TLS Crypto Seminar

February 21, 2019

Felix Günther
UC San Diego

based on joint work with Benjamin Dowling, Marc Fischlin, Sogol Mazaheri, Douglas Stebila

and discussions with many others

UC San Diego

DFG Deutsche Forschungsgemeinschaft
German Research Foundation

This Seminar, Part II

Part II TLS 1.3

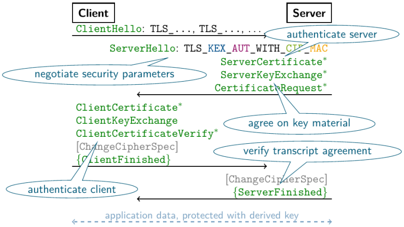
- The road to TLS 1.3 & its technical details.
- More analyses: understanding TLS 1.3's security and what drove design.

Schedule

Feb 21	**TLS 1.3** [TLS13] **& some security models** [FG17,GM17]	Felix
Feb 28	**Multiplexing channels** [PS18]	Vivek
Mar 7	**Symbolic Tamarin analysis** [CHH+17]	Baiyu
Mar 14	**Downgrade resilience** [BBF+16]	Ruth

The Road to TLS 1.3

Recap: TLS 1.2



Client **Server**

ClientHello: TLS_..., TLS_..., ...

ServerHello: TLS_KEX_AUT_WITH_CIP_MAC

ServerCertificate*

ServerKeyExchange*

CertificateRequest*

ClientCertificate*

ClientKeyExchange

ClientCertificateVerify*

[ChangeCipherSpec]

{ClientFinished}

[ChangeCipherSpec]

{ServerFinished}

application data, protected with derived key

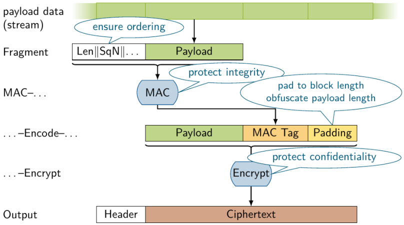
authenticate server

negotiate security parameters

agree on key material

verify transcript agreement

authenticate client



payload data (stream)

ensure ordering

Fragment: Len‖SqN‖... | Payload

protect integrity

MAC–...: MAC

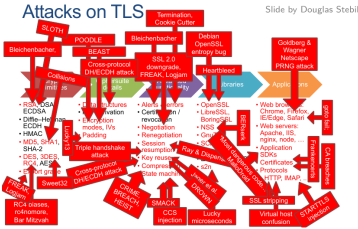
pad to block length obfuscate payload length

...–Encode–...: Payload | MAC Tag | Padding

protect confidentiality

...–Encrypt: Encrypt

Output: Header | Ciphertext



Attacks on TLS

Slide by Douglas Stebila

https://www.douglas.stebila.ca/research/presentations/tls-attacks/

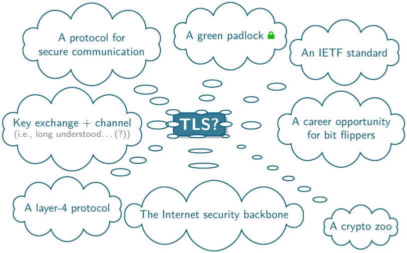
Cryptographic primitives
- RSA, DSA, ECDSA
- Diffie–Hellman, ECDH
- HMAC
- MD5, SHA1, SHA-2
- DES, 3DES, RC4, AES
- Export grade

Ciphersuite details
- Data structures
- Key derivation
- Encryption modes, IVs
- Padding

Advanced functionality
- Alerts & errors
- Certification / revocation
- Negotiation
- Renegotiation
- Session resumption
- Key reuse
- Compression
- State machine

Libraries
- OpenSSL
- LibreSSL, BoringSSL
- NSS
- GnuTLS
- SChannel
- s2n

Applications
- Web browsers: Chrome, Firefox, IE/Edge, Safari
- Web servers: Apache, IIS, nginx, node, ...
- Application SDKs
- Certificates
- Protocols HTTP, IMAP, ...



A protocol for secure communication

A green padlock 🔒

An IETF standard

Key exchange + channel (i.e., long understood...(?))

TLS?

