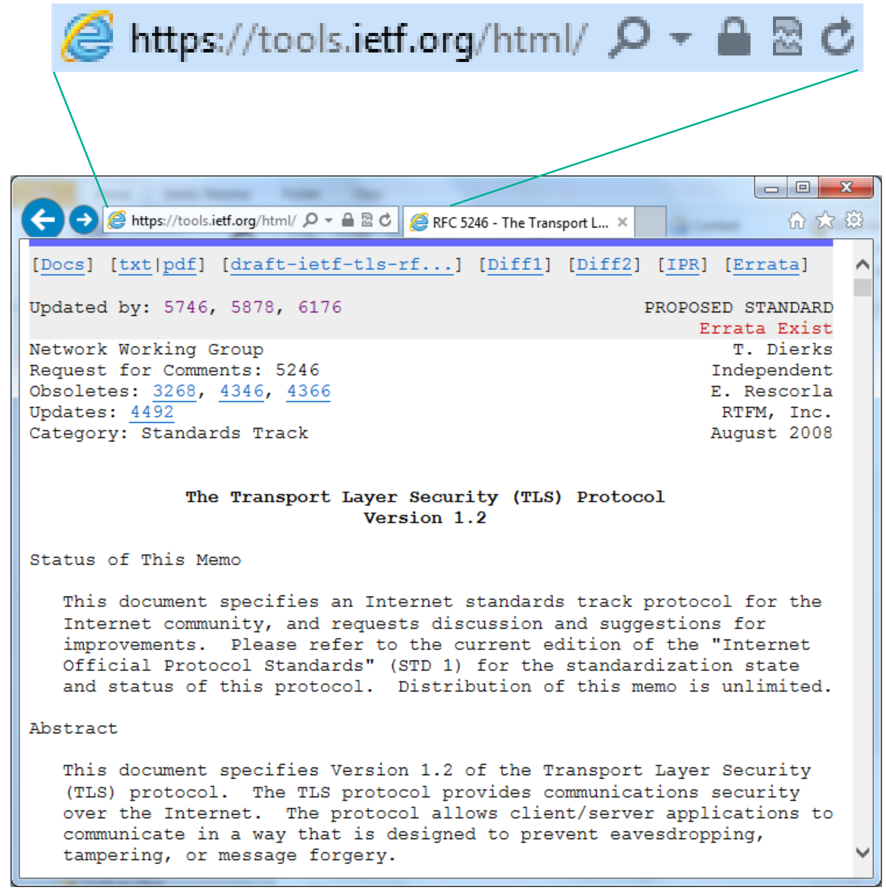
1994	Netscape's Secure Sockets Layer
1996	SSL3
1999	TLS1.0 (RFC2246)
2006	TLS1.1 (RFC4346)
2008	TLS1.2 (RFC5246)
2015	TLS1.3?

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



[Docs] [txt|pdf] [draft-ietf-tls-rf...] [Diff1] [Diff2] [IPR] [Errata]

Updated by: 5746, 5878, 6176 PROPOSED STANDARD
Errata Exist

Network Working Group T. Dierks
Request for Comments: 5246 Independent
Obsoletes: 3268, 4346, 4366 E. Rescorla
Updates: 4492 RTFM, Inc.
Category: Standards Track August 2008

The Transport Layer Security (TLS) Protocol
Version 1.2

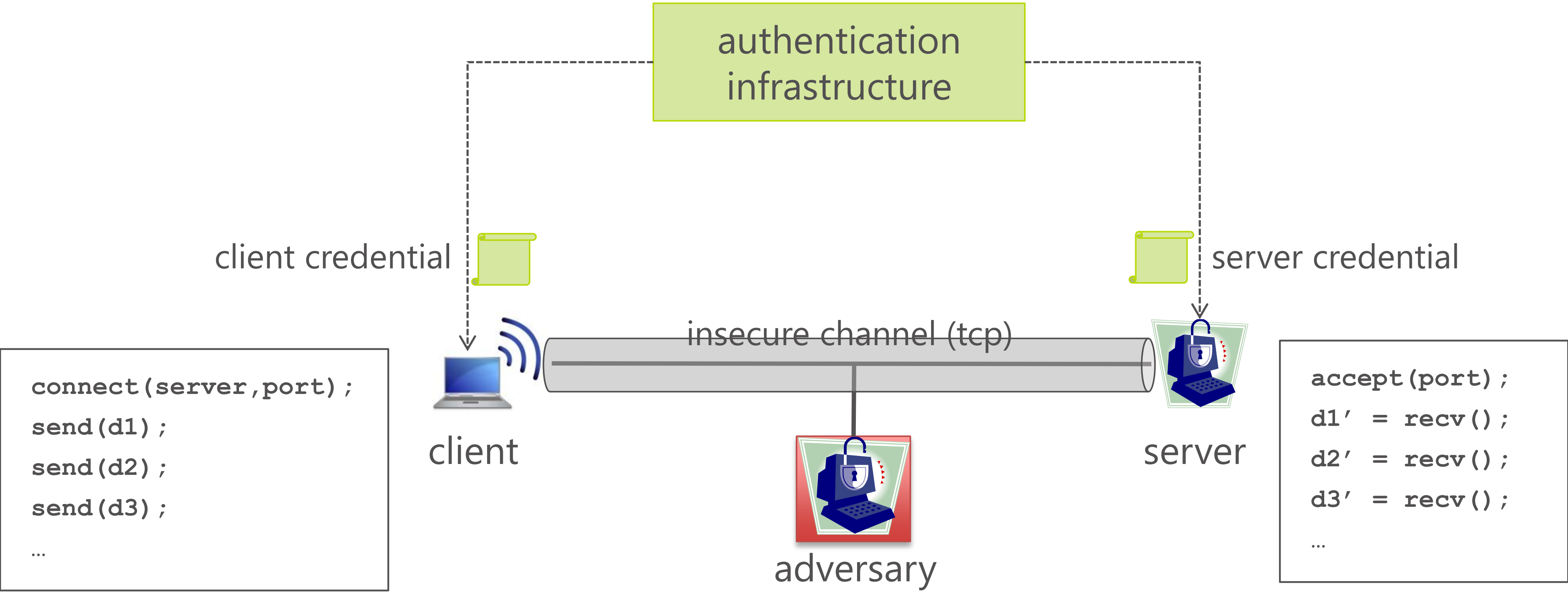
Status of This Memo

This document specifies an Internet standards track protocol for the Internet community, and requests discussion and suggestions for improvements. Please refer to the current edition of the "Internet Official Protocol Standards" (STD 1) for the standardization state and status of this protocol. Distribution of this memo is unlimited.

Abstract

This document specifies Version 1.2 of the Transport Layer Security (TLS) protocol. The TLS protocol provides communications security over the Internet. The protocol allows client/server applications to communicate in a way that is designed to prevent eavesdropping, tampering, or message forgery.

Goal: a secure channel



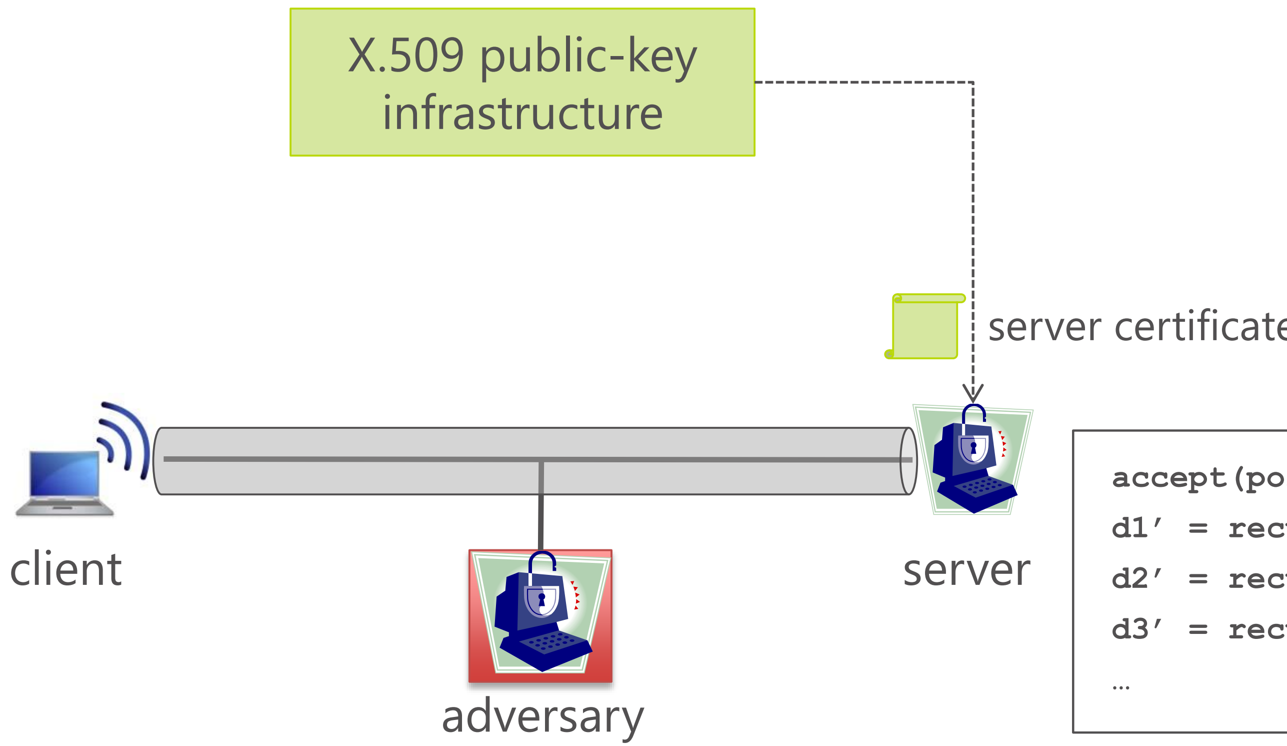
Security Goal: As long as the adversary does not control the long-term credentials of the client and server, it cannot

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

More formally: ACCE [Jager et al. '11] based on sLHAE [Paterson et al '11]

Secure channels for the Web

```
connect(server,port);
send(d1);
send(d2);
send(d3);
…
```



