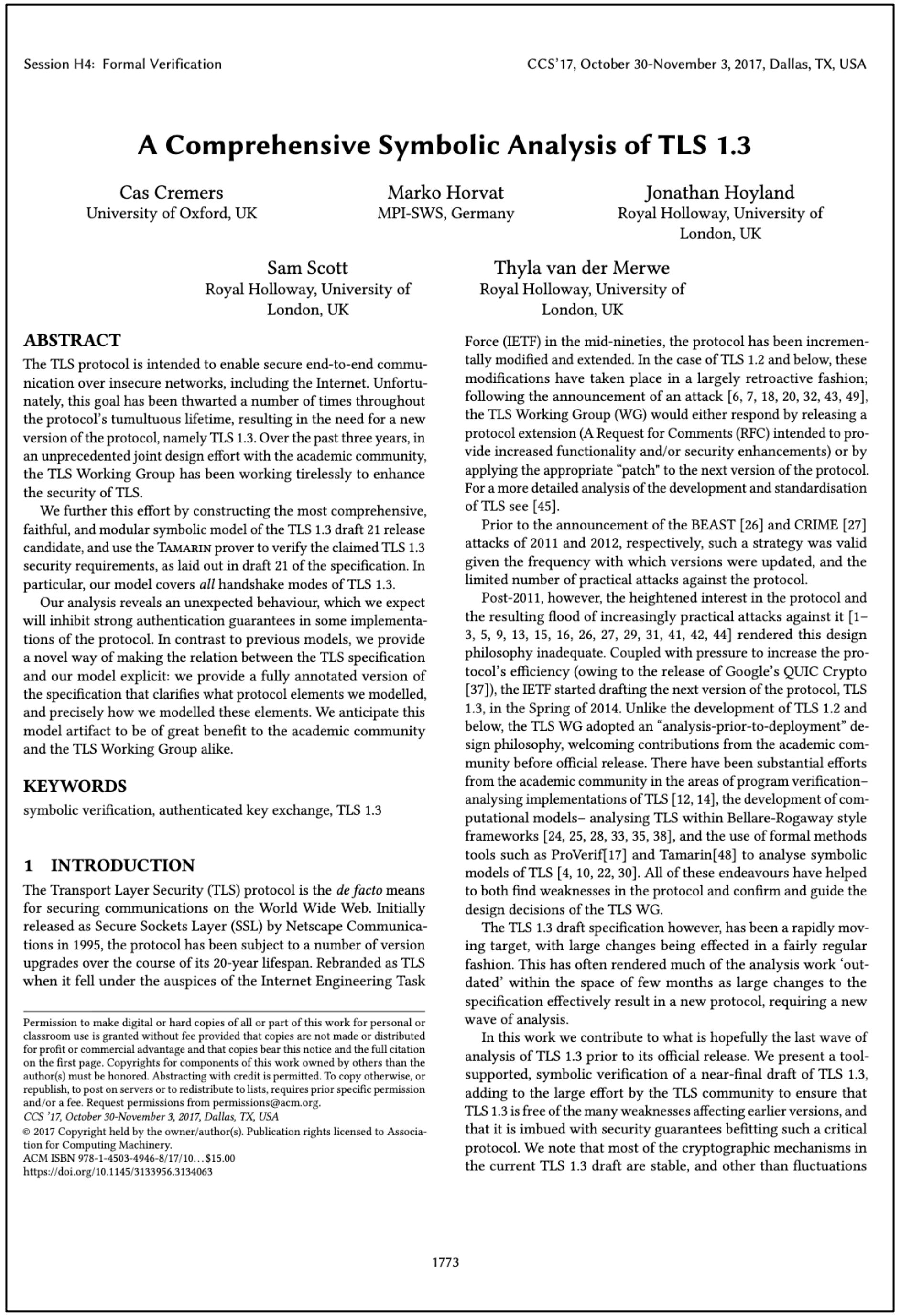
Proving KEMTLS in Tamarin or: I used Tamarin and you can too!

Douglas Stebila

UNIVERSITY OF WATERLOO

Based on joint work with Sofía Celi, Jonathan Hoyland, Thom Wiggers

SKECH 4 • 2022-07-12



A Comprehensive Symbolic Analysis of TLS 1.3

Cas Cremers
University of Oxford, UK

Marko Horvat
MPI-SWS, Germany

Jonathan Hoyland
Royal Holloway, University of London, UK

Sam Scott
Royal Holloway, University of London, UK

Thyla van der Merwe
Royal Holloway, University of London, UK

ABSTRACT

The TLS protocol is intended to enable secure end-to-end communication over insecure networks, including the Internet. Unfortunately, this goal has been thwarted a number of times throughout the protocol's tumultuous lifetime, resulting in the need for a new version of the protocol, namely TLS 1.3. Over the past three years, in an unprecedented joint design effort with the academic community, the TLS Working Group has been working tirelessly to enhance the security of TLS.

We further this effort by constructing the most comprehensive, faithful, and modular symbolic model of the TLS 1.3 draft 21 release candidate, and use the Tamarin prover to verify the claimed TLS 1.3 security requirements, as laid out in draft 21 of the specification. In particular, our model covers *all* handshake modes of TLS 1.3.

Our analysis reveals an unexpected behaviour, which we expect will inhibit strong authentication guarantees in some implementations of the protocol. In contrast to previous models, we provide a novel way of making the relation between the TLS specification and our model explicit: we provide a fully annotated version of the specification that clarifies what protocol elements we modelled, and precisely how we modelled these elements. We anticipate this model artifact to be of great benefit to the academic community and the TLS Working Group alike.

KEYWORDS

symbolic verification, authenticated key exchange, TLS 1.3

1 INTRODUCTION

The Transport Layer Security (TLS) protocol is the *de facto* means for securing communications on the World Wide Web. Initially released as Secure Sockets Layer (SSL) by Netscape Communications in 1995, the protocol has been subject to a number of version upgrades over the course of its 20-year lifespan. Rebranded as TLS when it fell under the auspices of the Internet Engineering Task Force (IETF) in the mid-nineties, the protocol has been incrementally modified and extended. In the case of TLS 1.2 and below, these modifications have taken place in a largely retroactive fashion; following the announcement of an attack [6, 7, 18, 20, 32, 43, 49], the TLS Working Group (WG) would either respond by releasing a protocol extension (A Request for Comments (RFC) intended to provide increased functionality and/or security enhancements) or by applying the appropriate "patch" to the next version of the protocol. For a more detailed analysis of the development and standardisation of TLS see [45].

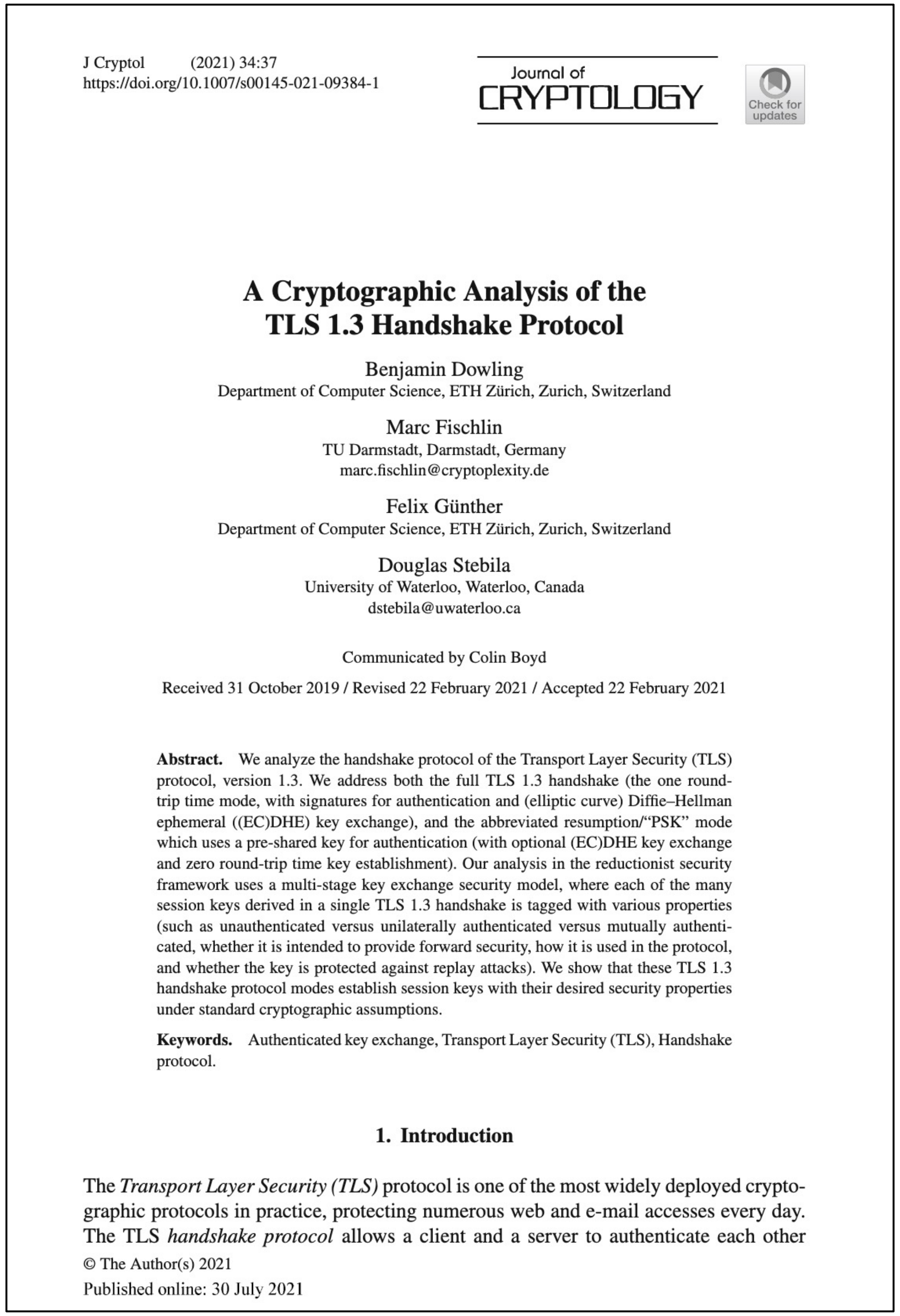
Prior to the announcement of the BEAST [26] and CRIME [27] attacks of 2011 and 2012, respectively, such a strategy was valid given the frequency with which versions were updated, and the limited number of practical attacks against the protocol.

Post-2011, however, the heightened interest in the protocol and the resulting flood of increasingly practical attacks against it [1–3, 5, 9, 13, 15, 16, 26, 27, 29, 31, 41, 42, 44] rendered this design philosophy inadequate. Coupled with pressure to increase the protocol's efficiency (owing to the release of Google's QUIC Crypto [37]), the IETF started drafting the next version of the protocol, TLS 1.3, in the Spring of 2014. Unlike the development of TLS 1.2 and below, the TLS WG adopted an "analysis-prior-to-deployment" design philosophy, welcoming contributions from the academic community before official release. There have been substantial efforts from the academic community in the areas of program verification–analysing implementations of TLS [12, 14], the development of computational models– analysing TLS within Bellare-Rogaway style frameworks [24, 25, 28, 33, 35, 38], and the use of formal methods tools such as ProVerif[17] and Tamarin[48] to analyse symbolic models of TLS [4, 10, 22, 30]. All of these endeavours have helped to both find weaknesses in the protocol and confirm and guide the design decisions of the TLS WG.

The TLS 1.3 draft specification however, has been a rapidly moving target, with large changes being effected in a fairly regular fashion. This has often rendered much of the analysis work 'outdated' within the space of few months as large changes to the specification effectively result in a new protocol, requiring a new wave of analysis.

In this work we contribute to what is hopefully the last wave of analysis of TLS 1.3 prior to its official release. We present a tool-supported, symbolic verification of a near-final draft of TLS 1.3, adding to the large effort by the TLS community to ensure that TLS 1.3 is free of the many weaknesses affecting earlier versions, and that it is imbued with security guarantees befitting such a critical protocol. We note that most of the cryptographic mechanisms in the current TLS 1.3 draft are stable, and other than fluctuations

ACM ISBN 978-1-4503-4946-8/17/10…$15.00
https://doi.org/10.1145/3133956.3134063



J Cryptol (2021) 34:37
https://doi.org/10.1007/s00145-021-09384-1

Journal of CRYPTOLOGY

A Cryptographic Analysis of the TLS 1.3 Handshake Protocol

Benjamin Dowling
Department of Computer Science, ETH Zürich, Zurich, Switzerland

Marc Fischlin
TU Darmstadt, Darmstadt, Germany
marc.fischlin@cryptoplexity.de

Felix Günther
Department of Computer Science, ETH Zürich, Zurich, Switzerland

Douglas Stebila
University of Waterloo, Waterloo, Canada
dstebila@uwaterloo.ca

Communicated by Colin Boyd

Received 31 October 2019 / Revised 22 February 2021 / Accepted 22 February 2021

Abstract. We analyze the handshake protocol of the Transport Layer Security (TLS) protocol, version 1.3. We address both the full TLS 1.3 handshake (the one round-trip time mode, with signatures for authentication and (elliptic curve) Diffie–Hellman ephemeral ((EC)DHE) key exchange), and the abbreviated resumption/"PSK" mode which uses a pre-shared key for authentication (with optional (EC)DHE key exchange and zero round-trip time key establishment). Our analysis in the reductionist security framework uses a multi-stage key exchange security model, where each of the many session keys derived in a single TLS 1.3 handshake is tagged with various properties (such as unauthenticated versus unilaterally authenticated versus mutually authenticated, whether it is intended to provide forward security, how it is used in the protocol, and whether the key is protected against replay attacks). We show that these TLS 1.3 handshake protocol modes establish session keys with their desired security properties under standard cryptographic assumptions.