A career opportunity for bit flippers

A layer-4 protocol

The Internet security backbone

A crypto zoo

TLS 1.3

A New Hope?

- IETF TLS WG begins in **early 2014** with developing new TLS 1.3 version

So… what would you change?

TLS 1.3

Design Goals

- **Clean up:** get rid of flawed and unused crypto & features
- **Improve latency:** for main handshake and repeated connections (while maintaining security)
- **Improve privacy:** hide as much of the handshake as possible
- **Continuity:** maintain interoperability with previous versions and support existing important use cases
- **Security Assurance (added later):** have supporting analyses for changes

TLS 1.3

Main changes (from TLS 1.2)

UC San Diego

Clean up

- removed **legacy and broken crypto**
  - ciphers: (3)DES, RC4, . . . , MtEE (CBC & generally) — only AEAD remains
  - hash functions: MD5, SHA1
  - authentication: Kerberos, RSA PKCS#1v1.5 key transport
  - custom (EC)DHE groups
- removed **broken features**
  - compression
  - renegotiation (but added key updates + late client auth)

quite some resistance from enterprises doing passive inspection

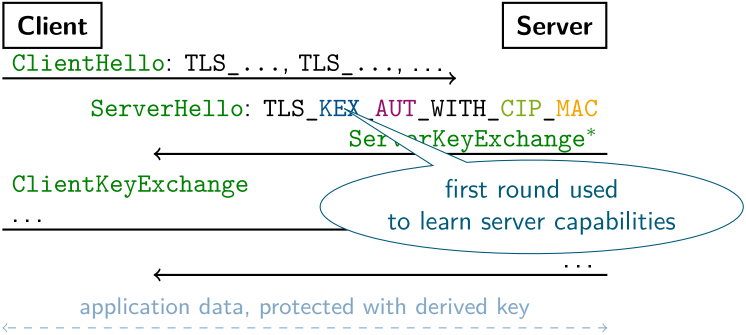
- removed **static RSA/DH**: public-key crypto = forward secrecy
- clean **key derivation** based on Extract-then-Expand HKDF
- **hardened negotiation** of version/cipher suite against downgrades

TLS 1.3

Main changes (from TLS 1.2)

Improve latency

- TLS 1.2 is slow: 2 round trips before client can send data



Client
Server
ClientHello: TLS_..., TLS_..., ...
ServerHello: TLS_KEX_AUT_WITH_CIP_MAC
ServerKeyExchange*
ClientKeyExchange
...
...
application data, protected with derived key

first round used to learn server capabilities

TLS 1.3

Main changes (from TLS 1.2)

Improve latency

- TLS 1.2 is slow: 2 round trips before client can send data

- TLS 1.3: **full handshake in 1 round trip**
  - feature reduction → we always do (EC)DHE
  - client speculatively sends several DH shares in supported groups
  - server picks one, replies with its share, and key can be already derived

- **0-RTT handshake** when resuming previous connection
  - client+server keep shared resumption secret (PSK)
  - client derives a key from that and can immediately encrypt data
  - but: 0-RTT *sacrifices* certain security properties (will come to that)

TLS 1.3

Main changes (from TLS 1.2)

Improve privacy

- TLS 1.2: complete handshake in the clear (incl. certificates, extensions)
- TLS 1.3: **encrypts almost all handshake messages**
  - derive separate key early to protect handshake messages
  - provides security against passive/active attackers (for server/client)

Continuity

- example: complex renegotiation only used for key updates + late client auth
  - just keep these features
- interoperability (idea): let `ClientHello` look like TLS <1.3
  - Well... we'll see.

TLS 1.3

Timeline, Proposals, and Security Analyses

2014	April	draft-00	copy of TLS 1.2
	July	draft-02	1-RTT, −custom DH, −compression −static RSA/DH, −non-AEAD
	October	draft-03	ECC in base standard
2015	January	draft-04	remove renegotiation
	March	draft-05	
		draft-dh	variant based on OPTLS

↳ [KW16] OPTLS: unified design. DH/PSK/0-RTT w static DH

↳ [DF**G**S15] draft-05/dh Analysis: first KE security result

	July	draft-07	merging OPTLS (partially): key schedule, HKDF, 0-RTT
	August	draft-08/9	deprecate MD5+SHA1, add RSA-PSS signatures