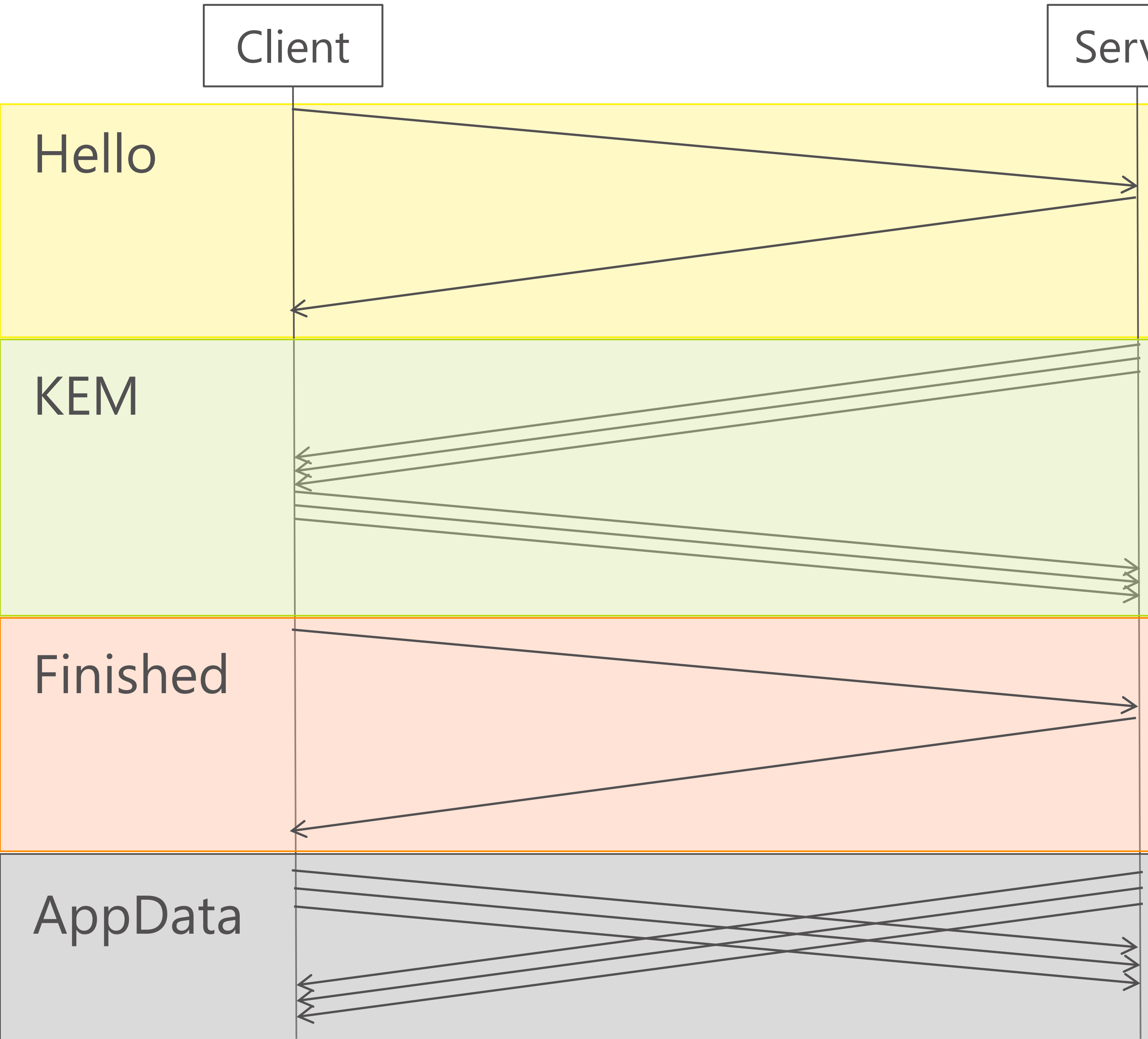

```
accept(port);
d1′ = recv();
d2′ = recv();
d3′ = recv();
…
```

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



Protocol negotiation

- Agree on version
- Agree on ciphersuite

Determines all crypto algos

Authenticated Key Exchange

- Verify server/client identity
- Generate master secret
- Derive connection keys