Keywords. Authenticated key exchange, Transport Layer Security (TLS), Handshake protocol.

1. Introduction

The *Transport Layer Security (TLS)* protocol is one of the most widely deployed cryptographic protocols in practice, protecting numerous web and e-mail accesses every day. The TLS *handshake protocol* allows a client and a server to authenticate each other

Published online: 30 July 2021

How do we compare them?

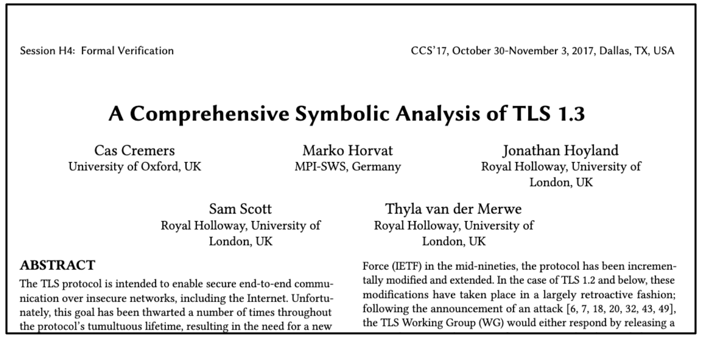
- Are they about the same protocol?

- Are they about the same security properties?
  - Adversary interaction
  - Adversary goals

- Are they using the same assumptions?

Are they about the same protocol?

Both are about the TLS 1.3 handshake, but as interpreted and abstracted by the authors



Session H4: Formal Verification CCS'17, October 30-November 3, 2017, Dallas, TX, USA

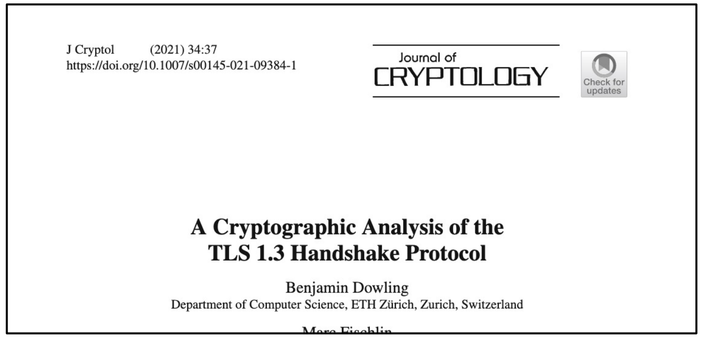
A Comprehensive Symbolic Analysis of TLS 1.3

Cas Cremers, University of Oxford, UK
Marko Horvat, MPI-SWS, Germany
Jonathan Hoyland, Royal Holloway, University of London, UK
Sam Scott, Royal Holloway, University of London, UK
Thyla van der Merwe, Royal Holloway, University of London, UK

ABSTRACT
The TLS protocol is intended to enable secure end-to-end communication over insecure networks, including the Internet. Unfortunately, this goal has been thwarted a number of times throughout the protocol's tumultuous lifetime, resulting in the need for a new

Force (IETF) in the mid-nineties, the protocol has been incrementally modified and extended. In the case of TLS 1.2 and below, these modifications have taken place in a largely retroactive fashion; following the announcement of an attack [6, 7, 18, 20, 32, 43, 49], the TLS Working Group (WG) would either respond by releasing a

- Closer to wire format – includes most fields, extensions
- Includes multiple modes in same analysis



J Cryptol (2021) 34:37
https://doi.org/10.1007/s00145-021-09384-1

Journal of CRYPTOLOGY

Check for updates

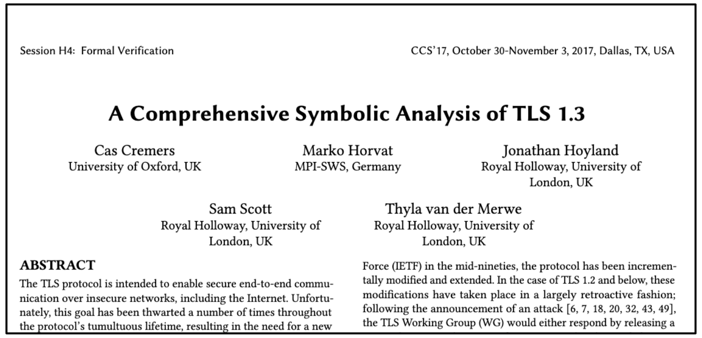
A Cryptographic Analysis of the TLS 1.3 Handshake Protocol

Benjamin Dowling
Department of Computer Science, ETH Zürich, Zurich, Switzerland

- Cryptographic core of TLS 1.3 handshake
- Multiple modes handled separately

Are they about the same security goals?

Both cover session key security, but starting from different places



Session H4: Formal Verification CCS'17, October 30-November 3, 2017, Dallas, TX, USA

A Comprehensive Symbolic Analysis of TLS 1.3

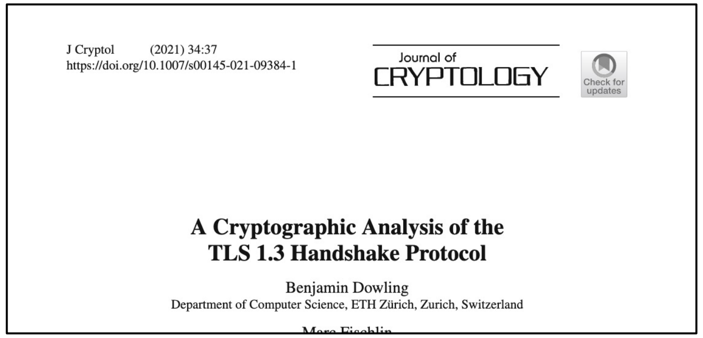
Cas Cremers, University of Oxford, UK
Marko Horvat, MPI-SWS, Germany
Jonathan Hoyland, Royal Holloway, University of London, UK
Sam Scott, Royal Holloway, University of London, UK
Thyla van der Merwe, Royal Holloway, University of London, UK

ABSTRACT

The TLS protocol is intended to enable secure end-to-end communication over insecure networks, including the Internet. Unfortunately, this goal has been thwarted a number of times throughout the protocol's tumultuous lifetime, resulting in the need for a new

Force (IETF) in the mid-nineties, the protocol has been incrementally modified and extended. In the case of TLS 1.2 and below, these modifications have taken place in a largely retroactive fashion; following the announcement of an attack [6, 7, 18, 20, 32, 43, 49], the TLS Working Group (WG) would either respond by releasing a

- Covers 6/8 security goals from TLS 1.3 specification
  - Session key secrecy with forward secrecy
  - Authentication
  - Agreement
  - …



J Cryptol (2021) 34:37
https://doi.org/10.1007/s00145-021-09384-1

Journal of CRYPTOLOGY

Check for updates

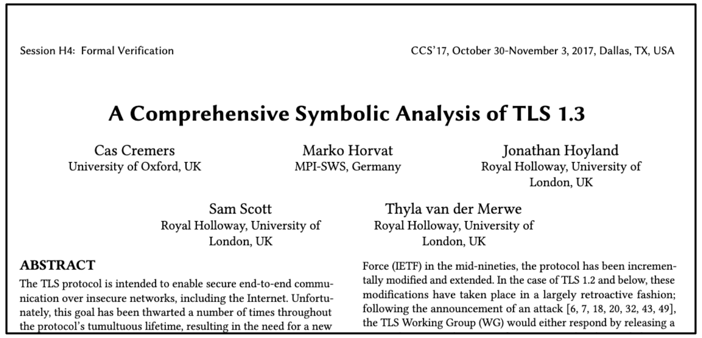
A Cryptographic Analysis of the TLS 1.3 Handshake Protocol

Benjamin Dowling
Department of Computer Science, ETH Zürich, Zurich, Switzerland

- Multi-stage AKE definition
  - Builds on long-standing AKE models from [BR93] onwards
  - Session key indistinguishability with forward secrecy
  - Match security

Are they using the same assumptions?

Both assume secure building blocks (signature, DH, KDF, …), but with very different modelling



Session H4: Formal Verification CCS'17, October 30-November 3, 2017, Dallas, TX, USA

A Comprehensive Symbolic Analysis of TLS 1.3

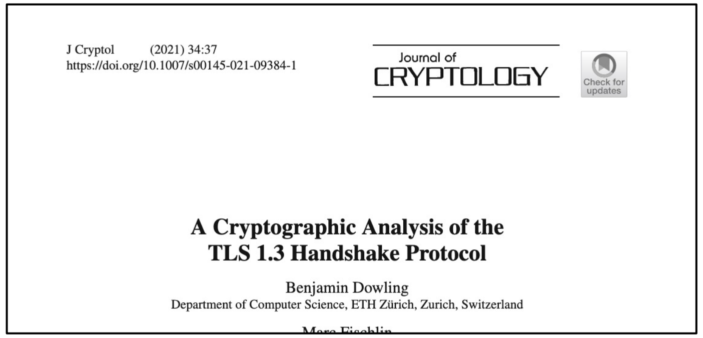
Cas Cremers, University of Oxford, UK
Marko Horvat, MPI-SWS, Germany
Jonathan Hoyland, Royal Holloway, University of London, UK
Sam Scott, Royal Holloway, University of London, UK
Thyla van der Merwe, Royal Holloway, University of London, UK

ABSTRACT

The TLS protocol is intended to enable secure end-to-end communication over insecure networks, including the Internet. Unfortunately, this goal has been thwarted a number of times throughout the protocol's tumultuous lifetime, resulting in the need for a new

Force (IETF) in the mid-nineties, the protocol has been incrementally modified and extended. In the case of TLS 1.2 and below, these modifications have taken place in a largely retroactive fashion; following the announcement of an attack [6, 7, 18, 20, 32, 43, 49], the TLS Working Group (WG) would either respond by releasing a

- Symbolic model with ideal primitives
- Reasoning based on what can be derived from known terms
- Does model HKDF down to the hash function



J Cryptol (2021) 34:37
https://doi.org/10.1007/s00145-021-09384-1

Journal of CRYPTOLOGY

Check for updates

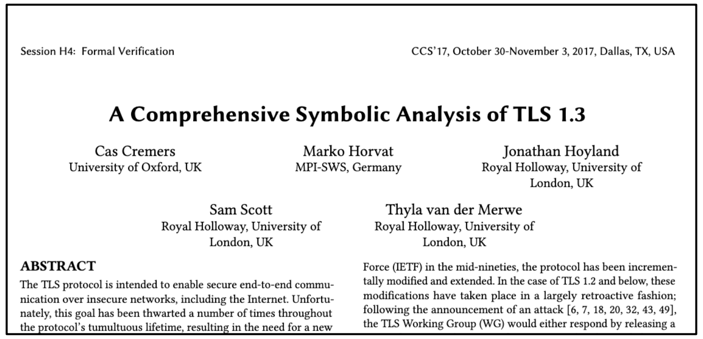
A Cryptographic Analysis of the TLS 1.3 Handshake Protocol

Benjamin Dowling
Department of Computer Science, ETH Zürich, Zurich, Switzerland

Marc Fischlin

- Computational model with computational assumptions
  - EUF-CMA, IND-1CCA, collision resistance, dual PRF, dual-snPRF-ODH, …
- Concrete non-tight bounds

Are they comparable?

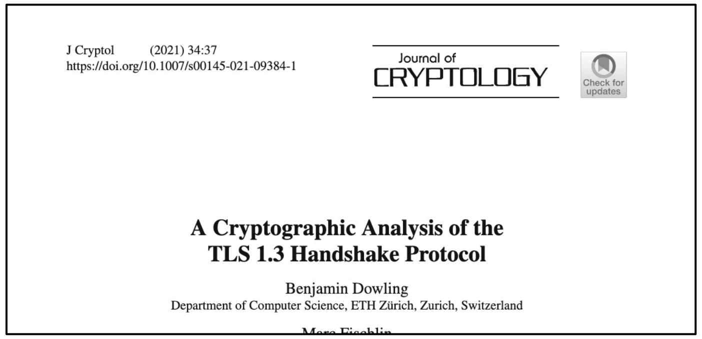


A Comprehensive Symbolic Analysis of TLS 1.3

Cas Cremers, University of Oxford, UK
Marko Horvat, MPI-SWS, Germany
Jonathan Hoyland, Royal Holloway, University of London, UK
Sam Scott, Royal Holloway, University of London, UK
Thyla van der Merwe, Royal Holloway, University of London, UK

ABSTRACT
The TLS protocol is intended to enable secure end-to-end communication over insecure networks, including the Internet. Unfortunately, this goal has been thwarted a number of times throughout the protocol's tumultuous lifetime, resulting in the need for a new

Force (IETF) in the mid-nineties, the protocol has been incrementally modified and extended. In the case of TLS 1.2 and below, these modifications have taken place in a largely retroactive fashion; following the announcement of an attack [6, 7, 18, 20, 32, 43, 49], the TLS Working Group (WG) would either respond by releasing a


J Cryptol (2021) 34:37
https://doi.org/10.1007/s00145-021-09384-1
Journal of CRYPTOLOGY
Check for updates

A Cryptographic Analysis of the TLS 1.3 Handshake Protocol

Benjamin Dowling
Department of Computer Science, ETH Zürich, Zurich, Switzerland
Marc Fischlin

Same protocol?	Incomparable	
Same security goals?	Incomparable	
Same assumptions?	Symbolic ≤ Computational (concrete)	
Same proof method?	Formal ≥ Pen-and-paper	
More citations?	184	165

Are they comparable?