↳ [BL16] SLOTH: transcript collision attacks

↳ [JSS15] TLS 1.3 vs. PKCS#1v1.5 Encryption: still bad

STANDARD UNDER CONSTRUCTION

https://tools.ietf.org/html/draft-ietf-tls-tls13

TLS 1.3

Timeline, Proposals, and Security Analyses [cont'd]

UC San Diego

Ruth
Mar 14

2015 October draft-10

December draft-11 + downgrade protection, + late client auth, es

↳ [BBF+16] Downgrade Resilience: proposed harde

↳ [Kra16] Post-Handshake Client Auth: formal treatment

2016 February **TRON** (TLS 1.3 – Ready or Not?) @ NDSS 2016

↳ [DF**G**S16] draft-10 Analysis: updated KE security analysis

↳ [BMM+15] Record Protocol Analysis: via constructive crypto

↳ [BBDL+16] miTLS: towards a verified implementation

↳ [CHSM16] Tamarin Analysis: symbolic, identified attack

⋮

May draft-13 restructure key schedule, only PSK-based 0-RTT

↳ [F**G**17] 0-RTT Analysis: PSK- & DH-based, security limitations

"**TRON2**" TLS 1.3 Meetup @ IEEE S&P 2016

↳ discussing key schedule, 0-RTT, early implementation results

https://tools.ietf.org/html/draft-ietf-tls-tls13

TLS 1.3

Timeline, Proposals, and Security Analyses [cont'd]

UC San Diego

2016 Aug-Oct `draft-15--17` lots of discussion around 0-RTT

October `draft-18`

↳ [BBK17] ProVerif Analysis: tool-based formal analysis

↳ [DLFK+17] miTLS: verified Record Protocol implementation

2017 April **TLS:DIV** (Design, Implem. & Verif.) @ EuroS&P / Eurocrypt 2017

↳ status update & still discussing 0-RTT [M . .

July `draft-21` + comment on 0-RTT security & d mitigations

↳ [CHH+17] Tamarin Analysis: updated

November `draft-22` "Implement changes for improved middlebox penetration"

↳ [Ben18] TLS Ecosystem Woes: Why your Crypto isn't Real World yet

2018 March `draft-25` include record header in associated data of

↳ [PS18] Record Protocol Model: multiplexing chan

`draft-26--28` clarifications and cleanup

Baiyu
Mar 7

Vivek
Feb 28

https://tools.ietf.org/html/draft-ietf-tls-tls13

TLS 1.3

The ~~End~~ RFC!

2018

August 10 TLS 1.3 = RFC 8446

August 19 **Crypto Welcomes TLS 1.3** @ Crypto 2018

- **already in:** Firefox, Chrome, Cloudflare, Google, Facebook, OpenSSL, ...
  - as of Sep 2018: ~5% @ Firefox, 2nd @ Cloudflare, ~50% @ Facebook
- **strong interaction:** TLS WG ↔ researchers ↔ engineers
  - high-paced draft progress (29 drafts in 4 years ≈ one every 2nd month)
  - proactive rather than reactive standardization process (see [PM16])
- **vibrant research topic:** 20+ papers sharpening understanding and tools

TLS 1.3
Handshake & Some Analysis

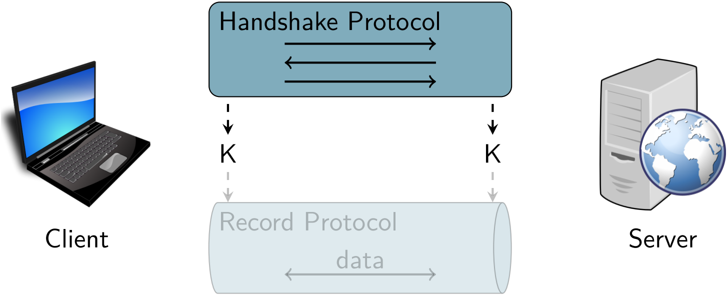
The TLS Protocol

Recap (again overly simplified)

UC San Diego

Handshake Protocol:

- negotiate security parameters (“cipher suite”)
- authenticate peers
- establish key material for data protection