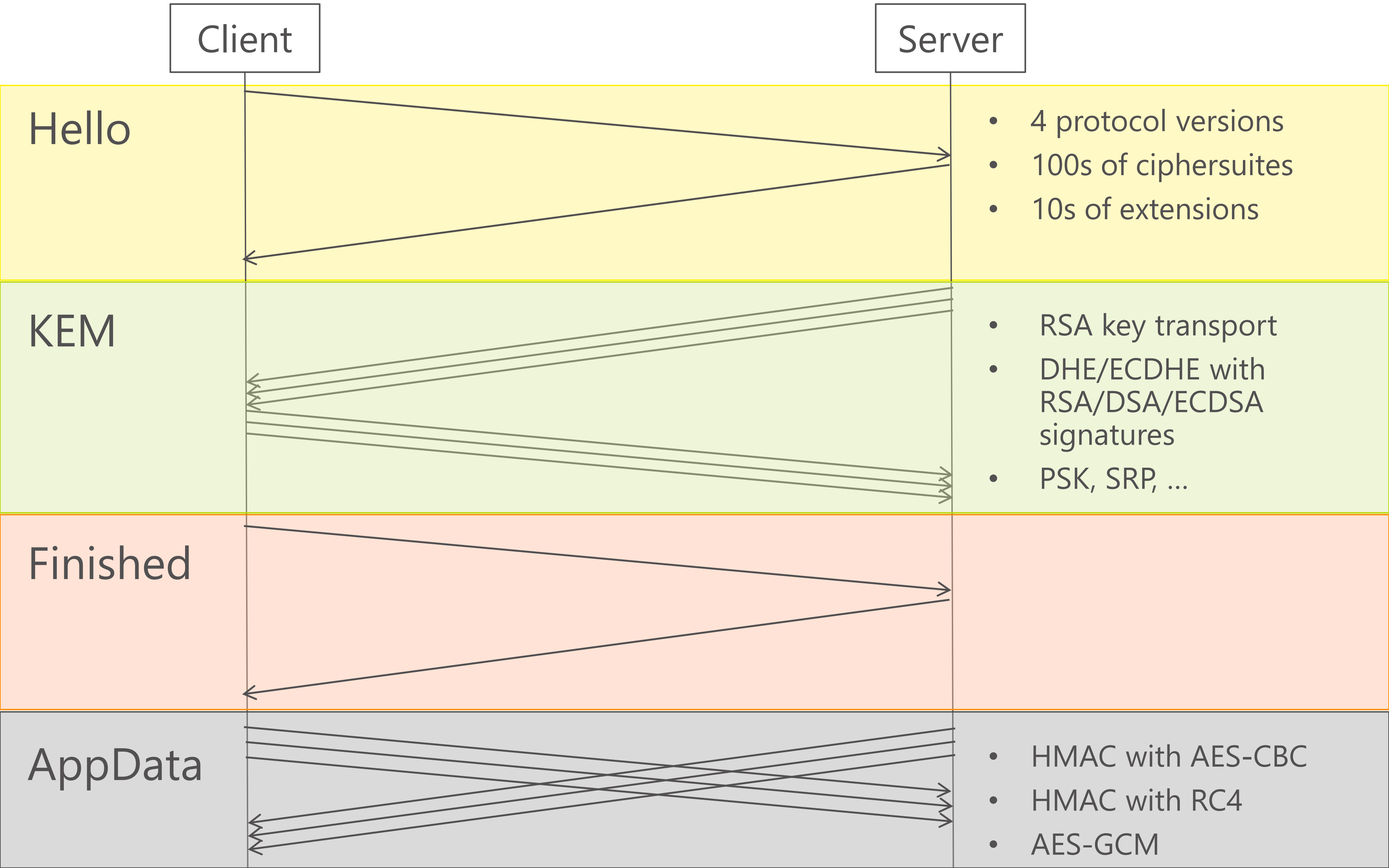
Key, transcript confirmation

- Completes authentication
- Matches transcripts
- Authenticated encryption

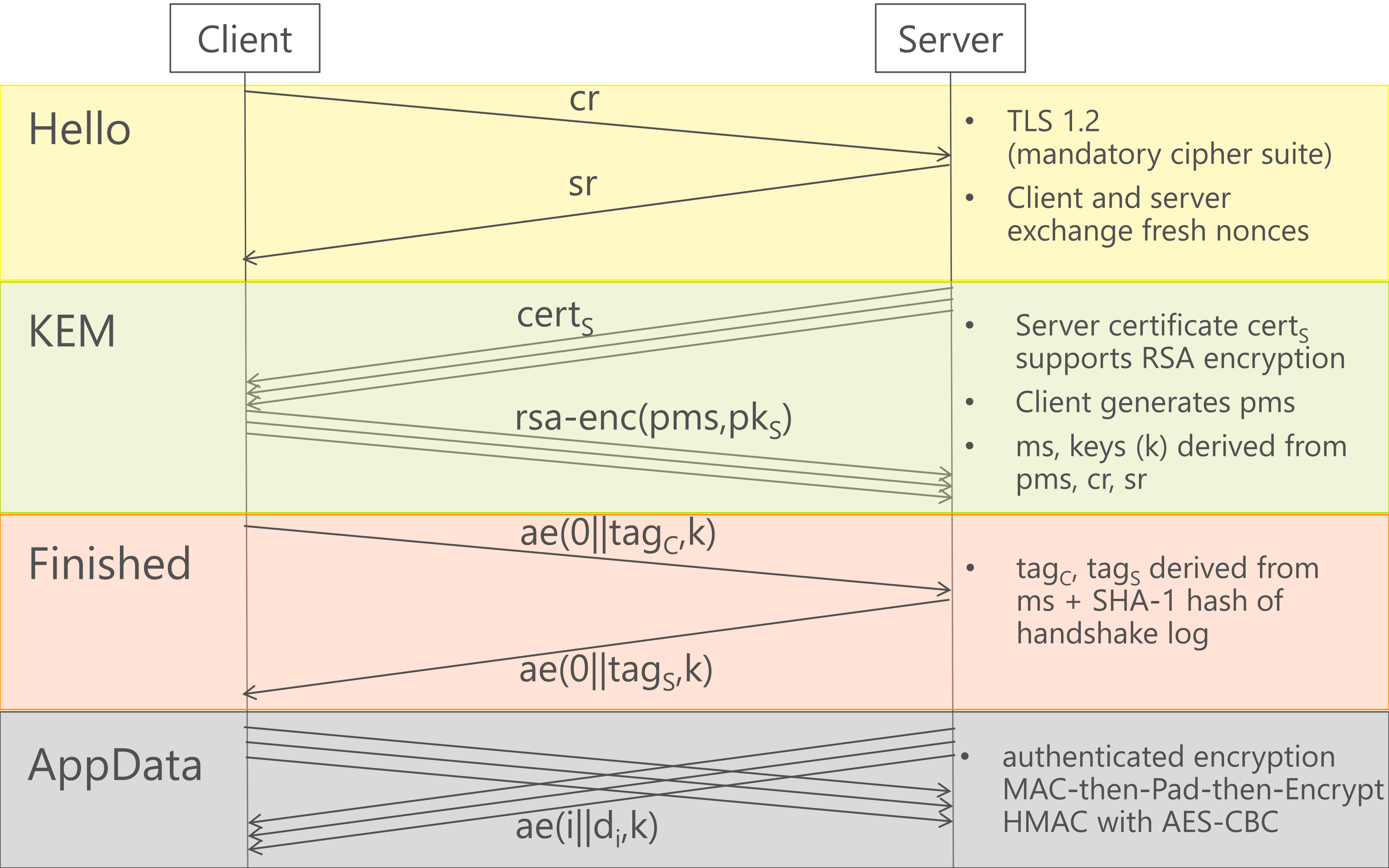
Application data streams

- Full duplex channel
- Authenticated encryption

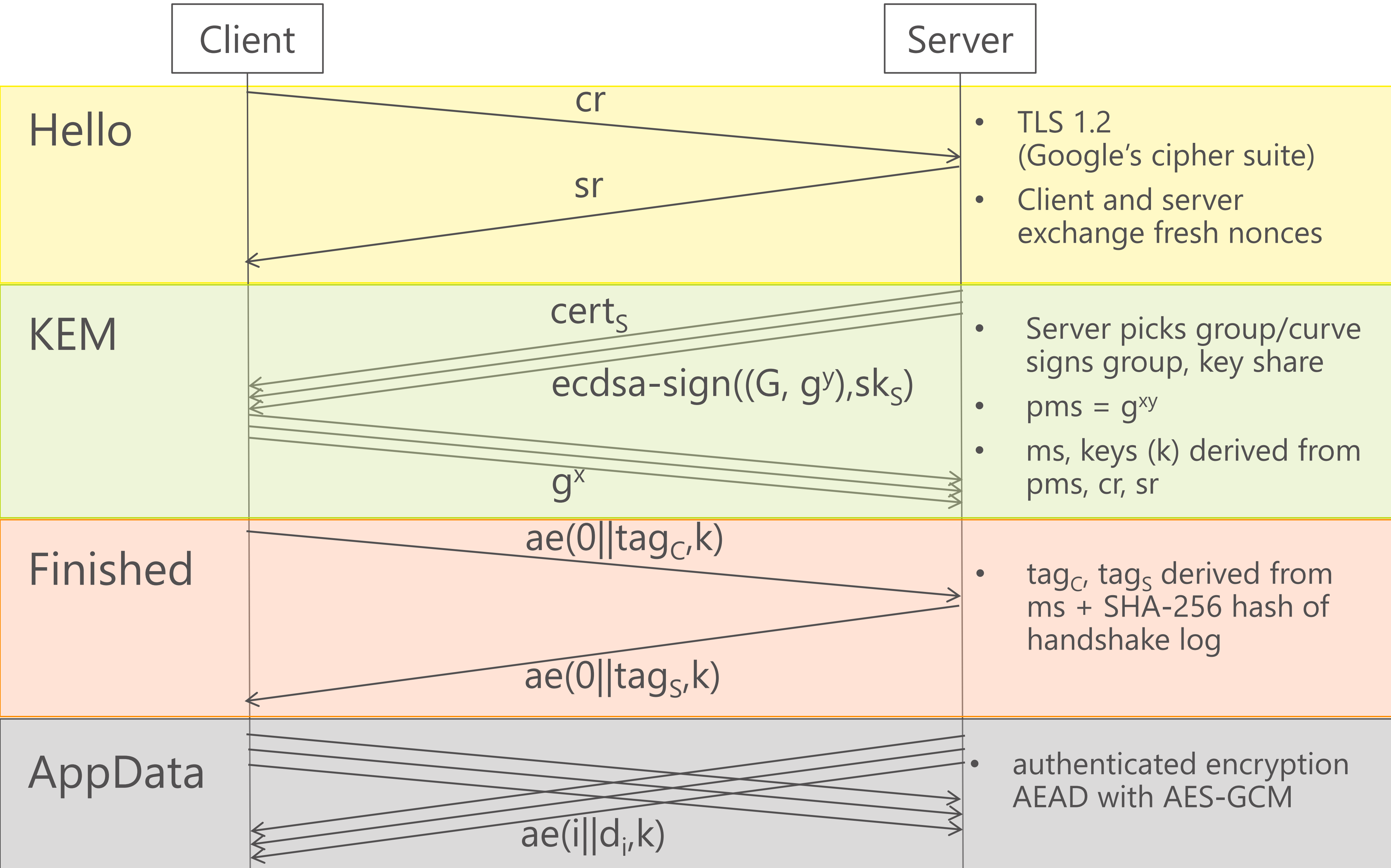
Common TLS configurations



TLS_RSA_WITH_AES_128_CBC_SHA



TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256



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

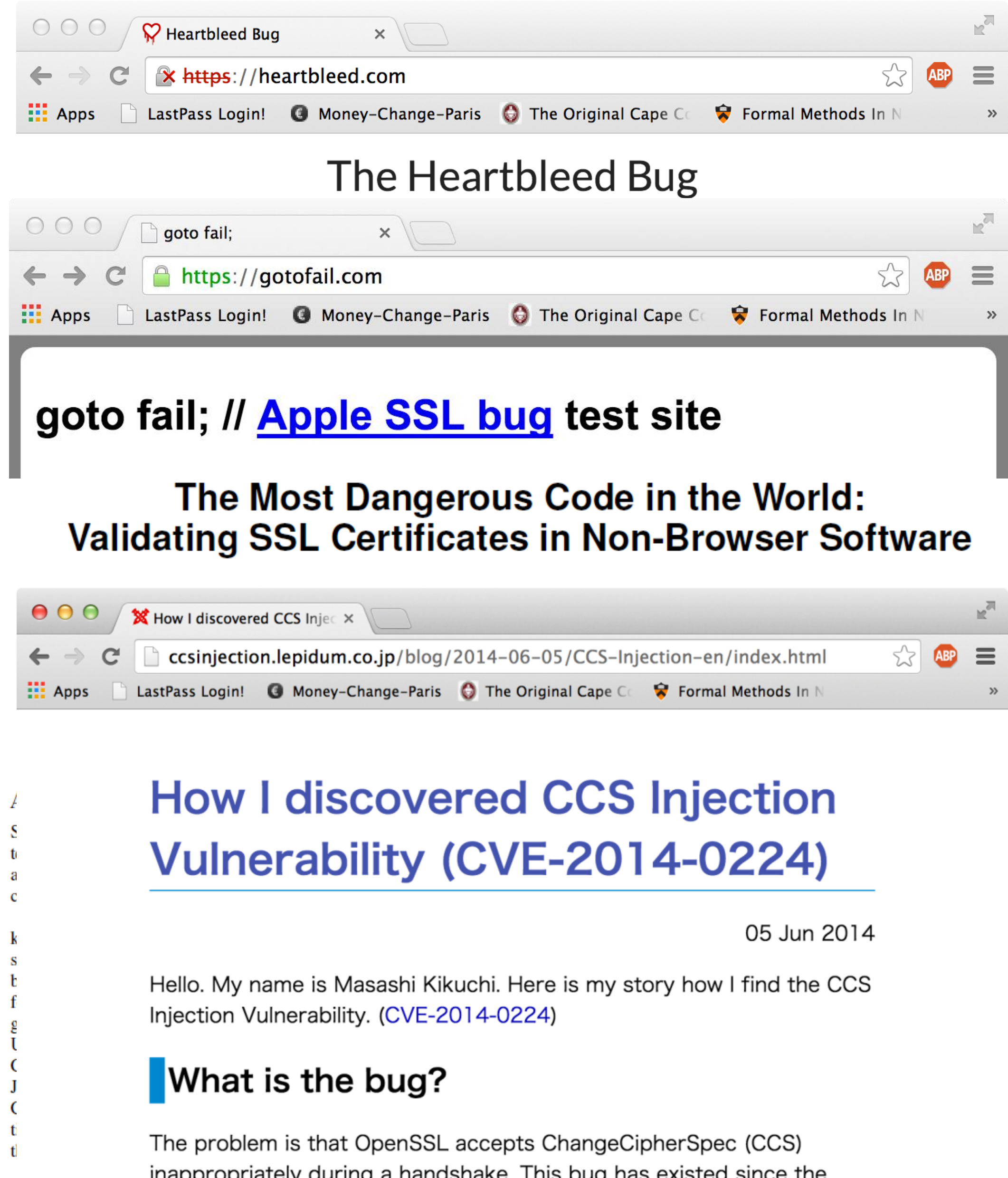
Memory safety
Buffer overruns leak secrets

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!

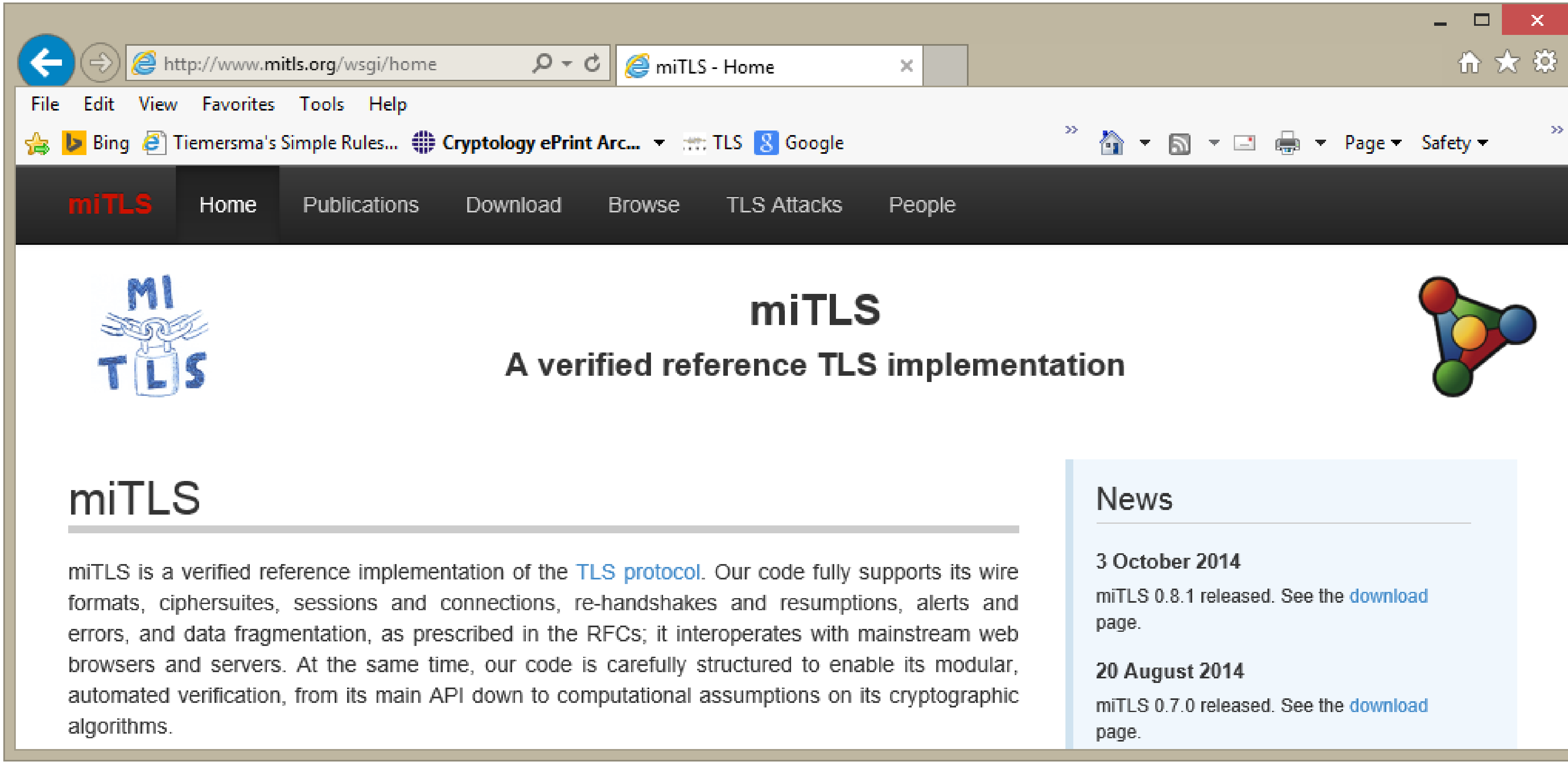
miTLS: a verified implementation

- 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

Protocol Versions	Key exchange	Record encryption	Record HMAC	Extensions
TLS 1.2	RSA	AES_256_GCM	SHA384	Secure renegotiation
TLS 1.1	DHE_DSS	AES_128_GCM	SHA256	
TLS 1.0	DHE_RSA	AES_256_CBC	SHA	
SSL 3	DH_RSA	AES_128_CBC	MD5	
	DH_DSA	3DES_EDE_CBC		
	DH_anon	RC4_128		

- How does this verification link to crypto assumptions and the secure channel goal?

miTLS: a verified implementation



- How does this verification link to crypto assumptions and the secure channel goal?