From a practical perspective, not necessarily bad that they are incomparable

More perspectives and more viewpoints means less likely flaws are overlooked

This talk

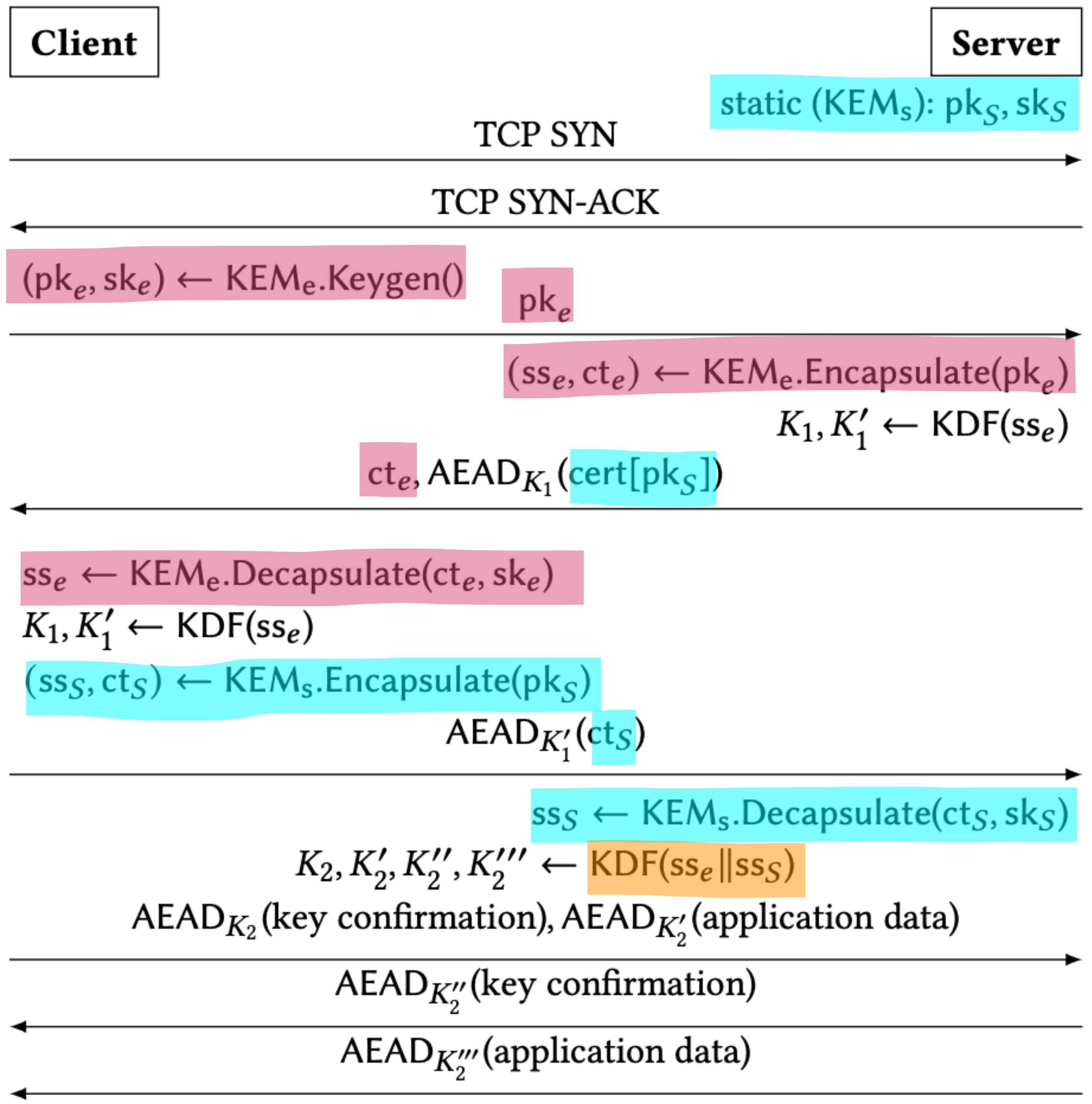
Proving KEMTLS in paper with the same protocol definition and same security properties as the pen-and-paper proof

KEMTLS

Reimagining of TLS 1.3 handshake to use key encapsulation mechanisms (KEMs) for implicit authentication, rather than digital signatures for explicit authentication

- Reduce communication sizes in PQ setting since PQ KEMs are in general smaller than PQ signatures
- Can reduce computation costs in some configurations

KEMTLS handshake


Client
Server
static (KEM$_s$): pk_S, sk_S
TCP SYN
TCP SYN-ACK
$(pk_e, sk_e) \leftarrow$ KEM$_e$.Keygen()
pk_e
$(ss_e, ct_e) \leftarrow$ KEM$_e$.Encapsulate(pk_e)
$K_1, K_1' \leftarrow$ KDF(ss_e)
ct_e, AEAD$_{K_1}$(cert[pk_S])
$ss_e \leftarrow$ KEM$_e$.Decapsulate(ct_e, sk_e)
$K_1, K_1' \leftarrow$ KDF(ss_e)
$(ss_S, ct_S) \leftarrow$ KEM$_s$.Encapsulate(pk_S)
AEAD$_{K_1'}(ct_S)$
$ss_S \leftarrow$ KEM$_s$.Decapsulate(ct_S, sk_S)
$K_2, K_2', K_2'', K_2''' \leftarrow$ KDF$(ss_e \| ss_S)$
AEAD$_{K_2}$(key confirmation), AEAD$_{K_2'}$(application data)
AEAD$_{K_2''}$(key confirmation)
AEAD$_{K_2'''}$(application data)

KEM for ephemeral key exchange

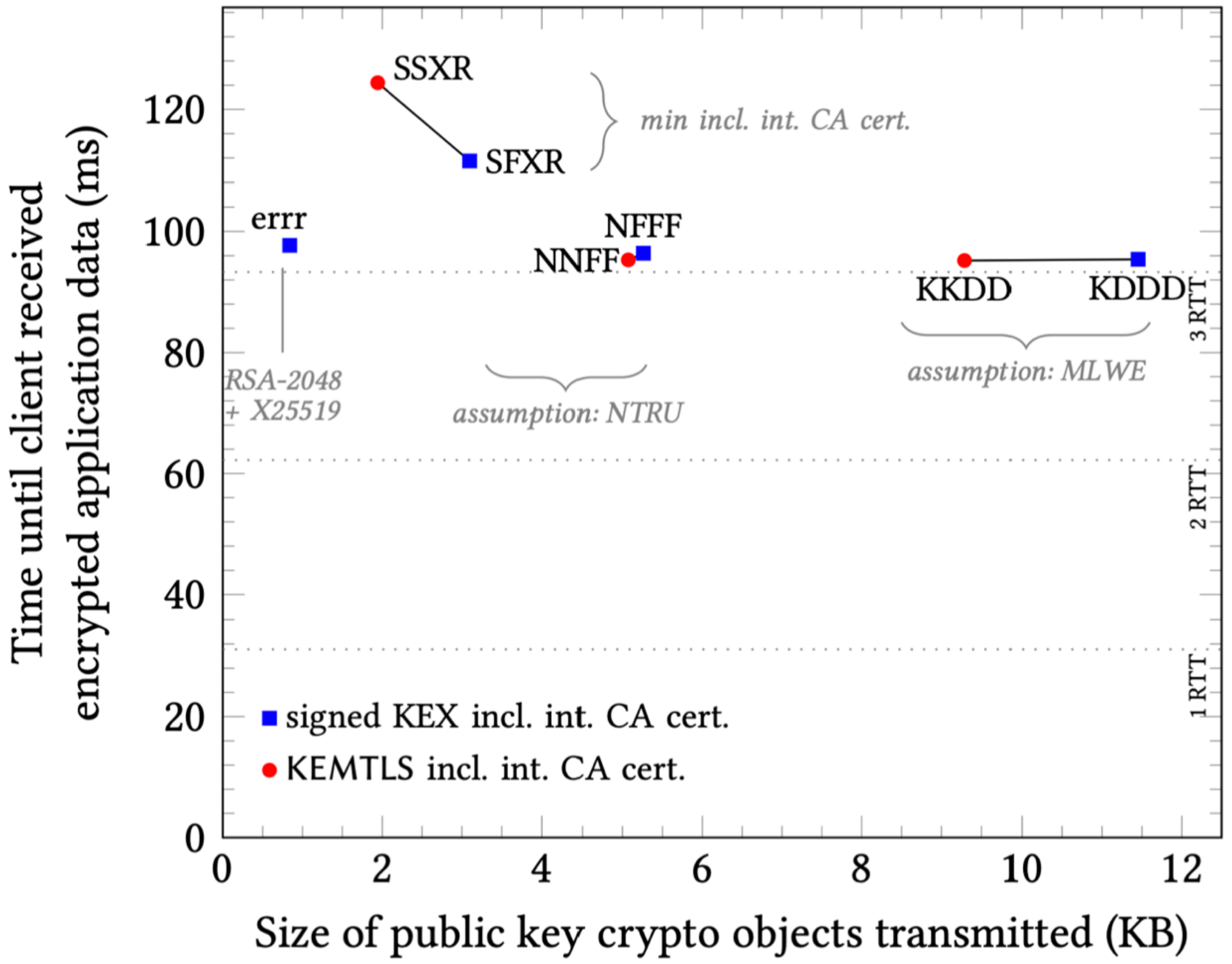
KEM for server-to-client authenticated key exchange

Combine shared secrets

Signed KEX versus KEMTLS

Labels ABCD:
A = ephemeral KEM
B = leaf certificate
C = intermediate CA
D = root CA

Algorithms: (all level 1)
Dilithium,
eCDH X25519,
Falcon,
Kyber,
NTRU,
Rainbow,
rSA-2048,
SIKE,
XMSS'



Rustls client/server with some AVX2 implementations. Emulated network: latency 31.1 ms, bandwidth 1000 Mbps, 0% packet loss. Average of 100000 iterations.

KEMTLS variants

Traditional communication flow:

1. KEMTLS server-only authentication
2. KEMTLS mutual authentication

Pre-distributed server public keys:

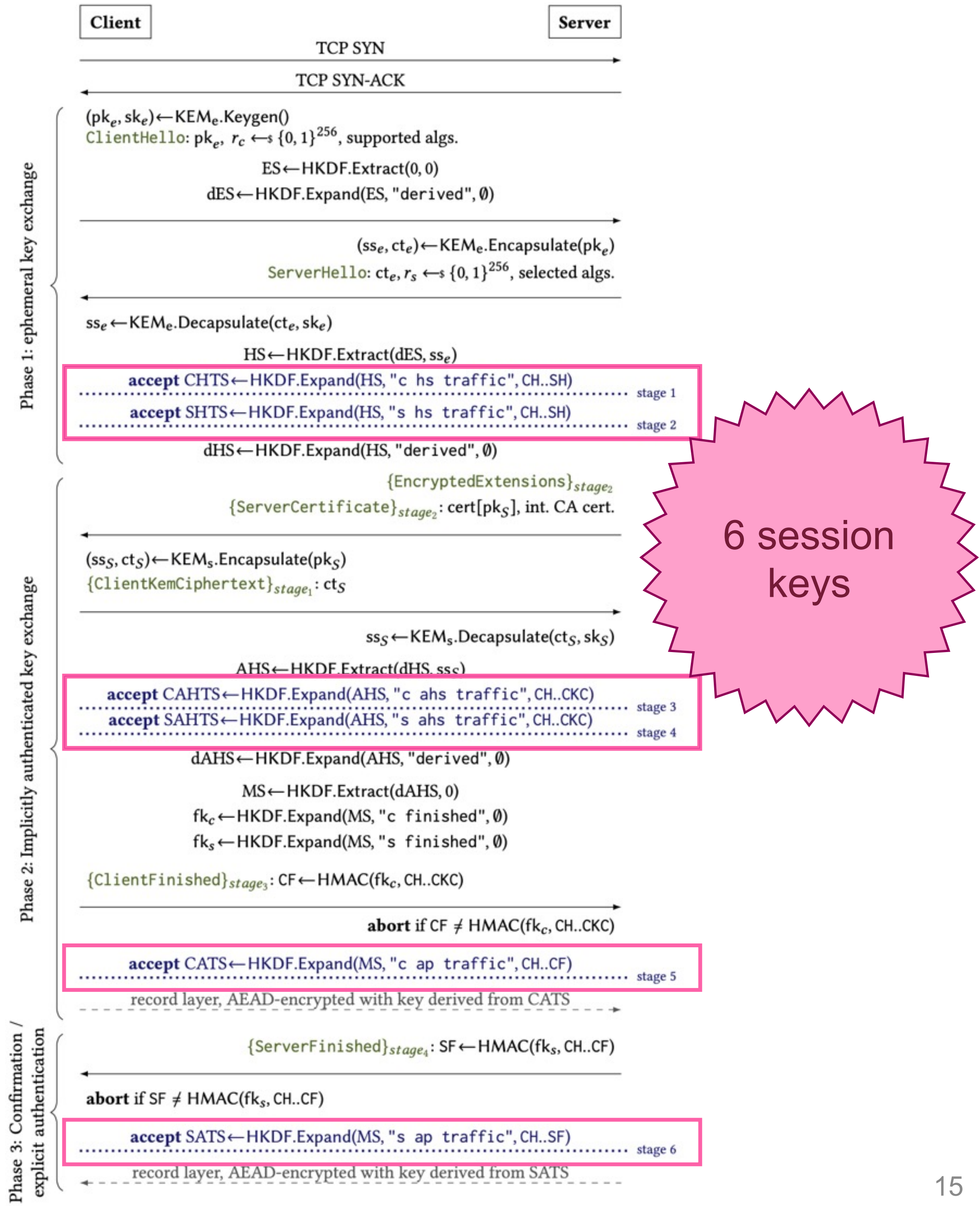
3. KEMTLS-PDK server-only authentication
4. KEMTLS-PDK mutual authentication

KEMTLS: ACM CCS 2020 https://eprint.iacr.org/2020/534
KEMTLS-PDK: ESORICS 2021 https://eprint.iacr.org/2021/779