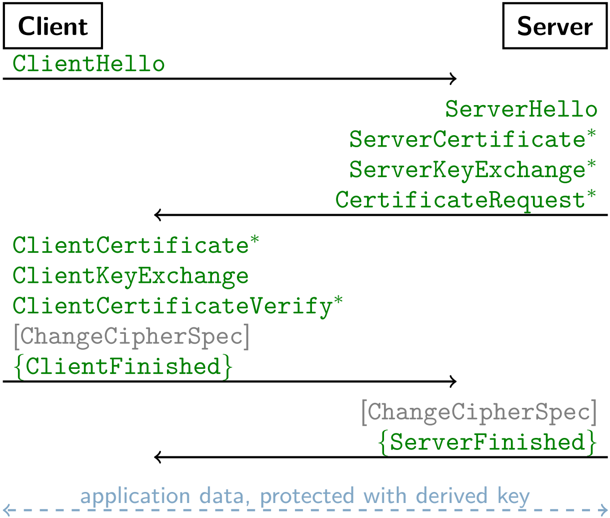


Record Protocol:

- protect data using key material from handshake
- ensuring confidentiality and integrity

The TLS Protocol

Recap: Handshake Protocol Structure up to TLS 1.2



Client

Server

ClientHello

ServerHello
ServerCertificate*
ServerKeyExchange*
CertificateRequest*

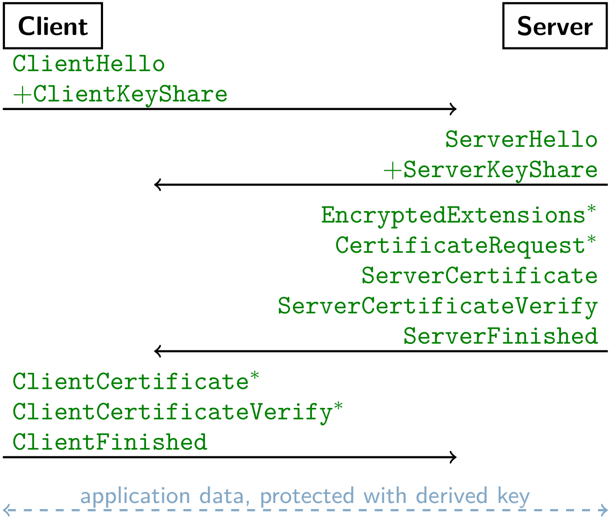
ClientCertificate*
ClientKeyExchange
ClientCertificateVerify*
[ChangeCipherSpec]
{ClientFinished}

[ChangeCipherSpec]
{ServerFinished}

application data, protected with derived key

The TLS 1.3 Handshake

Full (EC)DHE Mode



Client

Server

ClientHello
+ClientKeyShare

ServerHello
+ServerKeyShare

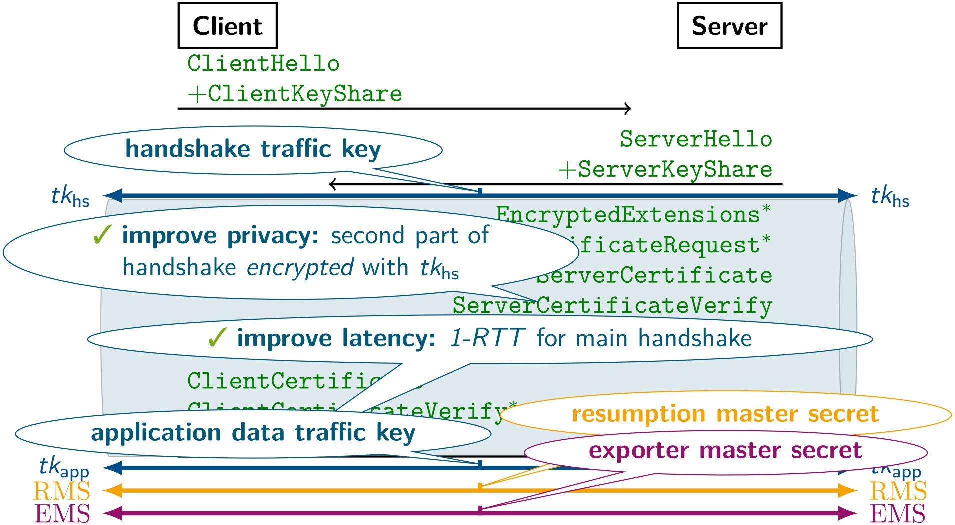
EncryptedExtensions*
CertificateRequest*
ServerCertificate
ServerCertificateVerify
ServerFinished