Sufficient assumptions on the pms-KEM

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

$$
\begin{aligned}
&\textbf{Oracle } \mathsf{PCO}(p, k, c) \stackrel{\text{def}}{=} \qquad \text{(Plaintext Checking Oracle)}\\
&\quad \textbf{if } p \notin P \textbf{ then return } \bot\\
&\quad k' \leftarrow \mathsf{dec}(p, sk, c)
\end{aligned}
$$

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

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

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

Proof formalized and checked in EasyCrypt (3,000 lines)

State of the art

[Jager et al. '11] **Security for TLS-DHE + authenticated encryption in the standard model**

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

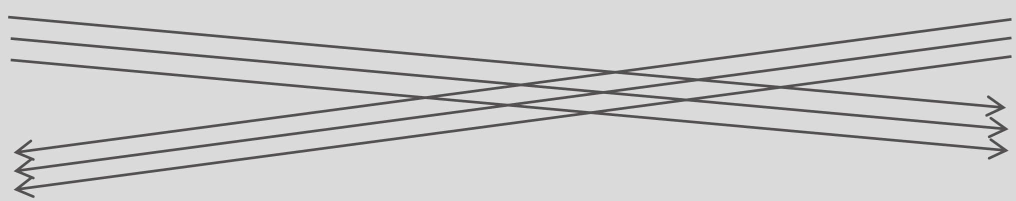
[Bhargavan'14 et al.] **Comprehensive modular treatment of a TLS handshake implementation**

Multi-ciphersuite, multi-handshake security

Cryptographic core of TLS

Client		**Server**
$\ell_C \leftarrow \$;$	`ClientHello`$[\ell_C]$ ⟶	$\ell_S \leftarrow \$;\ \ell := \ell_C \parallel \ell_S;$
$\ell := \ell_C \parallel \ell_S;$ $pk := \mathsf{pk}(certs_S)$ $c, ms \leftarrow \mathsf{Enc}(pk, \ell)$	⟵ `ServerHello`$[\ell_S]$ `ServerCertificate`$[certs_S]$ `ServerHelloDone`	
$k := \mathsf{KDF}(ms, \ell)$	`ClientKeyExchange`$[c]$ ⟶	$ms \leftarrow \mathsf{Dec}(sk, \ell, c)$
$log_C := \langle \text{all prior messages} \rangle$ $tag_C := \mathsf{MAC}(ms, \text{“C”}, log_C)$	`ClientFinished`$[tag_C]$ ⟶	$log_C := \langle \text{all prior messages} \rangle$ $tag_C \stackrel{?}{=} \mathsf{MAC}(ms, \text{“C”}, log_C)$ $k := \mathsf{KDF}(ms, \ell)$
$log_S := \langle \text{all prior messages} \rangle$ $tag_S \stackrel{?}{=} \mathsf{MAC}(ms, \text{“S”}, log_S)$	⟵ `ServerFinished`$[tag_S]$	$log_S := \langle \text{all prior messages} \rangle$ $tag_S := \mathsf{MAC}(ms, \text{“S”}, log_S)$

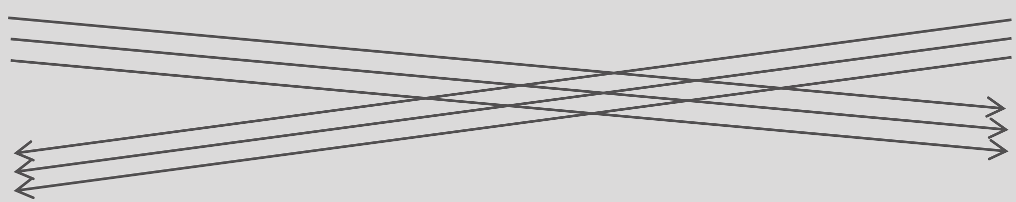
AppData



Cryptography of TLS 'as it is'

Client		Server
$\ell_C \leftarrow \$;\ a_C := cfg_C.a_C$	— `ClientHello`$[\ell_C, a_C, tag_C]$ ⟶	$\ell_S \leftarrow \$;\ \ell := \ell_C \,\Vert\, \ell_S;\ sid \leftarrow \$$ $cert_S := cfg_S.cert;\ cert_C := \bot$ $pk := \mathrm{pk}(cert_S)$ $sk :=$ lookup sk using pk $a, a_S := \mathrm{alg}_S(cfg_S, a_C);$
$\ell := \ell_C \,\Vert\, \ell_S;\ a := \mathrm{alg}_C(cfg_C, a_S)$ $pk := \mathrm{pk}(cert_S)$ $c, ms \leftarrow \mathrm{Enc}_e(p_E, pk, \ell)$ $k := \mathrm{KDF}(p_D, ms, \ell, r)$	← `ServerHello`$[\ell_S, a_S, sid, tag_{C,S}]$ `ServerCertificate`$[cert_S]$ `ServerHelloDone` —	
	— `ClientKeyExchange`$[c]$ ⟶	$ms \leftarrow \mathrm{Dec}_e(p_E, sk, \ell, c)$
$log_C :=$ ⟨all prior epoch messages⟩ $tag_C := \mathrm{MAC}(p_D, ms, \text{"C"}, log_C)$	— `ClientFinished`$[tag_C]$ ⟶	$log_C :=$ ⟨all prior epoch messages⟩ $tag_C \stackrel{?}{=} \mathrm{MAC}(p_D, ms, \text{"C"}, log_C)$ $k := \mathrm{KDF}(p_D, ms, \ell, r)$
$log_S :=$ ⟨all prior epoch messages⟩ $tag_S \stackrel{?}{=} \mathrm{MAC}(p_D, ms, \text{"S"}, log_S)$ $complete := 1$	⟵ `ServerFinished`$[tag_S]$ —	$log_S :=$ ⟨all prior epoch messages⟩ $tag_S := \mathrm{MAC}(p_D, ms, \text{"S"}, log_S)$ $complete := 1$; store (ℓ, sid, ms)

AppData



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, ,...)

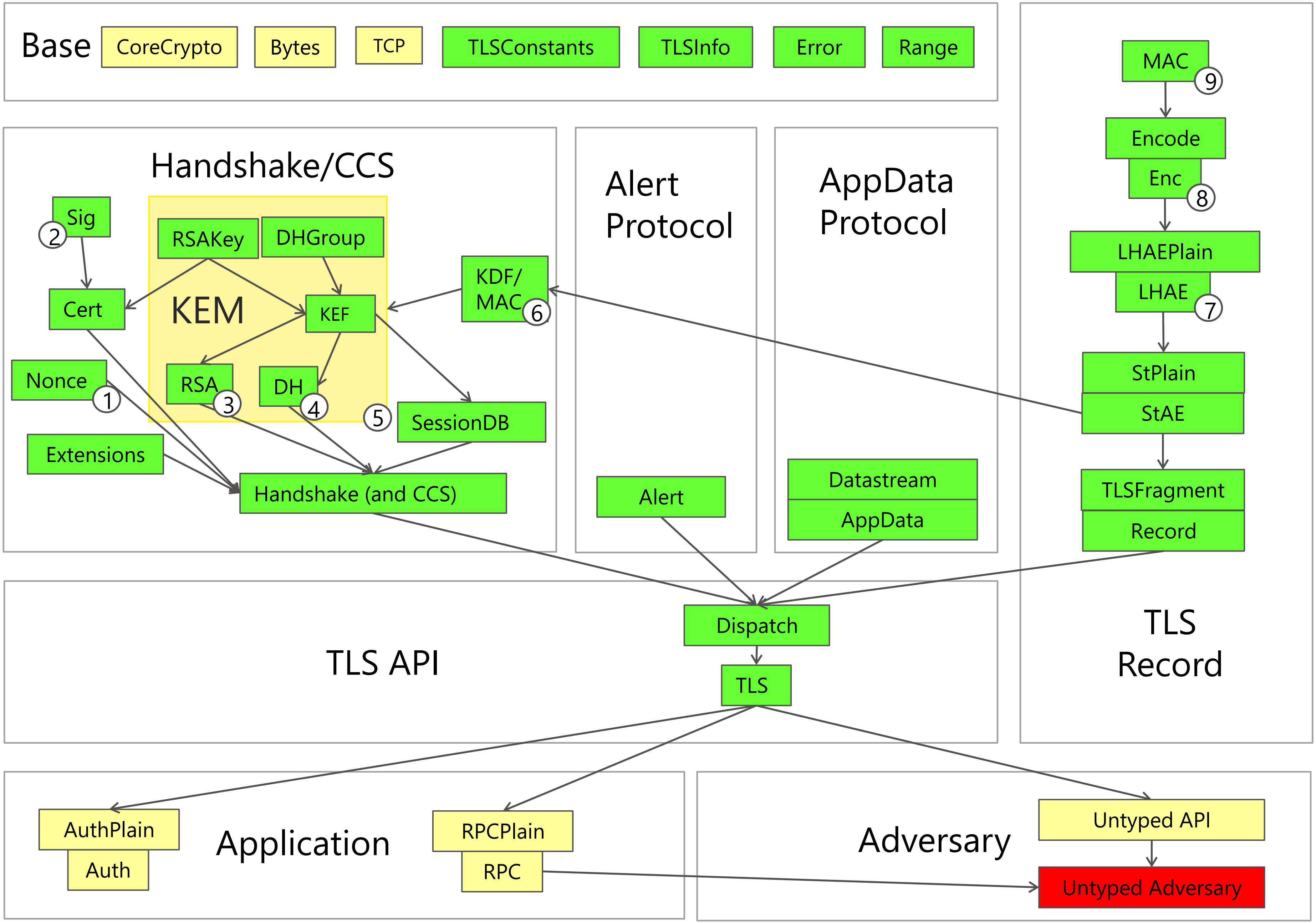
Authenticity: each endpoint only accepts a prefix of the data sequence send by its peer

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



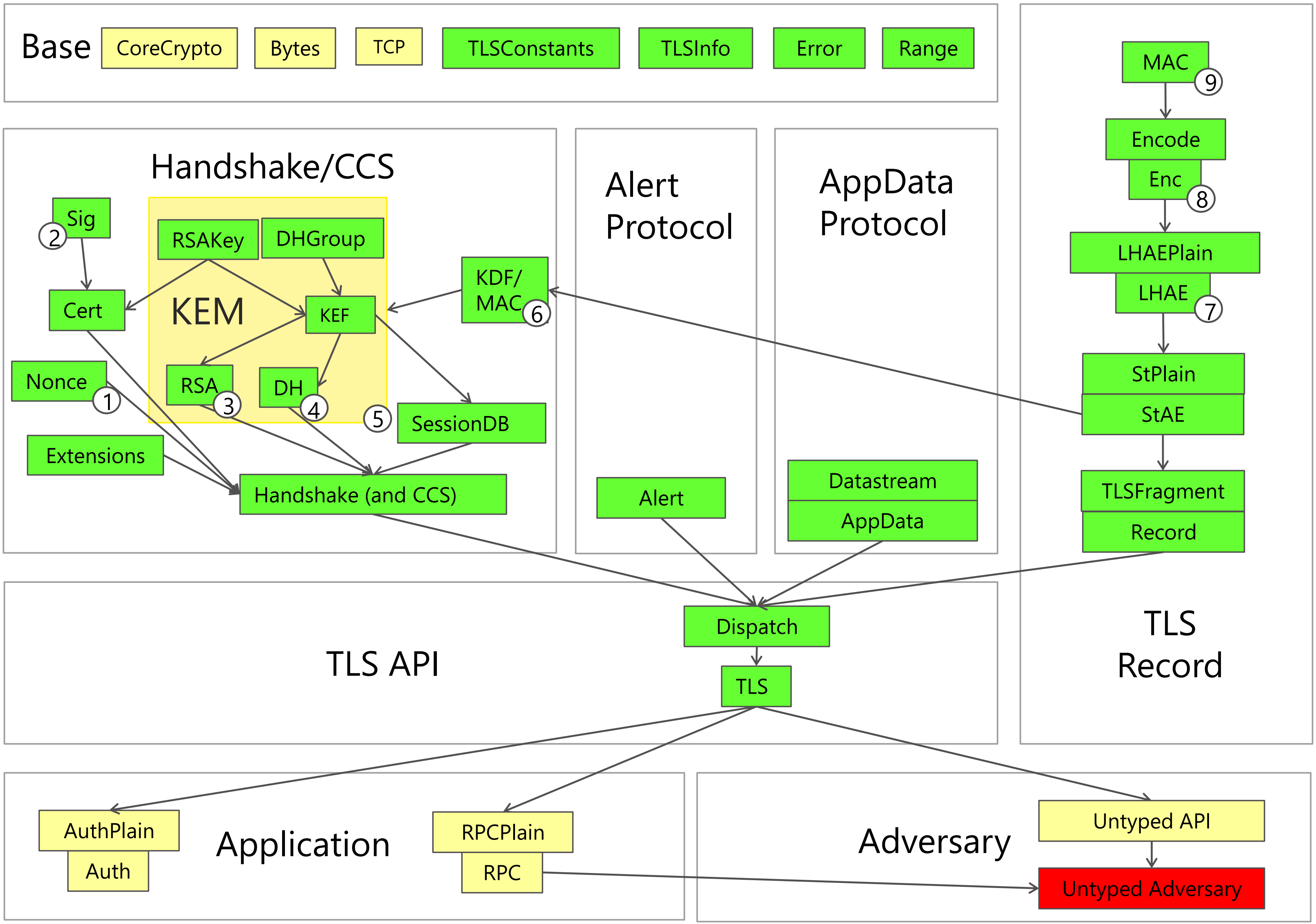
miTLS API & ideal functionality (outline)