Proving KEMTLS

Multi-stage authenticated key exchange model for KEMTLS

- → Bellare–Rogaway AKE model
- → Multi-stage AKE model [FG14]
- → Multi-stage AKE model for TLS 1.3 [DFGS15]



Client
Server
TCP SYN
TCP SYN-ACK
Phase 1: ephemeral key exchange
(pk_e, sk_e)←KEM_e.Keygen()
ClientHello: pk_e, r_c ←$ {0, 1}^256, supported algs.
ES←HKDF.Extract(0, 0)
dES←HKDF.Expand(ES, "derived", ∅)
(ss_e, ct_e)←KEM_e.Encapsulate(pk_e)
ServerHello: ct_e, r_s ←$ {0, 1}^256, selected algs.
ss_e←KEM_e.Decapsulate(ct_e, sk_e)
HS←HKDF.Extract(dES, ss_e)
accept CHTS←HKDF.Expand(HS, "c hs traffic", CH..SH)
stage 1
accept SHTS←HKDF.Expand(HS, "s hs traffic", CH..SH)
stage 2
dHS←HKDF.Expand(HS, "derived", ∅)
Phase 2: Implicitly authenticated key exchange
{EncryptedExtensions}_stage2
{ServerCertificate}_stage2: cert[pk_S], int. CA cert.
(ss_S, ct_S)←KEM_s.Encapsulate(pk_S)
{ClientKemCiphertext}_stage1: ct_S
ss_S←KEM_s.Decapsulate(ct_S, sk_S)
AHS←HKDF.Extract(dHS, ss_S)
accept CAHTS←HKDF.Expand(AHS, "c ahs traffic", CH..CKC)
stage 3
accept SAHTS←HKDF.Expand(AHS, "s ahs traffic", CH..CKC)
stage 4
dAHS←HKDF.Expand(AHS, "derived", ∅)
MS←HKDF.Extract(dAHS, 0)
fk_c←HKDF.Expand(MS, "c finished", ∅)
fk_s←HKDF.Expand(MS, "s finished", ∅)
{ClientFinished}_stage3: CF←HMAC(fk_c, CH..CKC)
abort if CF ≠ HMAC(fk_c, CH..CKC)
accept CATS←HKDF.Expand(MS, "c ap traffic", CH..CF)
stage 5
record layer, AEAD-encrypted with key derived from CATS
Phase 3: Confirmation / explicit authentication
{ServerFinished}_stage4: SF←HMAC(fk_s, CH..CF)
abort if SF ≠ HMAC(fk_s, CH..CF)
accept SATS←HKDF.Expand(MS, "s ap traffic", CH..SF)
stage 6
record layer, AEAD-encrypted with key derived from SATS
6 session keys

[BR93] Bellare, Rogaway, Crypto'93. [FG14] Fischlin, Günther, ACM CCS 2014. [DFGS15] Dowling, Fischlin, Günther, Stebila, ACM CCS 2015.

Multi-stage AKE model

Queries

- NewSession
- Send
- CorruptLongTermKey
- RevealSessionKey
- Test

Variables

- Session π
- π.owner
- π.peerid
- π.role
- π.stage
- π.sid[1..M]
- π.cid[1..M]
- π.key[1..M]
- π.state

Model parameters

- π.auth[1..M]
- π.fs[1..M][1..M]
- π.use[1..M]

Multi-stage AKE model

Security properties

- **Match security**
  - 6 conditions about session identifier matching
- **Offline deniability**
- **Multi-stage key indistinguishability**
  - 3 levels of forward secrecy
- **Authentication** (malicious acceptance)
- Per-stage properties
- Can be retroactively upgraded by acceptance of later stages

Limitations of pen-and-paper proofs

- Mostly written out for session-key indistinguishability for KEMTLS and KEMTLS-PDK server-only auth variants
  - No explicit games / reductions*
  - But only as reliable as the authors and the readers are
- Proof sketches for session-key indistinguishability of remaining variants
- Hand-waving argument for offline deniability
- Variants handled independently

* In the public versions. I did write them out for KEMTLS in a private version with painful LaTeX macros.

Formal verification using Tamarin

- **Tamarin prover** is a model checker for security protocols in the symbolic model
- Protocol and adversary powers are specified as a set of state machine transitions ("multiset rewriting rules")
- Security property is specified as a predicate over actions recorded during state machine transitions
- Tamarin prover explores (infinite) state space of all possible executions to find an execution trace that violates the security property or verifies that none exists (or fails to terminate)

https://tamarin-prover.github.io/

Formal verification using Tamarin

- Tamarin successfully used on many academic and real-world cryptographic protocols
- Especially effective on key exchange protocols
  - Note Tamarin models key exchange security based on *learning* session key, not *indistinguishability*

- Tamarin model of TLS 1.3 drafts [CHSV,CHHSV] found several flaws
  - Especially in interactions between different protocols modes
    - e.g. in TLS 1.3 pre-shared key resumption
  - Expensive: months of person-effort, 1 week of computation time, 100 GB RAM

[CHSV] Cremers, Horvat, Scott, van der Merwe, IEEE S&P 2016.
[CHHSV] Cremers, Horvat, Hoyland, Scott, van der Merwe, ACM CCS 2017.

Modelling KEMTLS using Tamarin

Approach 1

https://github.com/thomwiggers/TLS13Tamarin

- Adapt [CHHSV] full-scale Tamarin model of TLS 1.3 to KEMTLS
- High resolution protocol specification: captures TLS message format, internal KDF structure, …
- Lower resolution security properties
- Required more human effort to get proofs running automatically

Approach 2

https://github.com/dstebila/KEMTLS-Tamarin

- Encode pen-and-paper multi-stage AKE definitions in Tamarin
- Lower resolution protocol specification: "core cryptographic" of KEMTLS
  - E.g. No TLS message structure
- Higher resolution security properties
- Simpler to specify and automatically proves

[CHHSV] Cremers, Horvat, Hoyland, Scott, van der Merwe, ACM CCS 2017.

Feature	**In model of**	
	Approach 1	**Approach 2**
Protocol modelling		
Encrypted handshake messages	✓	✗
HKDF and HMAC decomposed into hash calls	✓	✗
Key exch. and auth. KEMs are the same algorithm	✓	✗
TLS message structure	✓	✗
Security properties		
Adversary can reveal long-term keys	✓	✓
Adversary can reveal ephemeral keys	✓	✗
Adversary can reveal intermediate session keys	✗	✓
Secrecy of handshake and application traffic keys	✓	✓
Forward secrecy	✓	✓
Multiple flavours of forward secrecy	✗	✓
Explicit authentication	✓	✓
Deniability	✗	✓

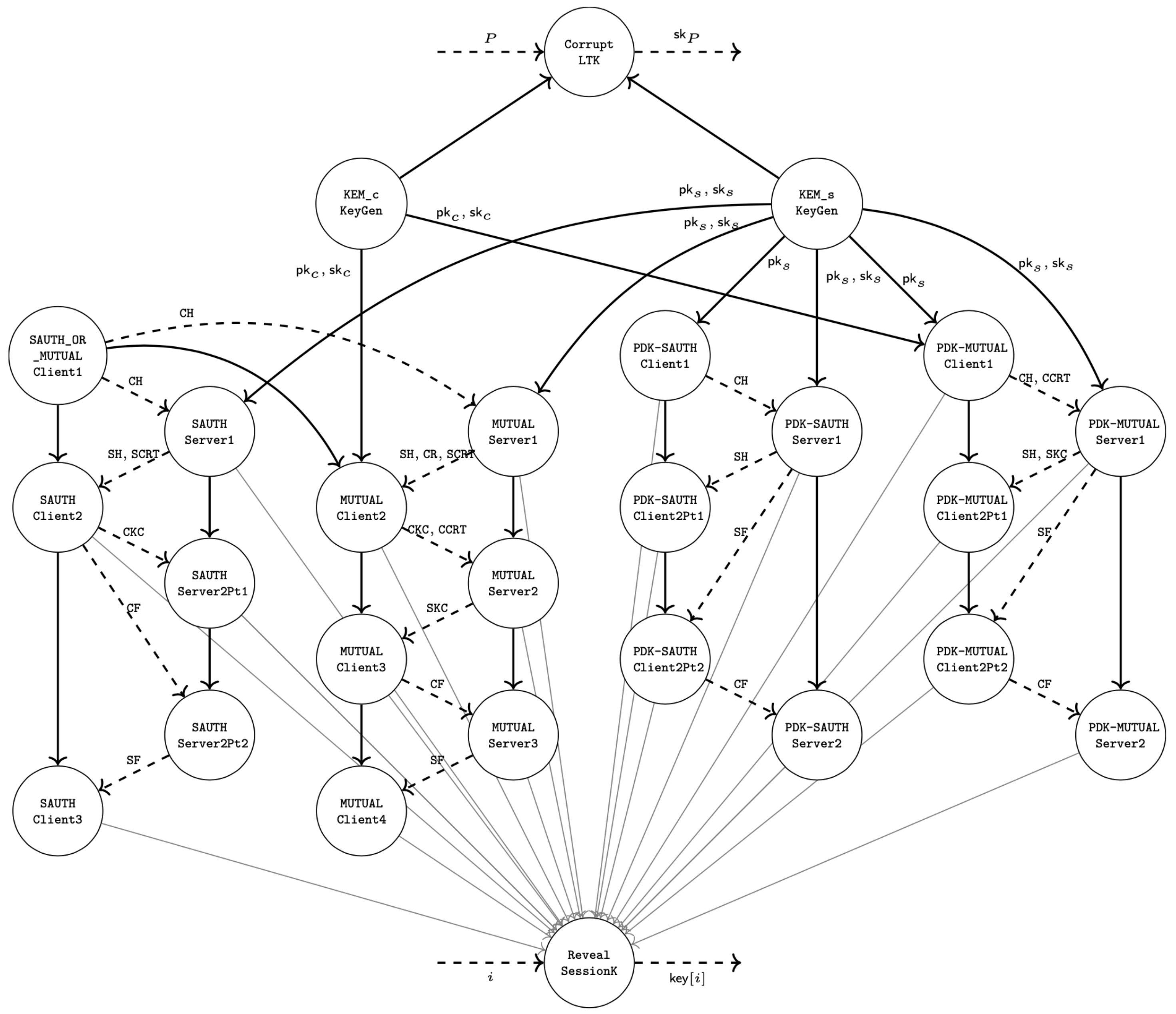
Queries

Pen-and-paper	Tamarin approach 2
	KetGenLTK
NewSession	KEMTLS_SAUTH_ClientAction1 KEMTLS_SAUTH_ServerAction1 KEMTLS_SAUTH_ClientAction2 …
Send	
CorruptLongTermKey	OCorruptLTK
RevealSessionKey	ORevealSessionKey
Test	*No Test oracle – symbolic model based on key recovery*

Variables

Pen-and-paper	Tamarin approach 2
Session π	Thread identifier tid
	ProtocolMode fact
π.owner	Owner fact
π.peerid	Peer fact
π.role	Role fact
π.stage	Implicit part of state
π.sid[1..M]	SID facts
π.cid[1..M]	CID facts
π.key[1..M]	SK facts
π.auth[1..M]	Auth facts
π.fs[1..M][1..M]	FS facts
π.use[1..M]	Replayable facts
π.state	State fact

Protocol state machine (oracles)



Model parameters

Parameter	KEMTLS **Server-only auth.**	KEMTLS **Mutual auth.**	KEMTLS-PDK **Server-only auth.**	KEMTLS-PDK **Mutual auth.**
auth^C	$(6^{\times 6})$	same as server-only auth.	$(4^{\times 4}, \infty)$	$(4^{\times 4}, \infty)$
auth^S	$(\infty^{\times 6})$	$(5^{\times 5}, \infty)$	$(\infty^{\times 5})$	$(5^{\times 5})$
FS^C	$\begin{pmatrix} \mathsf{wfs1} & \mathsf{wfs1} & \mathsf{wfs1} & \mathsf{wfs1} & \mathsf{wfs1} & \mathsf{fs} \\ & \mathsf{wfs1} & \mathsf{wfs1} & \mathsf{wfs1} & \mathsf{wfs1} & \mathsf{fs} \\ & & \mathsf{wfs2} & \mathsf{wfs2} & \mathsf{wfs2} & \mathsf{fs} \\ & & & \mathsf{wfs2} & \mathsf{wfs2} & \mathsf{fs} \\ & & & & \mathsf{wfs2} & \mathsf{fs} \\ & & & & & \mathsf{fs} \end{pmatrix}$	same as server-only auth.	$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ & \mathsf{wfs2} & \mathsf{wfs2} & \mathsf{fs} & \mathsf{fs} \\ & & \mathsf{wfs2} & \mathsf{fs} & \mathsf{fs} \\ & & & \mathsf{fs} & \mathsf{fs} \\ & & & & \mathsf{fs} \end{pmatrix}$	same as server-only auth.
FS^S	$\mathsf{FS}^S_{i,j} = \mathsf{wfs1}$ for all $j \geq i$	$\begin{pmatrix} \mathsf{wfs1} & \mathsf{wfs1} & \mathsf{wfs1} & \mathsf{wfs1} & \mathsf{fs} & \mathsf{fs} \\ & \mathsf{wfs1} & \mathsf{wfs1} & \mathsf{wfs1} & \mathsf{fs} & \mathsf{fs} \\ & & \mathsf{wfs1} & \mathsf{wfs1} & \mathsf{fs} & \mathsf{fs} \\ & & & \mathsf{wfs1} & \mathsf{fs} & \mathsf{fs} \\ & & & & \mathsf{fs} & \mathsf{fs} \\ & & & & & \mathsf{fs} \end{pmatrix}$	$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ & \mathsf{wfs1} & \mathsf{wfs1} & \mathsf{wfs1} & \mathsf{wfs1} \\ & & \mathsf{wfs1} & \mathsf{wfs1} & \mathsf{wfs1} \\ & & & \mathsf{wfs1} & \mathsf{wfs1} \\ & & & & \mathsf{wfs1} \end{pmatrix}$	$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ & \mathsf{wfs1} & \mathsf{wfs1} & \mathsf{wfs1} & \mathsf{fs} \\ & & \mathsf{wfs1} & \mathsf{wfs1} & \mathsf{fs} \\ & & & \mathsf{wfs2} & \mathsf{fs} \\ & & & & \mathsf{fs} \end{pmatrix}$
replay	$(\mathsf{nonreplayable}^{\times 6})$	same as server-only auth.	$(\mathsf{replayable}, \mathsf{nonreplayable}^{\times 4})$	same as server-only auth.