ClientCertificate*
ClientCertificateVerify*
ClientFinished

application data, protected with derived key

The TLS 1.3 Handshake

Full (EC)DHE Mode



Client

Server

ClientHello
+ClientKeyShare

ServerHello
+ServerKeyShare

handshake traffic key

tk_{hs} tk_{hs}

EncryptedExtensions*
ificateRequest*
erverCertificate
ServerCertificateVerify

✓ **improve privacy:** second part of handshake *encrypted* with tk_{hs}

✓ **improve latency:** *1-RTT* for main handshake

ClientCertific e
ClientCerti cateVerify*

resumption master secret

application data traffic key

exporter master secret

tk_{app} tk_{app}

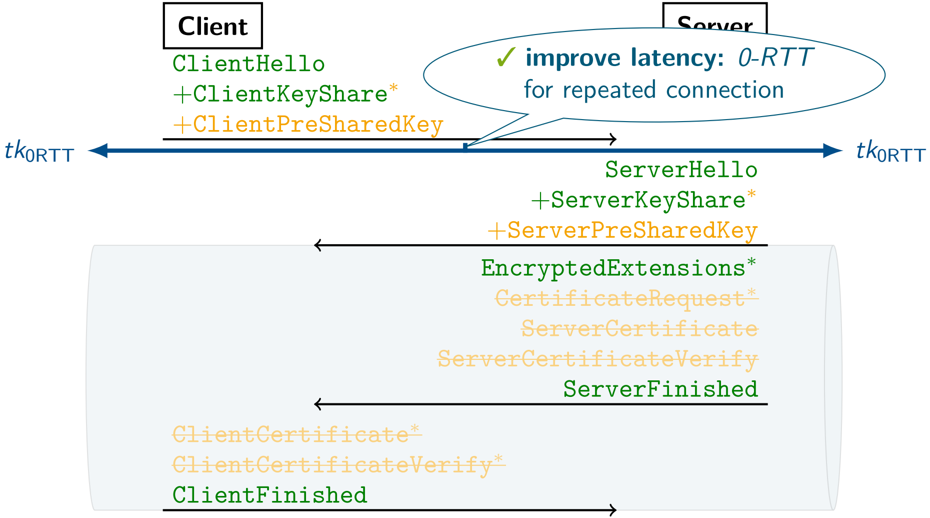
RMS RMS

EMS EMS

The TLS 1.3 Handshake

PSK / PSK-(EC)DHE Resumption Mode

UC San Diego



Client

Server

✓ **improve latency:** *0-RTT*
for repeated connection

ClientHello
+ClientKeyShare*
+ClientPreSharedKey

tk_{0RTT} tk_{0RTT}

ServerHello
+ServerKeyShare*
+ServerPreSharedKey

EncryptedExtensions*
~~CertificateRequest*~~
~~ServerCertificate~~
~~ServerCertificateVerify~~
ServerFinished

~~ClientCertificate*~~
~~ClientCertificateVerify*~~
ClientFinished

The TLS 1.3 Handshake

0.5-RTT and Post-Handshake Messages

Additional features (which we won't cover here...):

- **0.5-RTT**
  - server can already send data after its `Finished` message
  - client not yet authenticated, but can be done retroactively [Kra16]
- **Post-Handshake Client Authentication**
  - server can ask client to authenticate even after handshake is over
  - captures renegotiation functionality from $\leq$ TLS 1.2
  - again gives retroactive authentication [Kra16]
- **Key Updates**
  - both sides can initiate an update of the traffic key (post-handshake)
  - next key is then derived from master secret in forward-secure manner [**G**M17]

TLS 1.3 Handshake Security

- So: What kind of security do we expect for the TLS 1.3 handshake?