Standard socket API with embedded security specification

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

```
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
type (;c:Connection) IOResult_i =
| Read      of c':Connection * data:(;c) AppData
| CertQuery of c':Connection
| Complete  of c':Connection { Agreement(c') }
| Close     of TcpStream
| Warning   of c':Connection * a:AlertDescription
| Fatal     of a:AlertDescription
val read : c:Connection -> (;c) IOResult_i

// writing data
type (;c:Connection,data:(;c) AppData) IOResult_o =
| WriteComplete of c':Connection
| WritePartial  of c':Connection * rest:(;c') AppData
| MustRead      of c':Connection
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
```

The concrete implementation



Security of master secret KEM

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

Client		**Server**
$pk := \mathrm{pk}(certs)$		
$c, ms \leftarrow \mathrm{Enc}(\underline{p^*}, pk, \ell)$	— `ClientKeyExchange[c]` ⟶	$ms' \leftarrow \mathrm{Dec}(\underline{p \in P}, sk, \ell, c)$
$k := \mathrm{D.KDF}(p_{\mathrm{D}}, ms, \ell, r)$		

For $p^\star$ an agility parameter, P a set of parameters $\mathbf{Adv}^{\mathrm{RCCA}}_{p^\star,P}(\mathcal{A}) \stackrel{\mathrm{def}}{=} 2\Pr[\mathrm{RCCA}:1] - 1$

Game RCCA $\stackrel{\mathrm{def}}{=}$
- $pk, sk \leftarrow \mathrm{KeyGen}()$
- $K, L := \emptyset$
- $b \leftarrow \{0,1\}$
- $b' \leftarrow \mathcal{A}^{\mathrm{ENC,DEC}}(pk)$
- **return** $(b' = b)$

Oracle $\mathrm{ENC}(\ell) \stackrel{\mathrm{def}}{=}$
- **if** $\ell \in L$ **then return** $\perp$
- $k_0, c \leftarrow \mathrm{Enc}(p^\star, pk, \ell)$
- $k_1 \leftarrow \$$
- $K(\ell) := K(\ell) \cup \{k_0, k_1\}$
- **return** k_b, c

Oracle $\mathrm{DEC}(p, \ell, c) \stackrel{\mathrm{def}}{=}$
- **if** $\ell \in L \vee p \notin P$ **then return** $\perp$
- $L := L \cup \{\ell\}$
- $k \leftarrow \mathrm{Dec}(p, sk, \ell, c)$
- **if** $k \in K(\ell)$ **then return** $\perp$
- **return** k

Security of master secret KEM

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

Client		Server
$pk := \mathrm{pk}(certs_S)$		
$c, ms \leftarrow \mathrm{Enc}(p^*, pk, \ell)$	— `ClientKeyExchange[c]` ⟶	$ms' \leftarrow \mathrm{Dec}(p \in P, sk, \ell, c)$
$k := \mathrm{D.KDF}(p_D, ms, \ell, r)$		

For $p^\star$ an agility parameter, P a set of parameters $\mathbf{Adv}^{\mathrm{RCCA}}_{p^\star, P}(\mathcal{A}) \stackrel{\mathrm{def}}{=} 2\Pr[\mathrm{RCCA} : 1] - 1$

Game RCCA $\stackrel{\mathrm{def}}{=}$
- $pk, sk \leftarrow \mathrm{KeyGen}()$
- $K, L := \emptyset$
- $b \leftarrow \{0, 1\}$
- $b' \leftarrow \mathcal{A}^{\mathrm{ENC,DEC}}(pk)$
- **return** $(b' = b)$

Oracle $\mathrm{ENC}(\ell) \stackrel{\mathrm{def}}{=}$
- **if** $\ell \in L$ **then return** $\perp$
- $k_0, c \leftarrow \mathrm{Enc}(p^\star, pk, \ell)$
- $k_1 \leftarrow \$$
- $K(\ell) := K(\ell) \cup \{k_0, k_1\}$
- **return** k_b, c

Oracle $\mathrm{DEC}(p, \ell, c) \stackrel{\mathrm{def}}{=}$
- **if** $\ell \in L \vee p \notin P$ **then return** $\perp$
- $L := L \cup \{\ell\}$
- $k \leftarrow \mathrm{Dec}(p, sk, \ell, c)$
- **if** $k \in K(\ell)$ **then return** $\perp$
- **return** k

Sufficient assumptions on the pms-KEM

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

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

(One-Wayness) **Game** OW-PCA $\stackrel{\text{def}}{=}$
$pk, sk \leftarrow \text{keygen}()$
$k^\star, c^\star \leftarrow \text{enc}(p^\star, pk)$
$k \leftarrow \mathcal{A}^{\text{PCO}}(pk, c^\star)$
return $(k = k^\star)$

Game NR-PCA $\stackrel{\text{def}}{=}$ (Non-Randomizability)
$pk, sk \leftarrow \text{keygen}()$
$k^\star, c^\star \leftarrow \text{enc}(p^\star, pk)$
$c \leftarrow \mathcal{A}^{\text{PCO}}(pk, c^\star)$
return $c \neq c^\star \wedge$
$k^\star = \text{dec}(p^\star, sk, c)$

Sufficient assumptions on the pms-KEM

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

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

(One-Wayness)
Game $\text{OW-PCA} \stackrel{\text{def}}{=}$
$pk, sk \leftarrow \text{keygen}()$
$k^\star, c^\star \leftarrow \text{enc}(p^\star, pk)$
$k \leftarrow \mathcal{A}^{\text{PCO}}(pk, c^\star)$
return $(k = k^\star)$

(Non-Randomizability)
Game $\text{NR-PCA} \stackrel{\text{def}}{=}$
$pk, sk \leftarrow \text{keygen}()$
$k^\star, c^\star \leftarrow \text{enc}(p^\star, pk)$
$c \leftarrow \mathcal{A}^{\text{PCO}}(pk, c^\star)$
return $c \neq c^\star \wedge$
$k^\star = \text{dec}(p^\star, sk, c)$

Sufficient assumptions on the pms-KEM

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

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

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

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

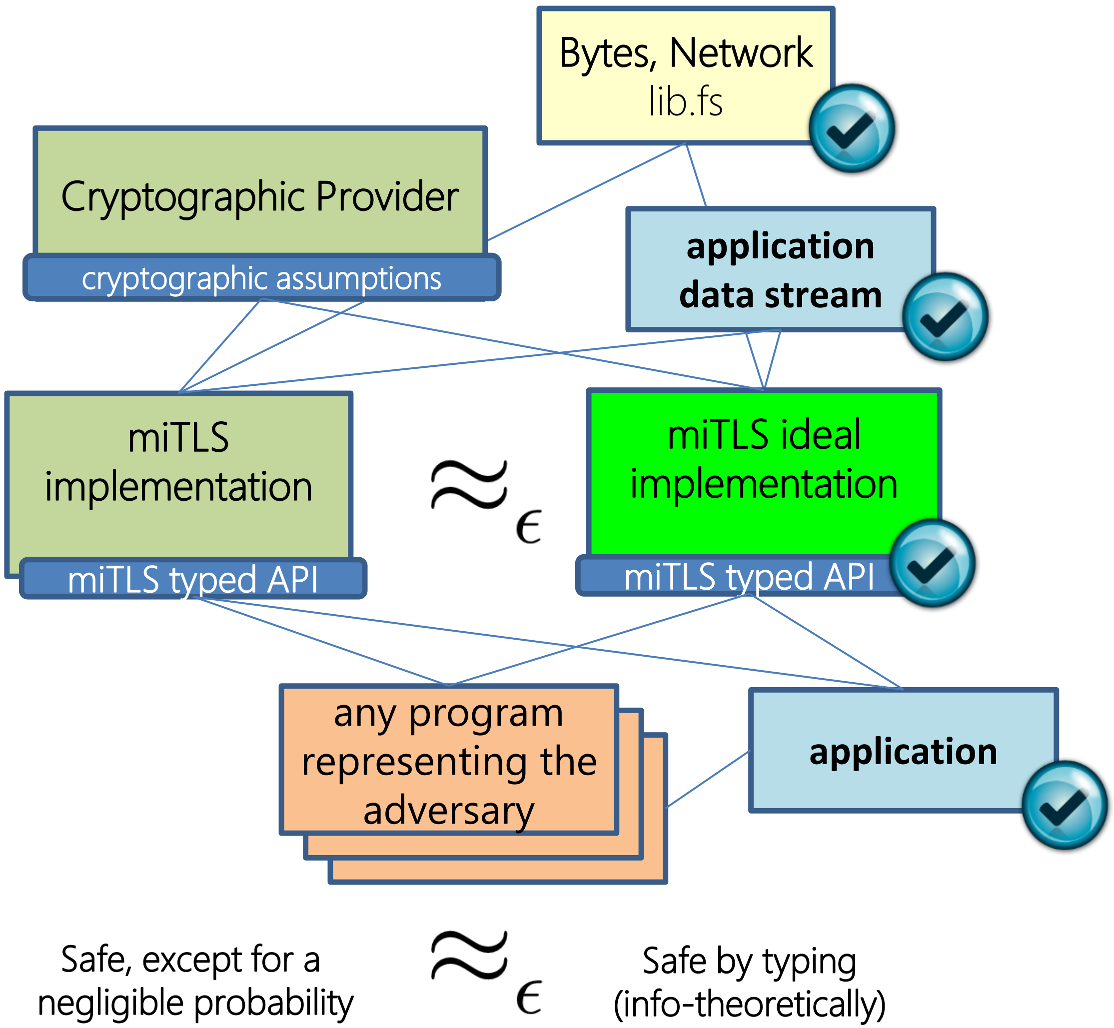
where P' includes all pv such that $(pv, h) \in P$

Proof formalized and checked in EasyCrypt (3,000 lines)

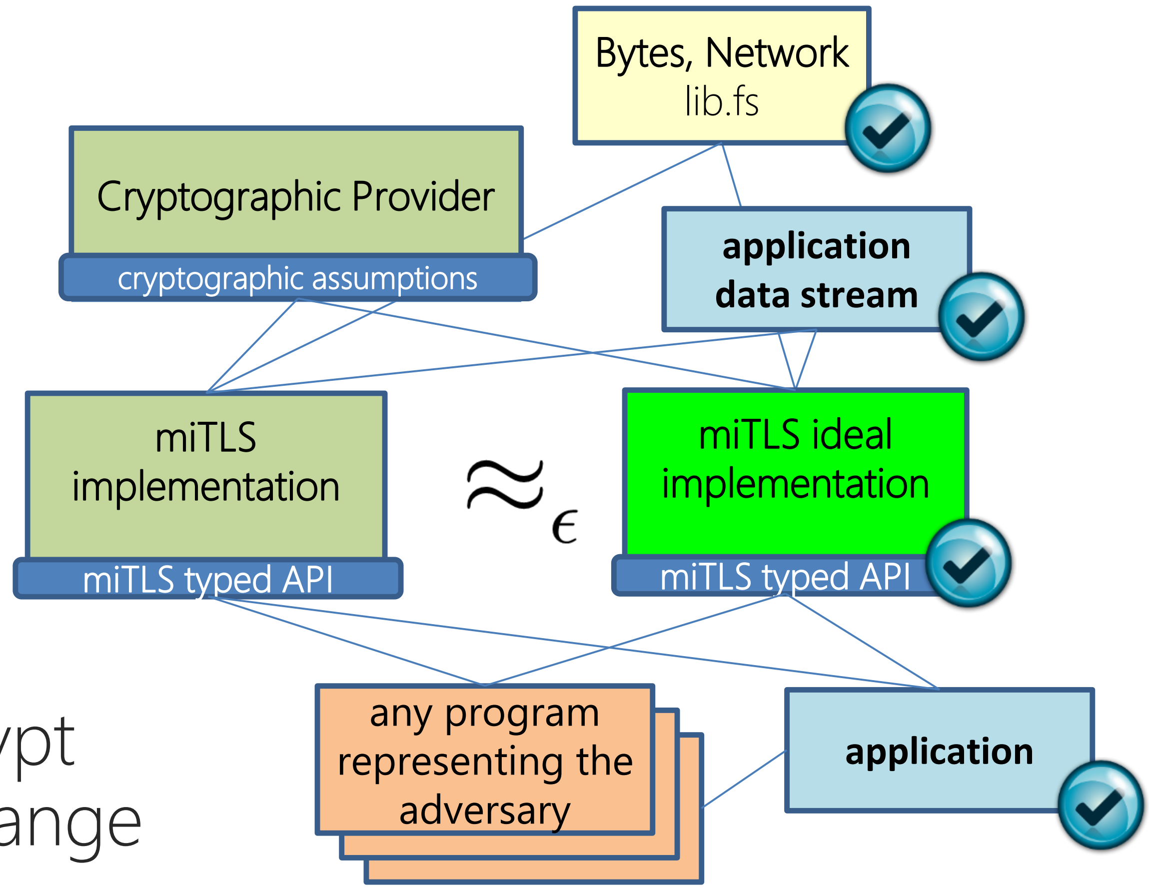
Security theorem

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)