Theorems

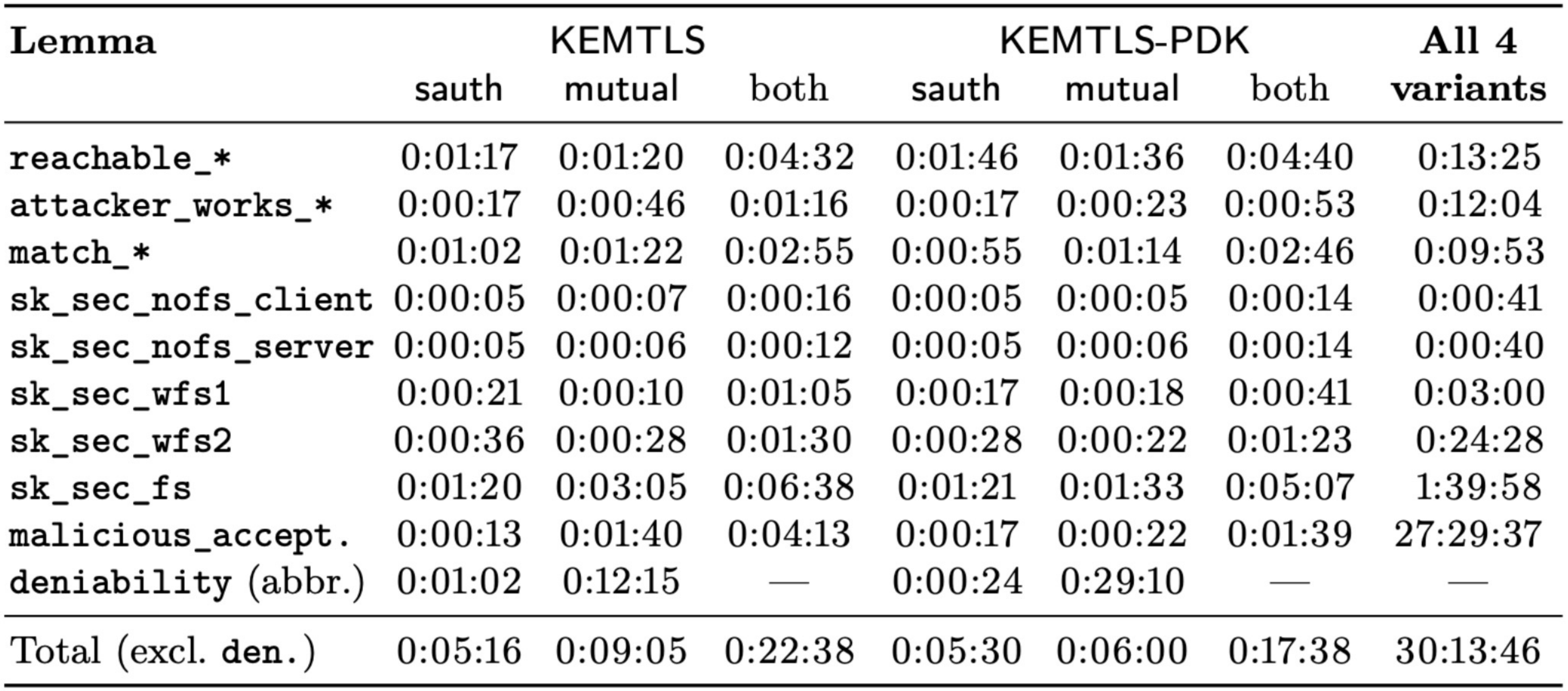
Pen-and-paper	Tamarin approach 2
	Reachable: It is possible for the adversary to cause stage i of protocol mode j to accept with its intended security properties.
	Attacker works: It is possible for the adversary to learn session key of stage i of protocol mode j (when no freshness restrictions).
Match security	**Match security** properties 1, 2, 3, 4, 5, 6, 7
Multi-stage session key indistinguishability * Single protocol mode available	**Session key unrecoverability** of keys tagged nofs (client), nofs (server), wfs1, wfs2, fs * All protocol modes available simultaneously
Authentication	**Authentication**
Offline deniability	**Offline deniability** (transcript indistinguishability using observational equivalence)

Definition B.3 (Freshness). Stage i of a session π is said to be *fresh* in the sense of *weak forward secrecy 1* if:

(1) the stage key was not revealed (π.revealed$_i$ = false);
(2) the stage key of the partner session at stage i, if the partner exists, has not been revealed (for all i, π' such that $\pi.\mathrm{sid}_i = \pi'.\mathrm{sid}_i$, we have that $\pi'.\mathrm{revealed}_i = \mathrm{false}$);
(3) there exists $j \geq i$ such that $\pi.\mathrm{FS}_{i,j} = \mathrm{wfs1}$, $\pi.\mathrm{status}_j = \mathrm{accepted}$, and there exists a contributive partner at stage i.

```
lemma sk_security_wfs1: "
  All tid_owner i cid_i sid_i sk_i #taccept #tcid #tsid #tsk .
  // if stage i has accepted a session key with corresponding contributive and session identifiers
  Accept(tid_owner, i) @ #taccept & SK(tid_owner, i, sk_i) @ #tsk
  & CID(tid_owner, i, cid_i) @ #tcid & SID(tid_owner, i, sid_i) @ #tsid
  // and it is fresh in the sense of wfs1, namely that
  // (1) the stage key was not revealed
  & not(Ex #t . RevealedSessionKey(tid_owner, i) @ #t)
  // and (2) the stage key of the partner session at stage i, if the partner exists,  has not been revealed
  & not(
    Ex tid_partner #tt1 #tt2 . not(tid_owner = tid_partner)
    & SID(tid_partner, i, sid_i) @ #tt1 & RevealedSessionKey(tid_partner, i) @ #tt2)
  // and (3) there exists j ≥ i s.t. Pi.FS_{i,j} = wfs1, Pi.status_j = accepted,
  // and there exists a contributive partner at stage i
  & (
    Ex j #tfs #tacceptj . FS(tid_owner, i, j, 'wfs1') @ #tfs
    & Accept(tid_owner, j) @ #tacceptj
    & (Ex tid_peer #tacceptjpeer #tcidpeer . not(tid_owner = tid_peer)
      & Accept(tid_peer, j) @ #tacceptjpeer & CID(tid_peer, i, cid_i) @ #tcidpeer))
  // then the session key cannot be learned by the adversary
  ==> not(Ex #t . KU(sk_i) @ #t)"
```

Tamarin runtimes for Approach 2



Lemma	KEMTLS			KEMTLS-PDK			All 4
	sauth	mutual	both	sauth	mutual	both	variants
reachable_*	0:01:17	0:01:20	0:04:32	0:01:46	0:01:36	0:04:40	0:13:25
attacker_works_*	0:00:17	0:00:46	0:01:16	0:00:17	0:00:23	0:00:53	0:12:04
match_*	0:01:02	0:01:22	0:02:55	0:00:55	0:01:14	0:02:46	0:09:53
sk_sec_nofs_client	0:00:05	0:00:07	0:00:16	0:00:05	0:00:05	0:00:14	0:00:41
sk_sec_nofs_server	0:00:05	0:00:06	0:00:12	0:00:05	0:00:06	0:00:14	0:00:40
sk_sec_wfs1	0:00:21	0:00:10	0:01:05	0:00:17	0:00:18	0:00:41	0:03:00
sk_sec_wfs2	0:00:36	0:00:28	0:01:30	0:00:28	0:00:22	0:01:23	0:24:28
sk_sec_fs	0:01:20	0:03:05	0:06:38	0:01:21	0:01:33	0:05:07	1:39:58
malicious_accept.	0:00:13	0:01:40	0:04:13	0:00:17	0:00:22	0:01:39	27:29:37
deniability (abbr.)	0:01:02	0:12:15	—	0:00:24	0:29:10	—	—
Total (excl. den.)	0:05:16	0:09:05	0:22:38	0:05:30	0:06:00	0:17:38	30:13:46

Are they comparable?

	KEMTLS pen-and-paper	KEMTLS Tamarin approach 2
Same protocol?	Pretty close	
Same security goals?	Pretty close*	
Same assumptions?	Computational ≥ Symbolic	
Same proof method	Pen-and-paper ≤ Formal	

* Most significant difference: session key indistinguishability versus session key recovery

Tamarin found bugs in pen-and-paper proof

Parameter	KEMTLS Server-only auth.	KEMTLS Mutual auth.	KEMTLS-PDK Server-only auth.	KEMTLS-PDK Mutual auth.
auth^C	$(6^{\times 6})$	same as server-only auth.	$(4^{\times 4}, \infty)$	$(4^{\times 4}, \infty)$
[illegible]	[illegible]	$(5^{\times 5}, \infty)$	$(\infty^{\times 5})$	$(5^{\times 5})$
[illegible]	[illegible]	same as server-only auth.	$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ & \mathsf{wfs2} & \mathsf{wfs2} & \mathsf{fs} & \mathsf{fs} \\ & & \mathsf{wfs2} & \mathsf{fs} & \mathsf{fs} \\ & & & \mathsf{fs} & \mathsf{fs} \\ & & & & \mathsf{fs} \end{pmatrix}$	same as server-only auth.
FS^S	$\mathsf{FS}^S_{i,j} = \mathsf{wfs1}$ for all $j \geq i$	$\begin{pmatrix} \mathsf{wfs1} & \mathsf{wfs1} & \mathsf{wfs1} & \mathsf{wfs1} & \mathsf{fs} & \mathsf{fs} \\ & \mathsf{wfs1} & \mathsf{wfs1} & \mathsf{wfs1} & \mathsf{fs} & \mathsf{fs} \\ & & \mathsf{wfs1} & \mathsf{wfs1} & \mathsf{fs} & \mathsf{fs} \\ & & & \mathsf{wfs1} & \mathsf{fs} & \mathsf{fs} \\ & & & & \mathsf{fs} & \mathsf{fs} \\ & & & & & \mathsf{fs} \end{pmatrix}$	$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ & \mathsf{wfs1} & \mathsf{wfs1} & \mathsf{wfs1} & \mathsf{wfs1} \\ & & \mathsf{wfs1} & \mathsf{wfs1} & \mathsf{wfs1} \\ & & & \mathsf{wfs1} & \mathsf{wfs1} \\ & & & & \mathsf{wfs1} \end{pmatrix}$	$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ & \mathsf{wfs1} & \mathsf{wfs1} & \mathsf{wfs1} & \mathsf{fs} \\ & & \mathsf{wfs1} & \mathsf{wfs1} & \mathsf{fs} \\ & & & \mathsf{wfs2} & \mathsf{fs} \\ & & & & \mathsf{fs} \end{pmatrix}$
replay	$(\mathsf{nonreplayable}^{\times 6})$	same as server-only auth.	$(\mathsf{replayable}, \mathsf{nonreplayable}^{\times 4})$	same as server-only auth.

Can easily tweak security definitions / lemmas to check you have the properties right / optimal.

It’s not that scary

- I’d only done a Tamarin tutorial before this
- No intermediate lemmas
- No manual proving in Tamarin; everything proved automatically
- About 30-40 hours of work encoding protocol and security properties

- Can it be used by someone who wasn’t a creator of the tool?

It was fun

- Got to run some big computing jobs!
- Validates that prose-based pen-and-paper models can be rigorized
- I found it rewarding to see it come together and validate the pen-and-paper work

Proving KEMTLS in Tamarin

I used Tamarin and you can too!