- **secure key exchange** (à la [BR94])

- here: **provable, game-based, reductionist security**
  - allows us to capture detailed cryptographic computations
  - get precise security bounds & crypto design recommendations

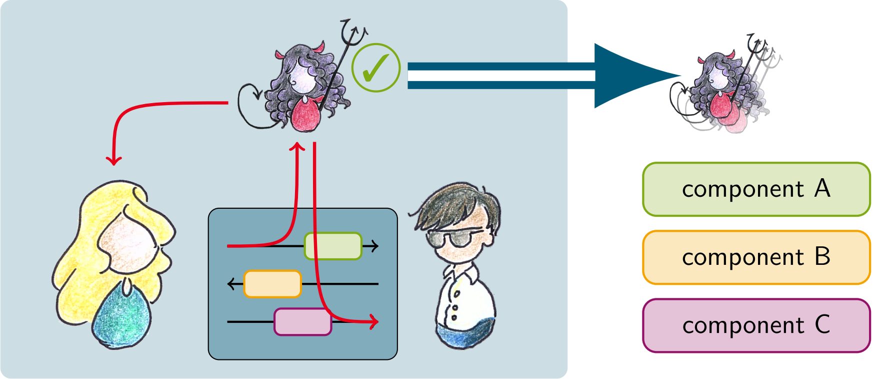
  - due to all the crypto details, security proofs can get complex
  - to handle complexity, we focus on one handshake mode at a time
  - and only look at the "cryptographic core"

  - symbolic analysis tools are better in analyzing interaction across modes
  - though somewhat coarser on the crypto details

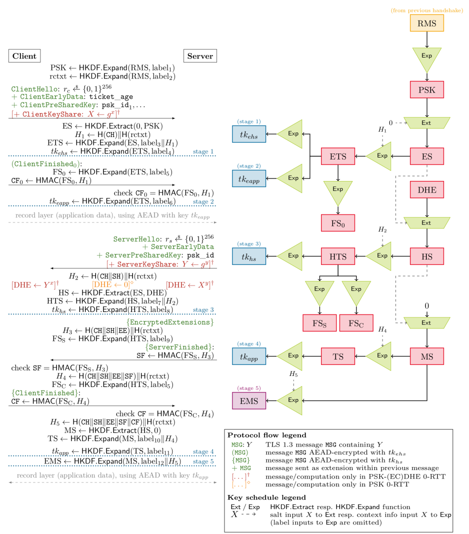
  - to be sure the actual code is secure, you need a verified implementation

Cryptographic Security Models and the Provable Security Approach

1. describe abstract protocol
2. define security
3. reduce to assumptions



TLS 1.3 Handshake as an Abstract Protocol



Client **Server**

$\mathrm{PSK} \leftarrow \mathsf{HKDF.Expand}(\mathrm{RMS}, \mathrm{label}_1)$
$\mathrm{rctxt} \leftarrow \mathsf{HKDF.Expand}(\mathrm{RMS}, \mathrm{label}_2)$

ClientHello: $r_c \xleftarrow{\$} \{0,1\}^{256}$
+ ClientEarlyData: ticket_age
+ ClientPreSharedKey: psk_id$_1, \ldots$
[+ ClientKeyShare: $X \leftarrow g^x]^\dagger$

$\mathrm{ES} \leftarrow \mathsf{HKDF.Extract}(0, \mathrm{PSK})$
$H_1 \leftarrow \mathsf{H}(\mathtt{CH}) \| \mathsf{H}(\mathrm{rctxt})$
$\mathrm{ETS} \leftarrow \mathsf{HKDF.Expand}(\mathrm{ES}, \mathrm{label}_3 \| H_1)$
$tk_{ehs} \leftarrow \mathsf{HKDF.Expand}(\mathrm{ETS}, \mathrm{label}_4)$ stage 1

(ClientFinished$_0$):
$\mathrm{FS}_0 \leftarrow \mathsf{HKDF.Expand}(\mathrm{ETS}, \mathrm{label}_5)$
$\mathtt{CF}_0 \leftarrow \mathsf{HMAC}(\mathrm{FS}_0, H_1)$

check $\mathtt{CF}_0 = \mathsf{HMAC}(\mathrm{FS}_0, H_1)$
$tk_{eapp} \leftarrow \mathsf{HKDF.Expand}(\mathrm{ETS}, \mathrm{label}_6)$ stage 2

record layer (application data), using AEAD with key tk_{eapp}

ServerHello: $r_s \xleftarrow{\$} \{0,1\}^{256}$
+ ServerEarlyData
+ ServerPreSharedKey: psk_id
[+ ServerKeyShare: $Y \leftarrow g^y]^\dagger$