Security theorem



Proof automation

7,000 lines of F#
checked against
3,000 lines of F7
type annotations

+

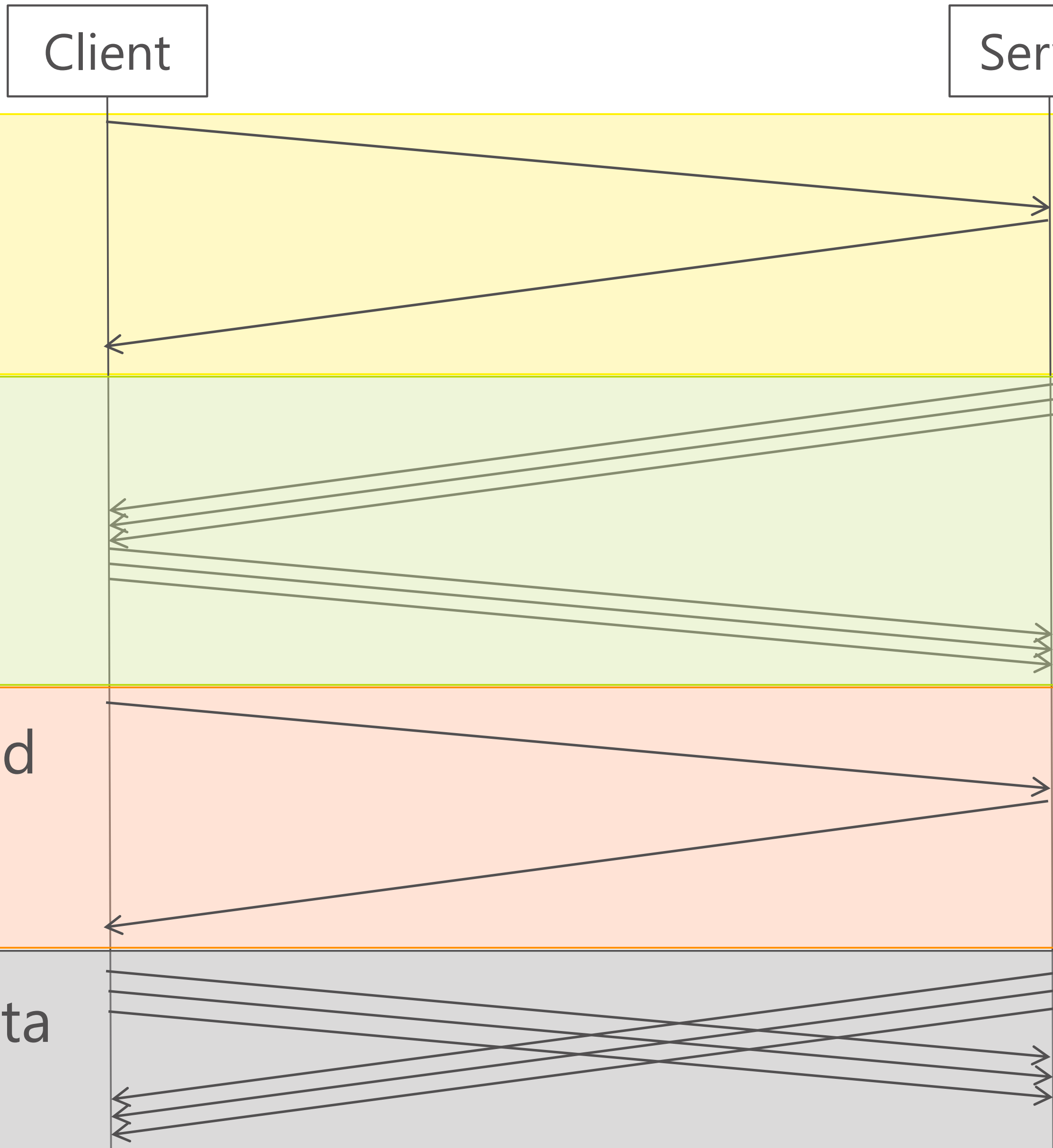
3,000 lines of EasyCrypt
for the core key exchange

Ongoing work

ECDHE, GCM, Certificates, Side-channels

Mission accomplished?

Reusing established sessions



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

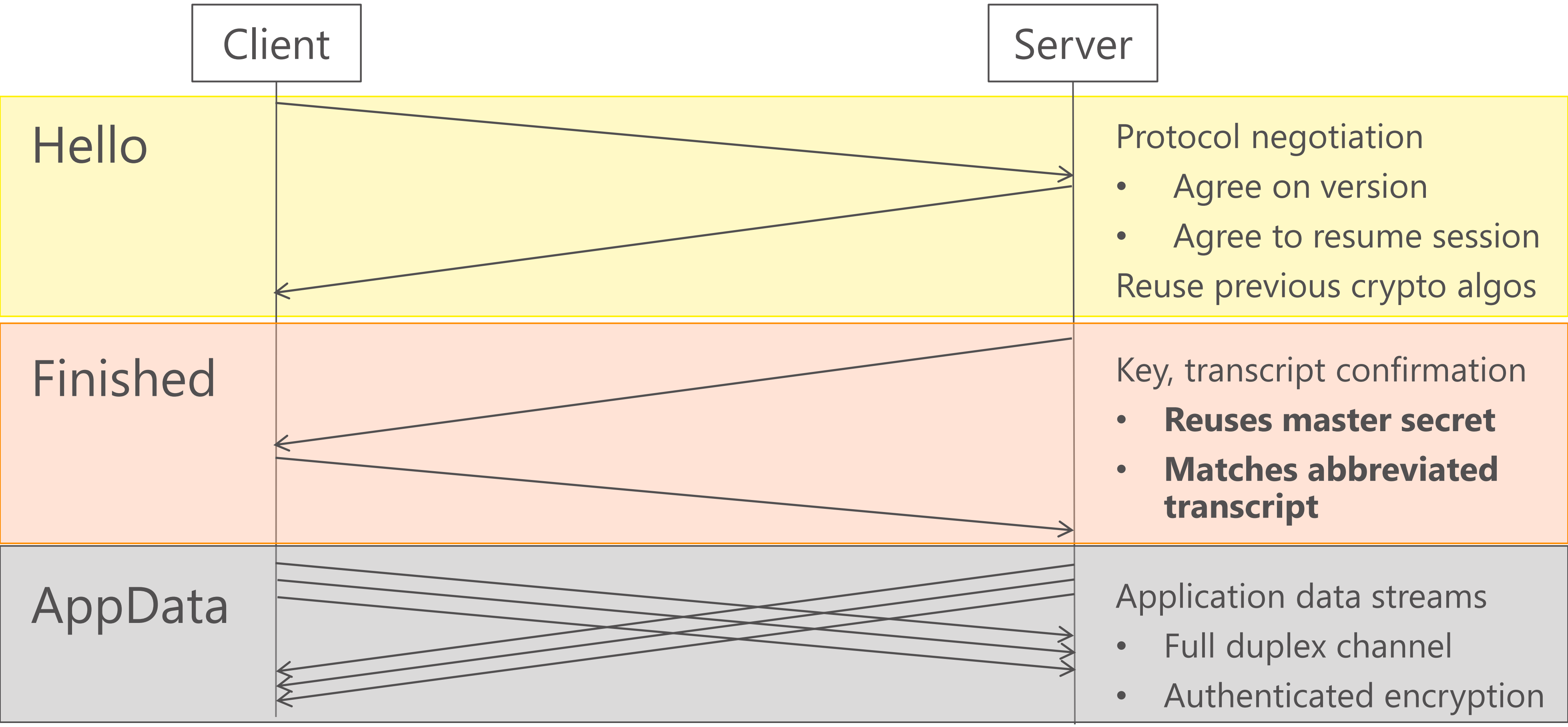
- Completes authentication
- Matches transcripts
- Authenticated encryption

AppData

Application data streams

- Full duplex channel
- Authenticated encryption

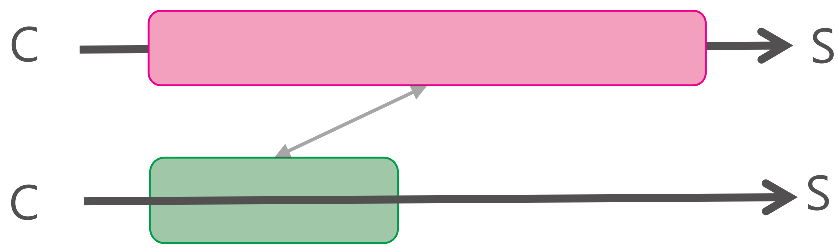
Abbreviated handshake



Efficiency: One round-trip before client sends data

Security?