Douglas Stebila

University of Waterloo

KEMTLS

Implicitly authenticated TLS without handshake signatures using KEMs

- Saves bytes on the wire, server cycles
- Variants for client authentication and pre-distributed public keys

Can encode multi-stage AKE model in Tamarin

- Same protocol, same security properties

All four protocol variants simultaneously

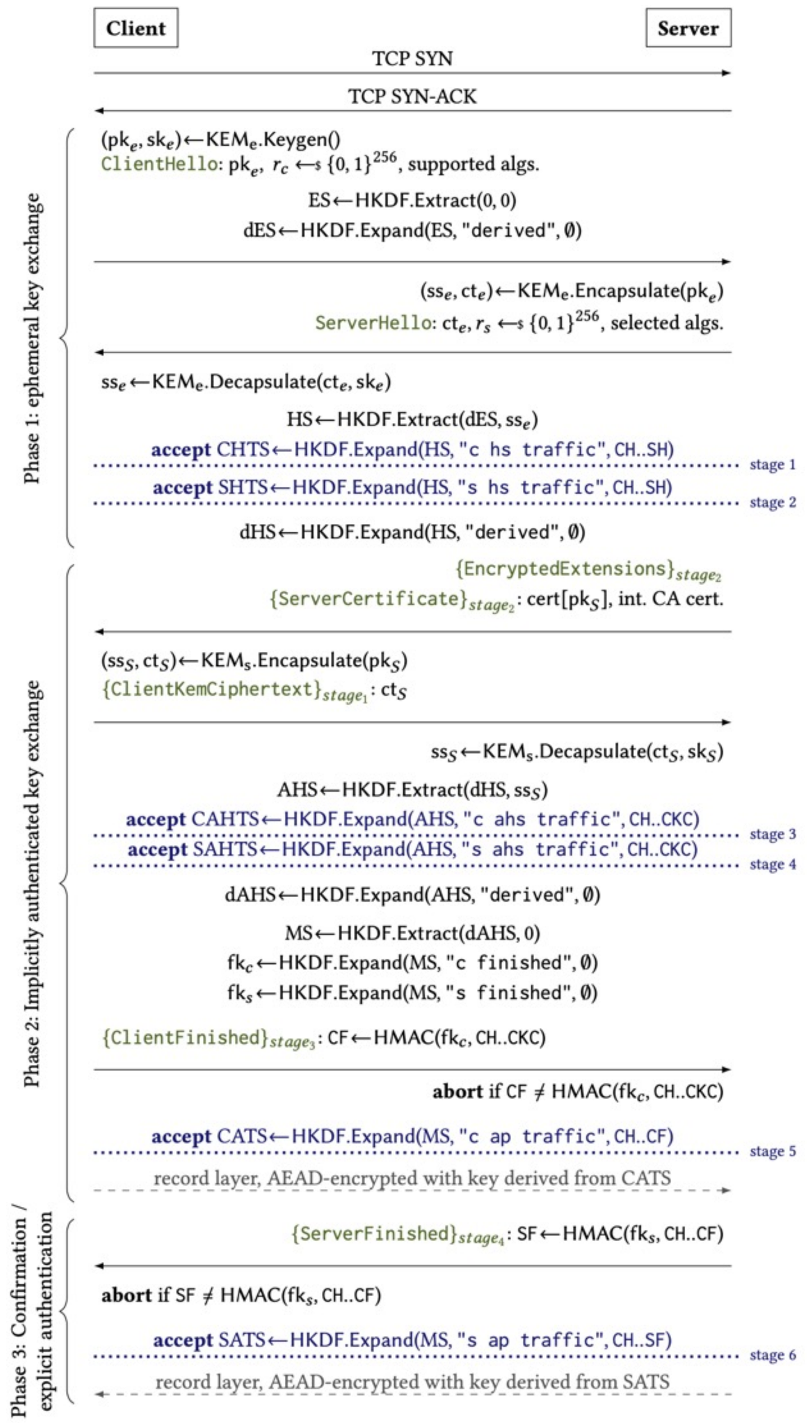
Not too hard to state or prove

Identified bugs in pen-and-paper proofs

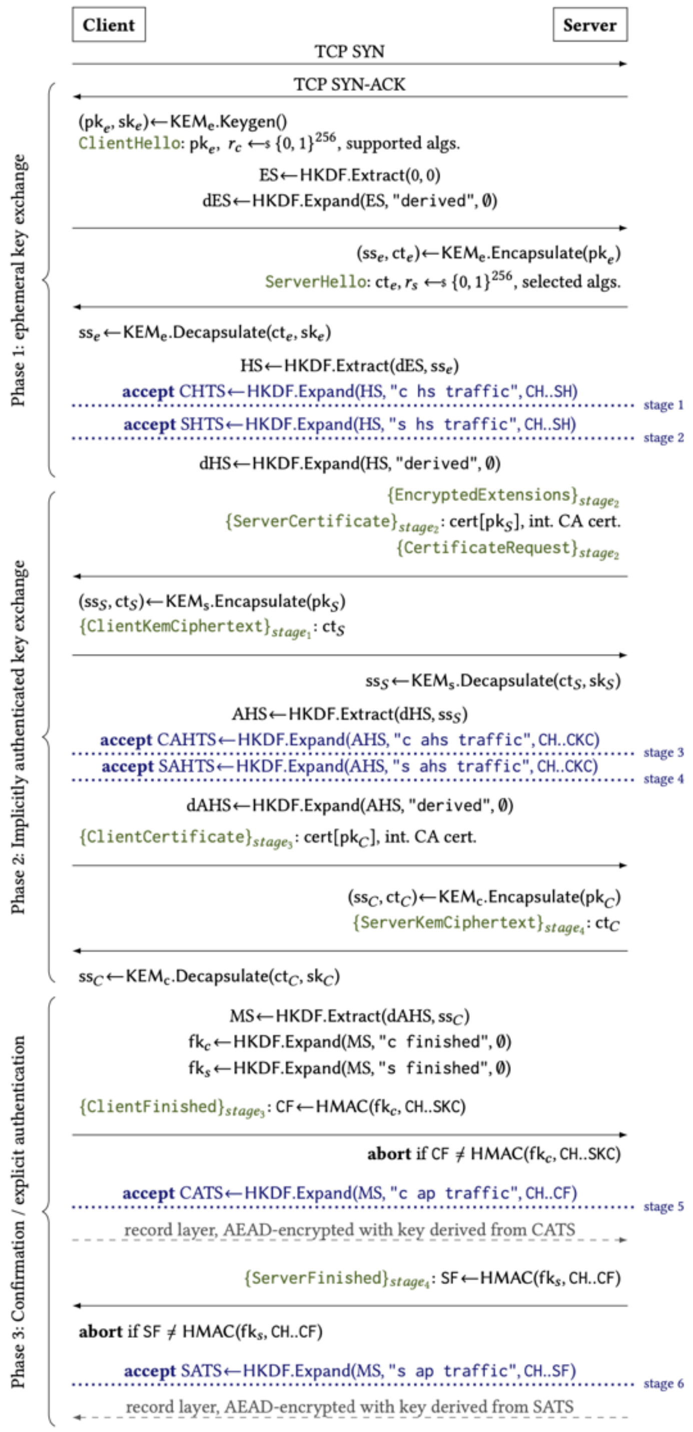
https://kemtls.org/
https://eprint.iacr.org/2020/534 • https://eprint.iacr.org/2021/779
https://datatracker.ietf.org/doc/html/draft-celi-wiggers-tls-authkem-00
https://github.com/thomwiggers/TLS13Tamarin • https://github.com/dstebila/KEMTLS-Tamarin/

Appendix

KEMTLS

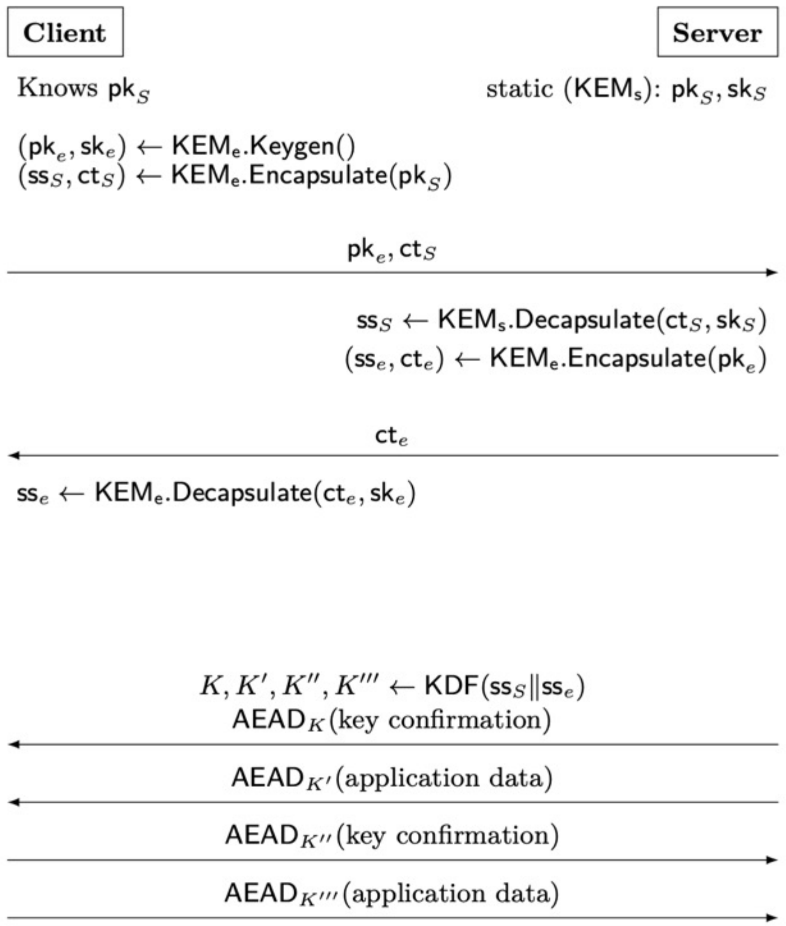

Client
Server
TCP SYN
TCP SYN-ACK
Phase 1: ephemeral key exchange
$(\mathsf{pk}_e, \mathsf{sk}_e) \leftarrow \mathsf{KEM}_e.\mathsf{Keygen}()$
ClientHello: pk_e, $r_c \leftarrow\!\!\$ \{0,1\}^{256}$, supported algs.
$\mathsf{ES} \leftarrow \mathsf{HKDF.Extract}(0, 0)$
$\mathsf{dES} \leftarrow \mathsf{HKDF.Expand}(\mathsf{ES}, \texttt{"derived"}, \emptyset)$
$(\mathsf{ss}_e, \mathsf{ct}_e) \leftarrow \mathsf{KEM}_e.\mathsf{Encapsulate}(\mathsf{pk}_e)$
ServerHello: ct_e, $r_s \leftarrow\!\!\$ \{0,1\}^{256}$, selected algs.
$\mathsf{ss}_e \leftarrow \mathsf{KEM}_e.\mathsf{Decapsulate}(\mathsf{ct}_e, \mathsf{sk}_e)$
$\mathsf{HS} \leftarrow \mathsf{HKDF.Extract}(\mathsf{dES}, \mathsf{ss}_e)$
accept $\mathsf{CHTS} \leftarrow \mathsf{HKDF.Expand}(\mathsf{HS}, \texttt{"c hs traffic"}, \mathsf{CH..SH})$
stage 1
accept $\mathsf{SHTS} \leftarrow \mathsf{HKDF.Expand}(\mathsf{HS}, \texttt{"s hs traffic"}, \mathsf{CH..SH})$
stage 2
$\mathsf{dHS} \leftarrow \mathsf{HKDF.Expand}(\mathsf{HS}, \texttt{"derived"}, \emptyset)$
Phase 2: Implicitly authenticated key exchange
$\{\texttt{EncryptedExtensions}\}_{stage_2}$
$\{\texttt{ServerCertificate}\}_{stage_2}$: $\mathsf{cert}[\mathsf{pk}_S]$, int. CA cert.
$(\mathsf{ss}_S, \mathsf{ct}_S) \leftarrow \mathsf{KEM}_s.\mathsf{Encapsulate}(\mathsf{pk}_S)$
$\{\texttt{ClientKemCiphertext}\}_{stage_1}$: ct_S
$\mathsf{ss}_S \leftarrow \mathsf{KEM}_s.\mathsf{Decapsulate}(\mathsf{ct}_S, \mathsf{sk}_S)$
$\mathsf{AHS} \leftarrow \mathsf{HKDF.Extract}(\mathsf{dHS}, \mathsf{ss}_S)$
accept $\mathsf{CAHTS} \leftarrow \mathsf{HKDF.Expand}(\mathsf{AHS}, \texttt{"c ahs traffic"}, \mathsf{CH..CKC})$
stage 3
accept $\mathsf{SAHTS} \leftarrow \mathsf{HKDF.Expand}(\mathsf{AHS}, \texttt{"s ahs traffic"}, \mathsf{CH..CKC})$
stage 4
$\mathsf{dAHS} \leftarrow \mathsf{HKDF.Expand}(\mathsf{AHS}, \texttt{"derived"}, \emptyset)$
$\mathsf{MS} \leftarrow \mathsf{HKDF.Extract}(\mathsf{dAHS}, 0)$
$\mathsf{fk}_c \leftarrow \mathsf{HKDF.Expand}(\mathsf{MS}, \texttt{"c finished"}, \emptyset)$
$\mathsf{fk}_s \leftarrow \mathsf{HKDF.Expand}(\mathsf{MS}, \texttt{"s finished"}, \emptyset)$
$\{\texttt{ClientFinished}\}_{stage_3}$: $\mathsf{CF} \leftarrow \mathsf{HMAC}(\mathsf{fk}_c, \mathsf{CH..CKC})$
abort if $\mathsf{CF} \neq \mathsf{HMAC}(\mathsf{fk}_c, \mathsf{CH..CKC})$
accept $\mathsf{CATS} \leftarrow \mathsf{HKDF.Expand}(\mathsf{MS}, \texttt{"c ap traffic"}, \mathsf{CH..CF})$
stage 5
record layer, AEAD-encrypted with key derived from CATS
Phase 3: Confirmation / explicit authentication
$\{\texttt{ServerFinished}\}_{stage_4}$: $\mathsf{SF} \leftarrow \mathsf{HMAC}(\mathsf{fk}_s, \mathsf{CH..CF})$
abort if $\mathsf{SF} \neq \mathsf{HMAC}(\mathsf{fk}_s, \mathsf{CH..CF})$
accept $\mathsf{SATS} \leftarrow \mathsf{HKDF.Expand}(\mathsf{MS}, \texttt{"s ap traffic"}, \mathsf{CH..SF})$
stage 6
record layer, AEAD-encrypted with key derived from SATS