$H_2 \leftarrow \mathsf{H}(\mathtt{CH} \| \mathtt{SH}) \| \mathsf{H}(\mathrm{rctxt})$
$[\mathrm{DHE} \leftarrow Y^x]^\dagger$ $[\mathrm{DHE} \leftarrow 0]^\diamond$ $[\mathrm{DHE} \leftarrow X^y]^\dagger$
$\mathrm{HS} \leftarrow \mathsf{HKDF.Extract}(\mathrm{ES}, \mathrm{DHE})$
$\mathrm{HTS} \leftarrow \mathsf{HKDF.Expand}(\mathrm{HS}, \mathrm{label}_7 \| H_2)$
$tk_{hs} \leftarrow \mathsf{HKDF.Expand}(\mathrm{HTS}, \mathrm{label}_8)$ stage 3

{EncryptedExtensions}
$H_3 \leftarrow \mathsf{H}(\mathtt{CH} \| \mathtt{SH} \| \mathtt{EE}) \| \mathsf{H}(\mathrm{rctxt})$
$\mathrm{FS_S} \leftarrow \mathsf{HKDF.Expand}(\mathrm{HTS}, \mathrm{label}_9)$
{ServerFinished}:
$\mathtt{SF} \leftarrow \mathsf{HMAC}(\mathrm{FS_S}, H_3)$

check $\mathtt{SF} = \mathsf{HMAC}(\mathrm{FS_S}, H_3)$
$H_4 \leftarrow \mathsf{H}(\mathtt{CH} \| \mathtt{SH} \| \mathtt{EE} \| \mathtt{SF}) \| \mathsf{H}(\mathrm{rctxt})$
$\mathrm{FS_C} \leftarrow \mathsf{HKDF.Expand}(\mathrm{HTS}, \mathrm{label}_5)$
{ClientFinished}:
$\mathtt{CF} \leftarrow \mathsf{HMAC}(\mathrm{FS_C}, H_4)$

check $\mathtt{CF} = \mathsf{HMAC}(\mathrm{FS_C}, H_4)$
$H_5 \leftarrow \mathsf{H}(\mathtt{CH} \| \mathtt{SH} \| \mathtt{EE} \| \mathtt{SF} \| \mathtt{CF}) \| \mathsf{H}(\mathrm{rctxt})$
$\mathrm{MS} \leftarrow \mathsf{HKDF.Extract}(\mathrm{HS}, 0)$
$\mathrm{TS} \leftarrow \mathsf{HKDF.Expand}(\mathrm{MS}, \mathrm{label}_{10} \| H_4)$

$tk_{app} \leftarrow \mathsf{HKDF.Expand}(\mathrm{TS}, \mathrm{label}_{11})$ stage 4
$\mathrm{EMS} \leftarrow \mathsf{HKDF.Expand}(\mathrm{MS}, \mathrm{label}_{12} \| H_5)$ stage 5

record layer (application data), using AEAD with key tk_{app}

Protocol flow legend

MSG: Y	TLS 1.3 message MSG containing Y
(MSG)	message MSG AEAD-encrypted with tk_{ehs}
{MSG}	message MSG AEAD-encrypted with tk_{hs}
+ MSG	message sent as extension within previous message
$[\ldots]^\dagger$	message/computation only in PSK-(EC)DHE 0-RTT
$[\ldots]^\diamond$	message/computation only in PSK 0-RTT