Security of session resumption



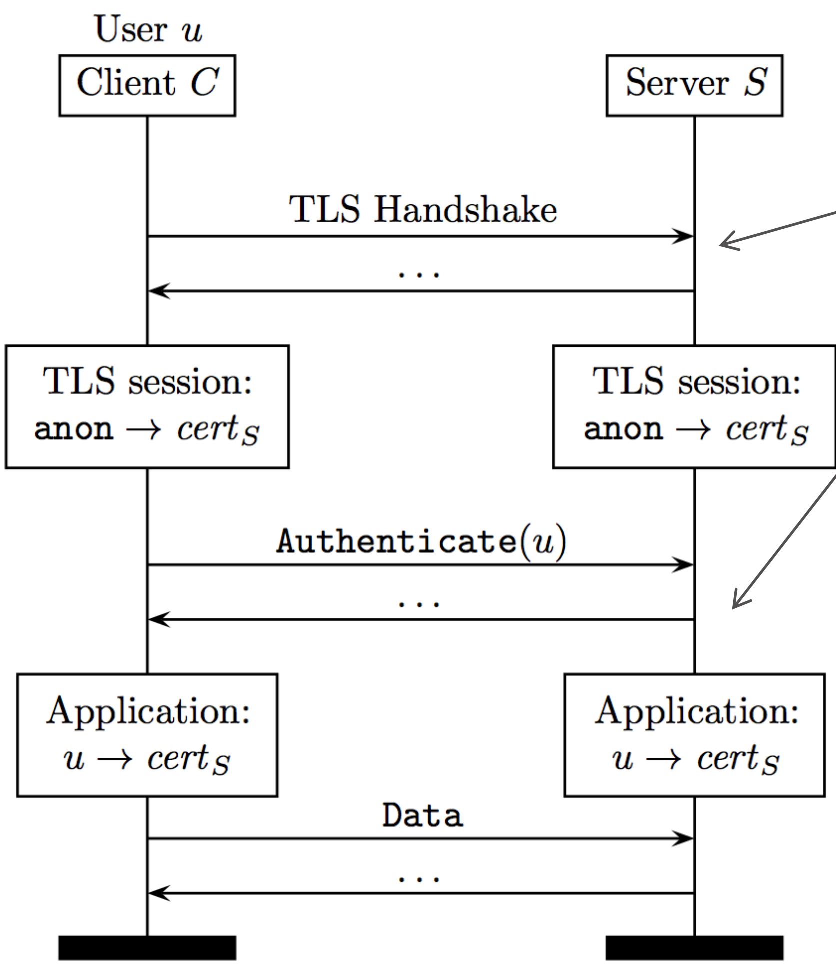
If a client completes an abbreviated handshake and the server in the original handshake was honest, and the master secret has not been leaked, then

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

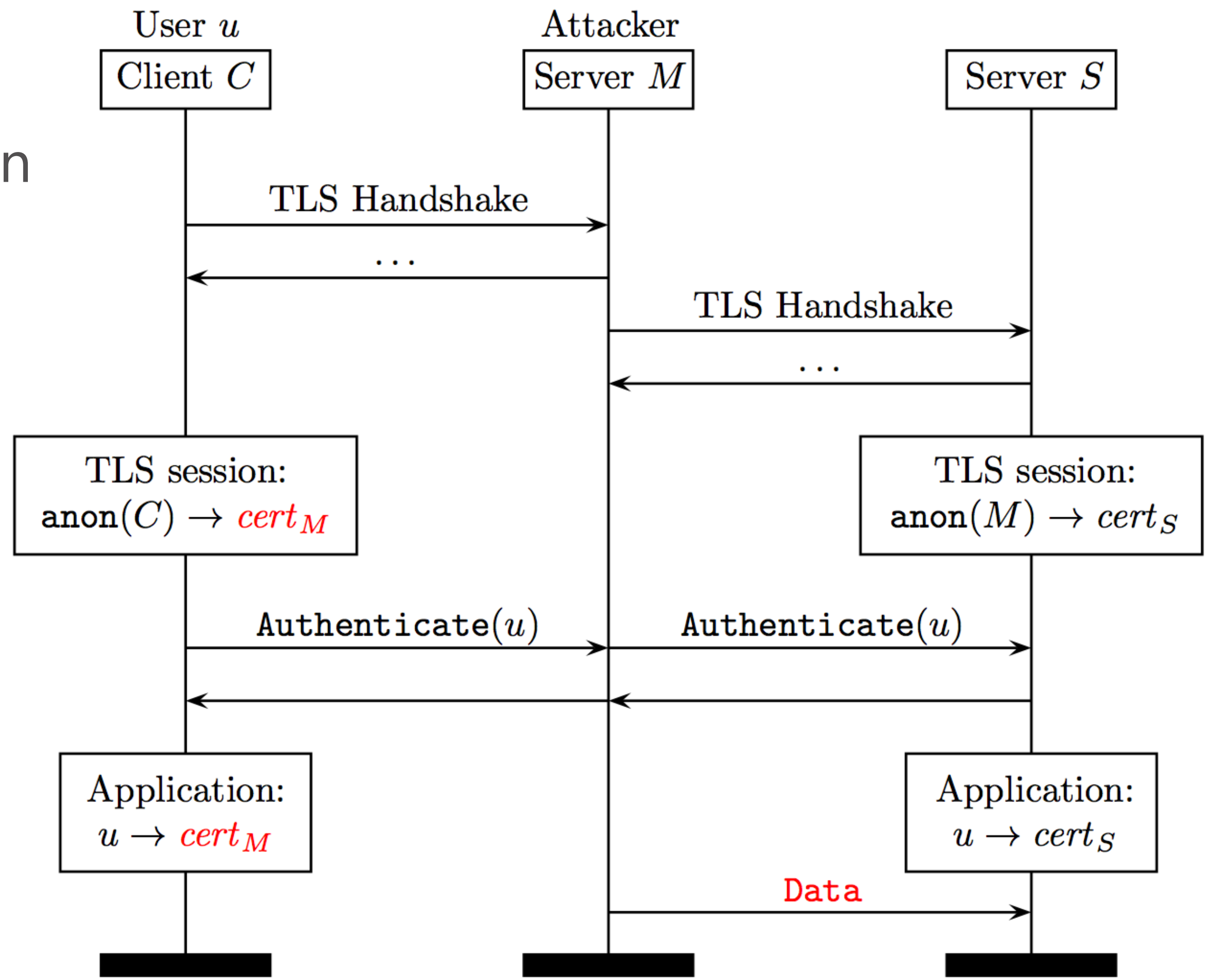
  Need to strongly bind the two protocol layers together!

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*!

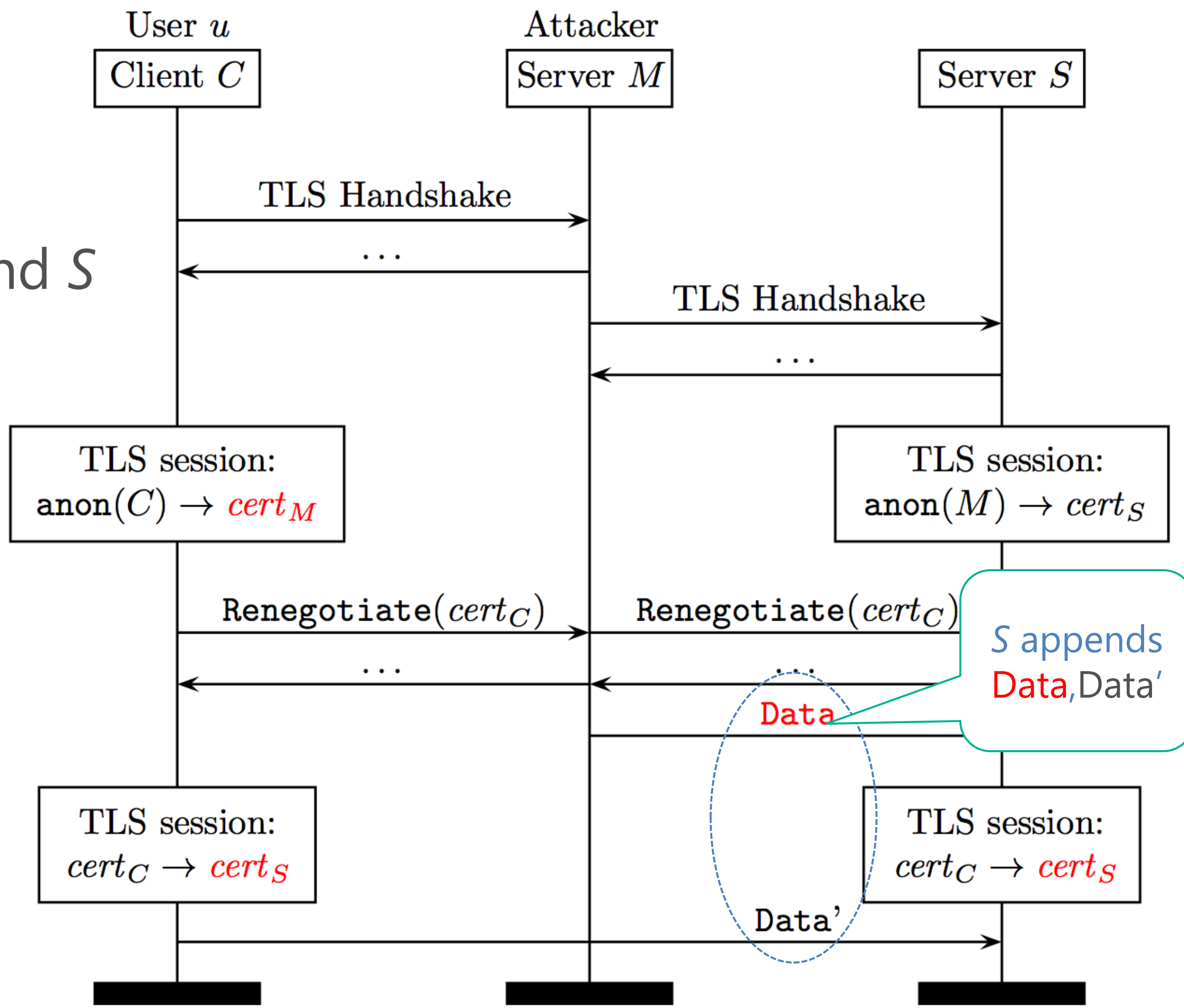


TLS renegotiation attack [2009]

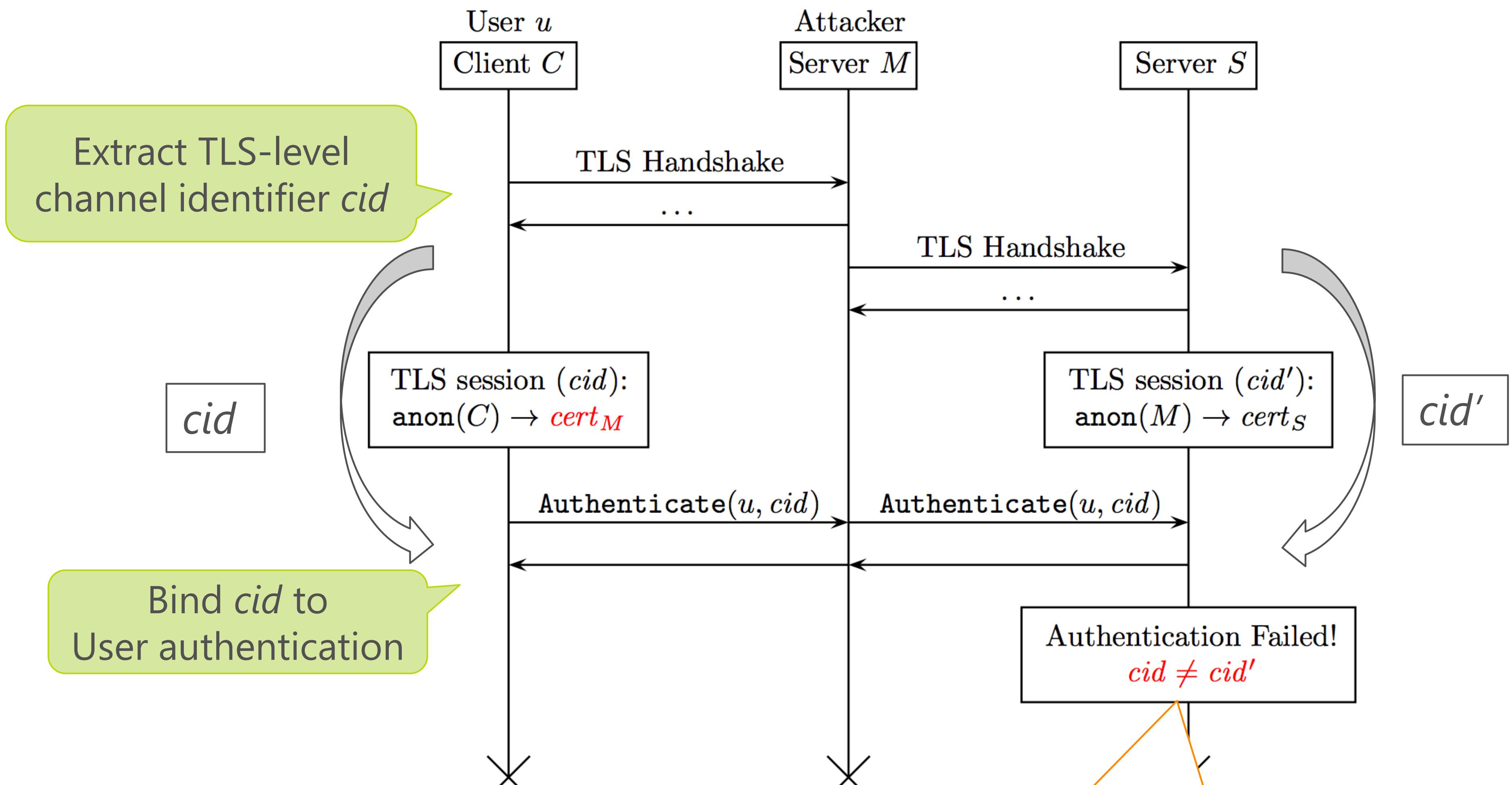
Martin Rex's Version

- 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



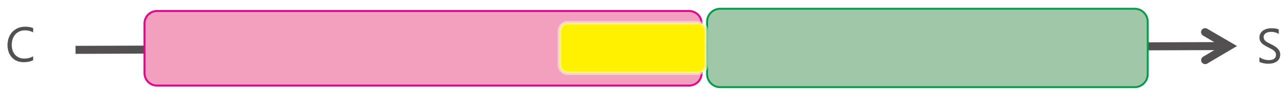
Computing a channel identifier (*cid*):