KEMTLS with client authentication

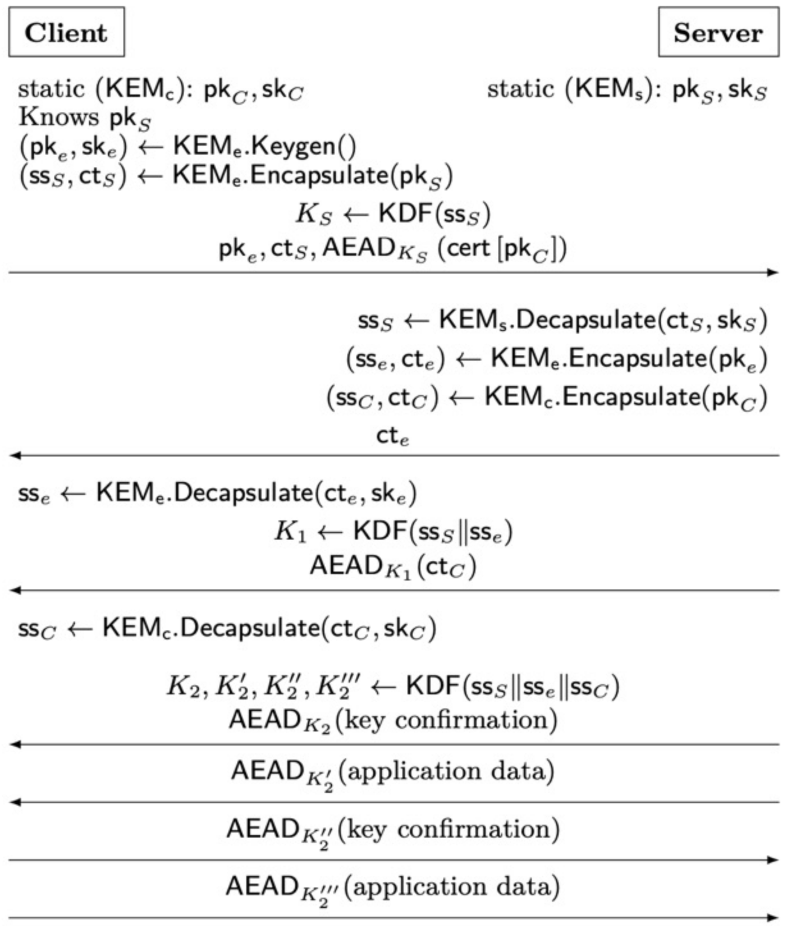


Client
Server
TCP SYN
TCP SYN-ACK
Phase 1: ephemeral key exchange
$(\mathsf{pk}_e, \mathsf{sk}_e) \leftarrow \mathsf{KEM}_e.\mathsf{Keygen}()$
ClientHello: pk_e, $r_c \leftarrow_s \{0,1\}^{256}$, supported algs.
$\mathsf{ES} \leftarrow \mathsf{HKDF.Extract}(0, 0)$
$\mathsf{dES} \leftarrow \mathsf{HKDF.Expand}(\mathsf{ES}, \texttt{"derived"}, \emptyset)$
$(\mathsf{ss}_e, \mathsf{ct}_e) \leftarrow \mathsf{KEM}_e.\mathsf{Encapsulate}(\mathsf{pk}_e)$
ServerHello: ct_e, $r_s \leftarrow_s \{0,1\}^{256}$, selected algs.
$\mathsf{ss}_e \leftarrow \mathsf{KEM}_e.\mathsf{Decapsulate}(\mathsf{ct}_e, \mathsf{sk}_e)$
$\mathsf{HS} \leftarrow \mathsf{HKDF.Extract}(\mathsf{dES}, \mathsf{ss}_e)$
accept CHTS ← HKDF.Expand(HS, "c hs traffic", CH..SH)
stage 1
accept SHTS ← HKDF.Expand(HS, "s hs traffic", CH..SH)
stage 2
$\mathsf{dHS} \leftarrow \mathsf{HKDF.Expand}(\mathsf{HS}, \texttt{"derived"}, \emptyset)$
Phase 2: Implicitly authenticated key exchange
$\{\texttt{EncryptedExtensions}\}_{stage_2}$
$\{\texttt{ServerCertificate}\}_{stage_2}$: cert[$\mathsf{pk}_S$], int. CA cert.
$\{\texttt{CertificateRequest}\}_{stage_2}$
$(\mathsf{ss}_S, \mathsf{ct}_S) \leftarrow \mathsf{KEM}_s.\mathsf{Encapsulate}(\mathsf{pk}_S)$
$\{\texttt{ClientKemCiphertext}\}_{stage_1}$: ct_S
$\mathsf{ss}_S \leftarrow \mathsf{KEM}_s.\mathsf{Decapsulate}(\mathsf{ct}_S, \mathsf{sk}_S)$
$\mathsf{AHS} \leftarrow \mathsf{HKDF.Extract}(\mathsf{dHS}, \mathsf{ss}_S)$
accept CAHTS ← HKDF.Expand(AHS, "c ahs traffic", CH..CKC)
stage 3
accept SAHTS ← HKDF.Expand(AHS, "s ahs traffic", CH..CKC)
stage 4
$\mathsf{dAHS} \leftarrow \mathsf{HKDF.Expand}(\mathsf{AHS}, \texttt{"derived"}, \emptyset)$
$\{\texttt{ClientCertificate}\}_{stage_3}$: cert[$\mathsf{pk}_C$], int. CA cert.
$(\mathsf{ss}_C, \mathsf{ct}_C) \leftarrow \mathsf{KEM}_c.\mathsf{Encapsulate}(\mathsf{pk}_C)$
$\{\texttt{ServerKemCiphertext}\}_{stage_4}$: ct_C
$\mathsf{ss}_C \leftarrow \mathsf{KEM}_c.\mathsf{Decapsulate}(\mathsf{ct}_C, \mathsf{sk}_C)$
Phase 3: Confirmation / explicit authentication
$\mathsf{MS} \leftarrow \mathsf{HKDF.Extract}(\mathsf{dAHS}, \mathsf{ss}_C)$
$\mathsf{fk}_c \leftarrow \mathsf{HKDF.Expand}(\mathsf{MS}, \texttt{"c finished"}, \emptyset)$
$\mathsf{fk}_s \leftarrow \mathsf{HKDF.Expand}(\mathsf{MS}, \texttt{"s finished"}, \emptyset)$
$\{\texttt{ClientFinished}\}_{stage_3}$: $\mathsf{CF} \leftarrow \mathsf{HMAC}(\mathsf{fk}_c, \mathsf{CH..SKC})$
abort if $\mathsf{CF} \neq \mathsf{HMAC}(\mathsf{fk}_c, \mathsf{CH..SKC})$
accept CATS ← HKDF.Expand(MS, "c ap traffic", CH..CF)
stage 5
record layer, AEAD-encrypted with key derived from CATS
$\{\texttt{ServerFinished}\}_{stage_4}$: $\mathsf{SF} \leftarrow \mathsf{HMAC}(\mathsf{fk}_s, \mathsf{CH..CF})$
abort if $\mathsf{SF} \neq \mathsf{HMAC}(\mathsf{fk}_s, \mathsf{CH..CF})$
accept SATS ← HKDF.Expand(MS, "s ap traffic", CH..SF)
stage 6
record layer, AEAD-encrypted with key derived from SATS

KEMTLS-PDK overview

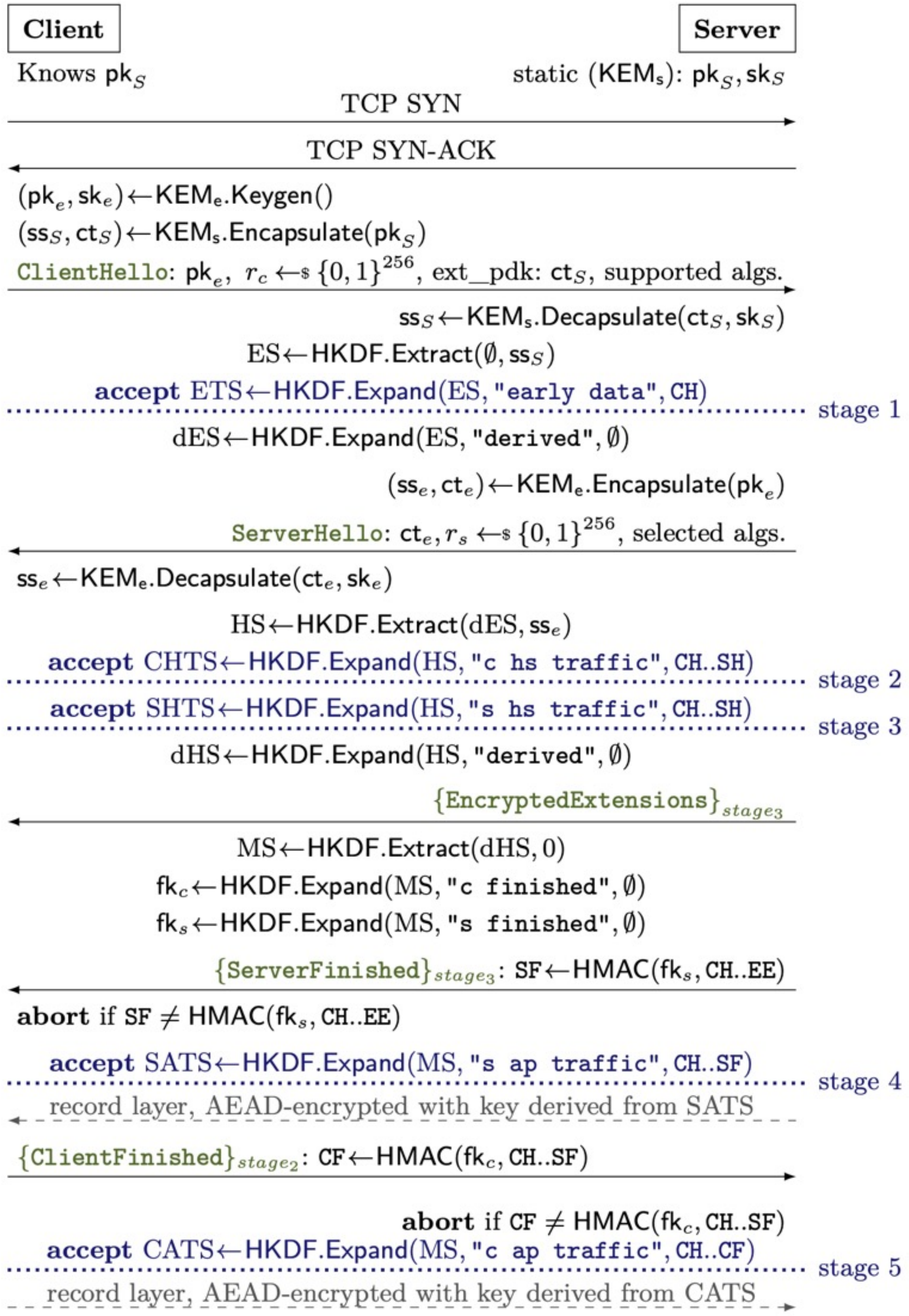

Client
Server
Knows pk_S
static ($\mathsf{KEM_s}$): $\mathsf{pk}_S, \mathsf{sk}_S$
$(\mathsf{pk}_e, \mathsf{sk}_e) \leftarrow \mathsf{KEM_e.Keygen}()$
$(\mathsf{ss}_S, \mathsf{ct}_S) \leftarrow \mathsf{KEM_e.Encapsulate}(\mathsf{pk}_S)$
$\mathsf{pk}_e, \mathsf{ct}_S$
$\mathsf{ss}_S \leftarrow \mathsf{KEM_s.Decapsulate}(\mathsf{ct}_S, \mathsf{sk}_S)$
$(\mathsf{ss}_e, \mathsf{ct}_e) \leftarrow \mathsf{KEM_e.Encapsulate}(\mathsf{pk}_e)$
ct_e
$\mathsf{ss}_e \leftarrow \mathsf{KEM_e.Decapsulate}(\mathsf{ct}_e, \mathsf{sk}_e)$
$K, K', K'', K''' \leftarrow \mathsf{KDF}(\mathsf{ss}_S \| \mathsf{ss}_e)$
AEAD_K(key confirmation)
$\mathsf{AEAD}_{K'}$(application data)
$\mathsf{AEAD}_{K''}$(key confirmation)
$\mathsf{AEAD}_{K'''}$(application data)

(a) Unilaterally authenticated


Client
Server
static ($\mathsf{KEM_c}$): $\mathsf{pk}_C, \mathsf{sk}_C$
Knows pk_S
static ($\mathsf{KEM_s}$): $\mathsf{pk}_S, \mathsf{sk}_S$
$(\mathsf{pk}_e, \mathsf{sk}_e) \leftarrow \mathsf{KEM_e.Keygen}()$
$(\mathsf{ss}_S, \mathsf{ct}_S) \leftarrow \mathsf{KEM_e.Encapsulate}(\mathsf{pk}_S)$
$K_S \leftarrow \mathsf{KDF}(\mathsf{ss}_S)$
$\mathsf{pk}_e, \mathsf{ct}_S, \mathsf{AEAD}_{K_S}(\mathsf{cert}[\mathsf{pk}_C])$
$\mathsf{ss}_S \leftarrow \mathsf{KEM_s.Decapsulate}(\mathsf{ct}_S, \mathsf{sk}_S)$
$(\mathsf{ss}_e, \mathsf{ct}_e) \leftarrow \mathsf{KEM_e.Encapsulate}(\mathsf{pk}_e)$
$(\mathsf{ss}_C, \mathsf{ct}_C) \leftarrow \mathsf{KEM_c.Encapsulate}(\mathsf{pk}_C)$
ct_e
$\mathsf{ss}_e \leftarrow \mathsf{KEM_e.Decapsulate}(\mathsf{ct}_e, \mathsf{sk}_e)$
$K_1 \leftarrow \mathsf{KDF}(\mathsf{ss}_S \| \mathsf{ss}_e)$
$\mathsf{AEAD}_{K_1}(\mathsf{ct}_C)$
$\mathsf{ss}_C \leftarrow \mathsf{KEM_c.Decapsulate}(\mathsf{ct}_C, \mathsf{sk}_C)$
$K_2, K_2', K_2'', K_2''' \leftarrow \mathsf{KDF}(\mathsf{ss}_S \| \mathsf{ss}_e \| \mathsf{ss}_C)$
AEAD_{K_2}(key confirmation)
$\mathsf{AEAD}_{K_2'}$(application data)
$\mathsf{AEAD}_{K_2''}$(key confirmation)
$\mathsf{AEAD}_{K_2'''}$(application data)