- *f(master secret)* (EAP)
- *f(handshake log)* (Renegotiation Indication,SASL)

Does not work if *M* can ensure that cid = cid’

Security of (fixed) renegotiation

[Giesen et al '13]



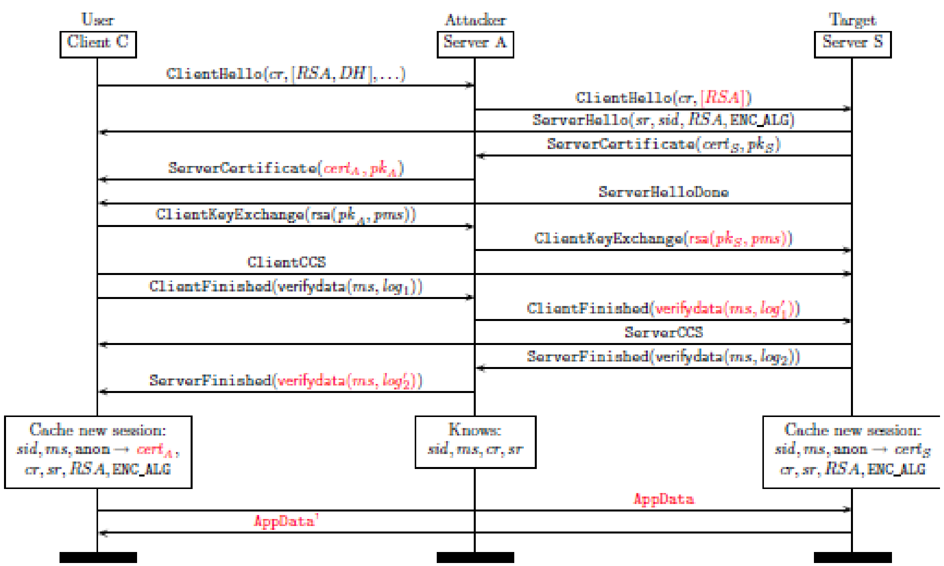
If an endpoint completes renegotiation with an honest peer and (all) strong algorithms, then

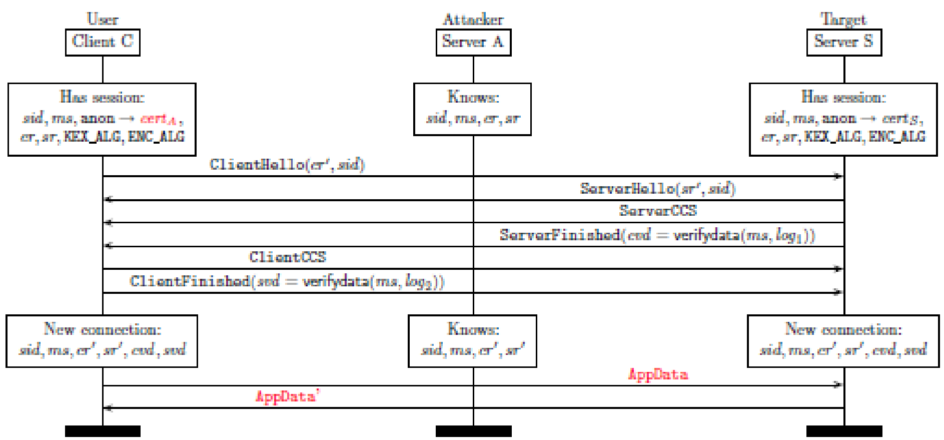
Agreement: there must be a peer endpoint that agrees on all variables in the new and the old handshake *(even if the peer in the old handshake was compromised or unauthenticated)*

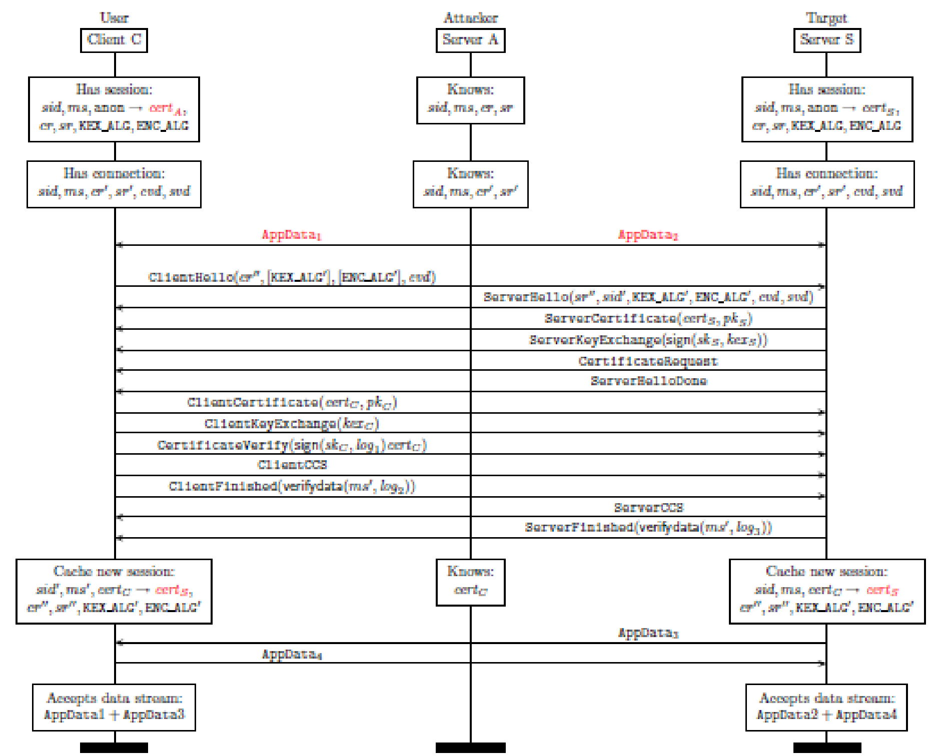
More generally, in a sequence of handshakes, the last handshake guarantees agreement on all previous ones

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

Triple Handshake attack: step 1

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

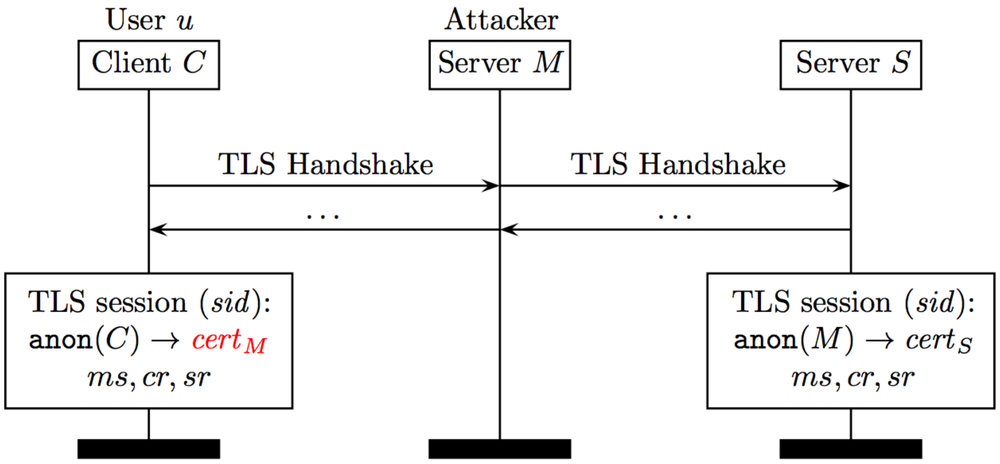
RSA Key Synchronization

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.

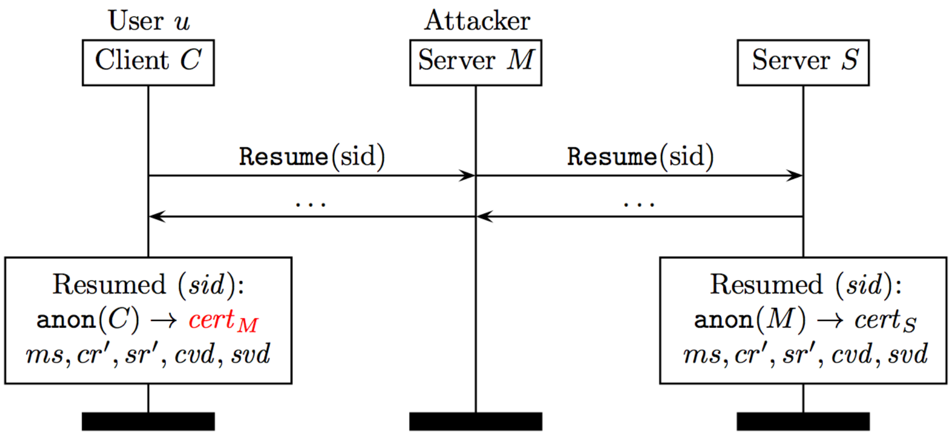
Triple Handshake attack: step 2

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

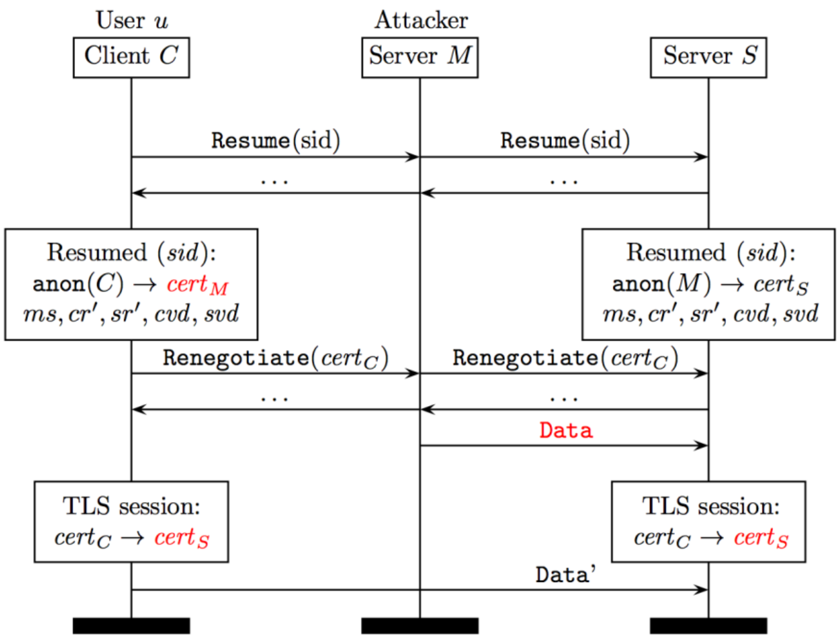
After resumption, handshake log is not a good channel identifier

Breaks tls-unique (SASL), renegotiation indication

Triple Handshake attack: step 3

User Impersonation Attack (reenables 2009 attack)

cid = hash(abbreviated handshake log) 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