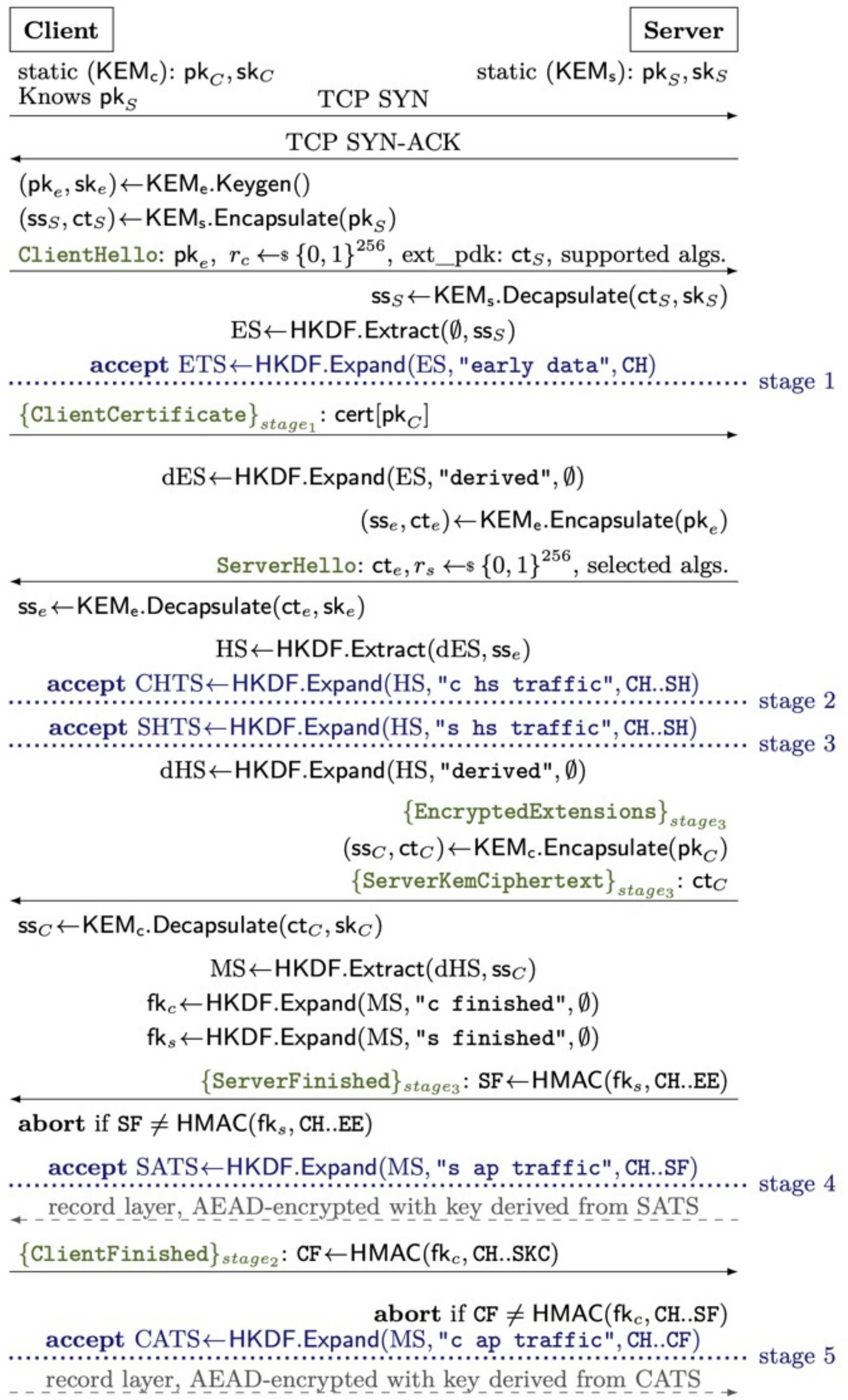
(b) With proactive client authentication

KEMTLS-PDK



Client
Server
Knows pk_S
static ($\mathsf{KEM_s}$): $\mathsf{pk}_S, \mathsf{sk}_S$
TCP SYN
TCP SYN-ACK
$(\mathsf{pk}_e, \mathsf{sk}_e) \leftarrow \mathsf{KEM_e.Keygen}()$
$(\mathsf{ss}_S, \mathsf{ct}_S) \leftarrow \mathsf{KEM_s.Encapsulate}(\mathsf{pk}_S)$
ClientHello: pk_e, $r_c \leftarrow_\$ \{0,1\}^{256}$, ext_pdk: ct_S, supported algs.
$\mathsf{ss}_S \leftarrow \mathsf{KEM_s.Decapsulate}(\mathsf{ct}_S, \mathsf{sk}_S)$
$\mathrm{ES} \leftarrow \mathsf{HKDF.Extract}(\emptyset, \mathsf{ss}_S)$
accept $\mathrm{ETS} \leftarrow \mathsf{HKDF.Expand}(\mathrm{ES}, \texttt{"early data"}, \mathtt{CH})$
stage 1
$\mathrm{dES} \leftarrow \mathsf{HKDF.Expand}(\mathrm{ES}, \texttt{"derived"}, \emptyset)$
$(\mathsf{ss}_e, \mathsf{ct}_e) \leftarrow \mathsf{KEM_e.Encapsulate}(\mathsf{pk}_e)$
ServerHello: $\mathsf{ct}_e, r_s \leftarrow_\$ \{0,1\}^{256}$, selected algs.
$\mathsf{ss}_e \leftarrow \mathsf{KEM_e.Decapsulate}(\mathsf{ct}_e, \mathsf{sk}_e)$
$\mathrm{HS} \leftarrow \mathsf{HKDF.Extract}(\mathrm{dES}, \mathsf{ss}_e)$
accept $\mathrm{CHTS} \leftarrow \mathsf{HKDF.Expand}(\mathrm{HS}, \texttt{"c hs traffic"}, \mathtt{CH..SH})$
stage 2
accept $\mathrm{SHTS} \leftarrow \mathsf{HKDF.Expand}(\mathrm{HS}, \texttt{"s hs traffic"}, \mathtt{CH..SH})$
stage 3
$\mathrm{dHS} \leftarrow \mathsf{HKDF.Expand}(\mathrm{HS}, \texttt{"derived"}, \emptyset)$
$\{\texttt{EncryptedExtensions}\}_{stage_3}$
$\mathrm{MS} \leftarrow \mathsf{HKDF.Extract}(\mathrm{dHS}, 0)$
$\mathsf{fk}_c \leftarrow \mathsf{HKDF.Expand}(\mathrm{MS}, \texttt{"c finished"}, \emptyset)$
$\mathsf{fk}_s \leftarrow \mathsf{HKDF.Expand}(\mathrm{MS}, \texttt{"s finished"}, \emptyset)$
$\{\texttt{ServerFinished}\}_{stage_3}$: $\mathtt{SF} \leftarrow \mathsf{HMAC}(\mathsf{fk}_s, \mathtt{CH..EE})$
abort if $\mathtt{SF} \neq \mathsf{HMAC}(\mathsf{fk}_s, \mathtt{CH..EE})$
accept $\mathrm{SATS} \leftarrow \mathsf{HKDF.Expand}(\mathrm{MS}, \texttt{"s ap traffic"}, \mathtt{CH..SF})$
stage 4
record layer, AEAD-encrypted with key derived from SATS
$\{\texttt{ClientFinished}\}_{stage_2}$: $\mathtt{CF} \leftarrow \mathsf{HMAC}(\mathsf{fk}_c, \mathtt{CH..SF})$
abort if $\mathtt{CF} \neq \mathsf{HMAC}(\mathsf{fk}_c, \mathtt{CH..SF})$
accept $\mathrm{CATS} \leftarrow \mathsf{HKDF.Expand}(\mathrm{MS}, \texttt{"c ap traffic"}, \mathtt{CH..CF})$
stage 5
record layer, AEAD-encrypted with key derived from CATS

KEMTLS-PDK with proactive client authentication



Client | **Server**

Client: static ($\mathsf{KEM_c}$): $\mathsf{pk}_C, \mathsf{sk}_C$; Knows pk_S

Server: static ($\mathsf{KEM_s}$): $\mathsf{pk}_S, \mathsf{sk}_S$

→ TCP SYN

← TCP SYN-ACK

$(\mathsf{pk}_e, \mathsf{sk}_e) \leftarrow \mathsf{KEM_e.Keygen}()$

$(\mathsf{ss}_S, \mathsf{ct}_S) \leftarrow \mathsf{KEM_s.Encapsulate}(\mathsf{pk}_S)$

→ ClientHello: pk_e, $r_c \leftarrow\!\$ \{0,1\}^{256}$, ext_pdk: ct_S, supported algs.

$\mathsf{ss}_S \leftarrow \mathsf{KEM_s.Decapsulate}(\mathsf{ct}_S, \mathsf{sk}_S)$

$\mathrm{ES} \leftarrow \mathsf{HKDF.Extract}(\emptyset, \mathsf{ss}_S)$

accept $\mathrm{ETS} \leftarrow \mathsf{HKDF.Expand}(\mathrm{ES}, \texttt{"early data"}, \texttt{CH})$ ······ stage 1

→ {ClientCertificate}$_{stage_1}$: $\mathsf{cert}[\mathsf{pk}_C]$

$\mathrm{dES} \leftarrow \mathsf{HKDF.Expand}(\mathrm{ES}, \texttt{"derived"}, \emptyset)$

$(\mathsf{ss}_e, \mathsf{ct}_e) \leftarrow \mathsf{KEM_e.Encapsulate}(\mathsf{pk}_e)$

← ServerHello: ct_e, $r_s \leftarrow\!\$ \{0,1\}^{256}$, selected algs.

$\mathsf{ss}_e \leftarrow \mathsf{KEM_e.Decapsulate}(\mathsf{ct}_e, \mathsf{sk}_e)$

$\mathrm{HS} \leftarrow \mathsf{HKDF.Extract}(\mathrm{dES}, \mathsf{ss}_e)$

accept $\mathrm{CHTS} \leftarrow \mathsf{HKDF.Expand}(\mathrm{HS}, \texttt{"c hs traffic"}, \texttt{CH..SH})$ ······ stage 2

accept $\mathrm{SHTS} \leftarrow \mathsf{HKDF.Expand}(\mathrm{HS}, \texttt{"s hs traffic"}, \texttt{CH..SH})$ ······ stage 3

$\mathrm{dHS} \leftarrow \mathsf{HKDF.Expand}(\mathrm{HS}, \texttt{"derived"}, \emptyset)$

{EncryptedExtensions}$_{stage_3}$

$(\mathsf{ss}_C, \mathsf{ct}_C) \leftarrow \mathsf{KEM_c.Encapsulate}(\mathsf{pk}_C)$

← {ServerKemCiphertext}$_{stage_3}$: ct_C

$\mathsf{ss}_C \leftarrow \mathsf{KEM_c.Decapsulate}(\mathsf{ct}_C, \mathsf{sk}_C)$

$\mathrm{MS} \leftarrow \mathsf{HKDF.Extract}(\mathrm{dHS}, \mathsf{ss}_C)$

$\mathsf{fk}_c \leftarrow \mathsf{HKDF.Expand}(\mathrm{MS}, \texttt{"c finished"}, \emptyset)$

$\mathsf{fk}_s \leftarrow \mathsf{HKDF.Expand}(\mathrm{MS}, \texttt{"s finished"}, \emptyset)$

← {ServerFinished}$_{stage_3}$: $\texttt{SF} \leftarrow \mathsf{HMAC}(\mathsf{fk}_s, \texttt{CH..EE})$

abort if $\texttt{SF} \neq \mathsf{HMAC}(\mathsf{fk}_s, \texttt{CH..EE})$

accept $\mathrm{SATS} \leftarrow \mathsf{HKDF.Expand}(\mathrm{MS}, \texttt{"s ap traffic"}, \texttt{CH..SF})$ ······ stage 4

← record layer, AEAD-encrypted with key derived from SATS

→ {ClientFinished}$_{stage_2}$: $\texttt{CF} \leftarrow \mathsf{HMAC}(\mathsf{fk}_c, \texttt{CH..SKC})$

abort if $\texttt{CF} \neq \mathsf{HMAC}(\mathsf{fk}_c, \texttt{CH..SF})$

accept $\mathrm{CATS} \leftarrow \mathsf{HKDF.Expand}(\mathrm{MS}, \texttt{"c ap traffic"}, \texttt{CH..CF})$ ······ stage 5

→ record layer, AEAD-encrypted with key derived from CATS

Security subtleties: forward secrecy

Does compromise of a party’s long-term key allow decryption of past sessions?

- **Weak forward secrecy 1:** adversary passive in the test stage
- **Weak forward secrecy 2:** adversary passive in the test stage or never corrupted peer’s long-term key
- **Forward secrecy:** adversary passive in the test stage or didn’t corrupt peer’s long-term key before acceptance

Lessons learned from formal verification

- Higher assurance in protocol design
- Captures potential interactions between all 4 protocol variants
- Exhibits difficulty trade-off in formal verification:
  granularity of protocol specification
  versus
  granularity of security properties

- Formal verification identified bugs in previous work:
  - Approach 1 identified minor bugs in original TLS 1.3 Tamarin model of [CHHSV]
  - Approach 2 identified minor bugs in security properties stated in original KEMTLS and KEMTLS-PDK papers
    - E.g. Wrong retroactive authentication stages or incorrect forward secrecy levels for some stages