Key schedule legend

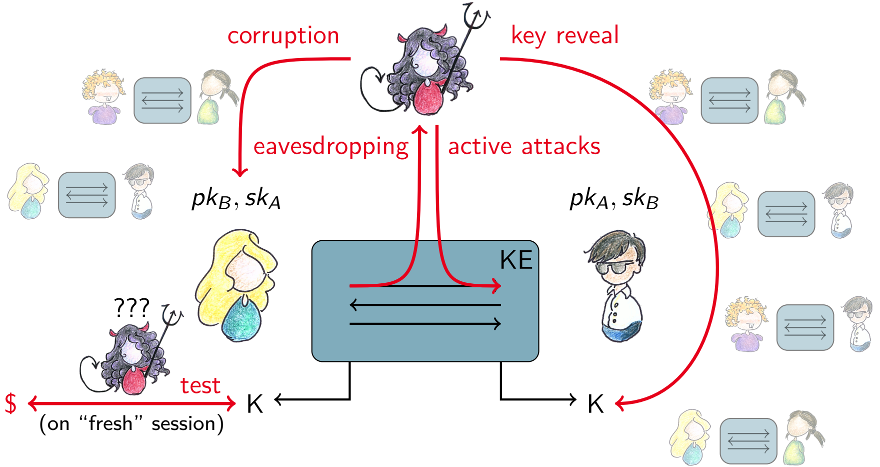
Ext / Exp	HKDF.Extract resp. HKDF.Expand function
$X \dashrightarrow$	salt input X to Ext resp. context info input X to Exp (label inputs to Exp are omitted)

Figure 7.2: The TLS 1.3 `draft-14` PSK and PSK-(EC)DHE 0-RTT handshake protocols (left) and key schedule (right).

can be done,
but let's skip that
for now. . .

Key Exchange Security

Recap: Classical Definition [BR94]

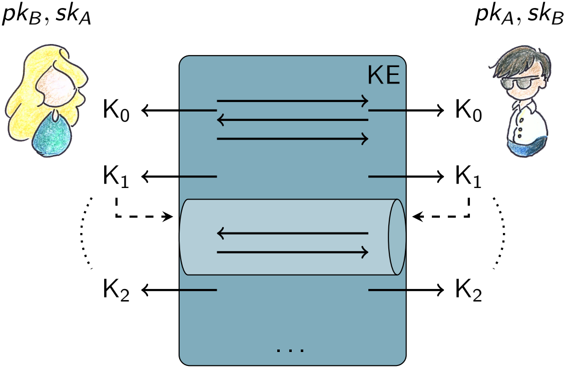


Key Exchange Security

Novel Designs

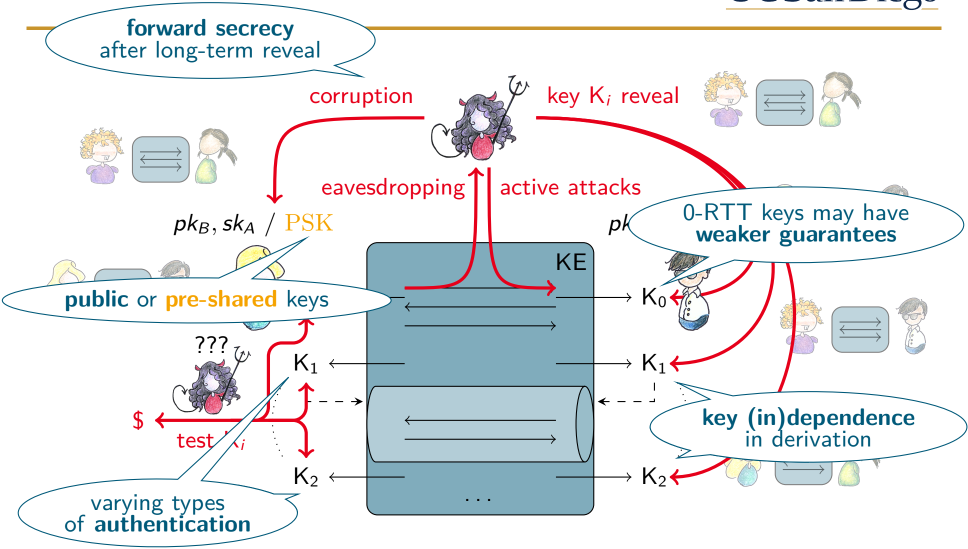
- go beyond what classical models can capture
- e.g., Google QUIC, **TLS 1.3**, Signal, . . .

- multiple keys
- potential dependencies
- mixed usage within KE
- low-latency / 0-RTT



Key Exchange Security

Multi-Stage Key Exchange



forward secrecy after long-term reveal
corruption
key K_i reveal
eavesdropping
active attacks
pk_B, sk_A / PSK
pk
0-RTT keys may have **weaker guarantees**
KE
K_0
public or **pre-shared** keys
???
K_1
K_1
$
test K_i
key (in)dependence in derivation
K_2
K_2
. . .
varying types of **authentication**

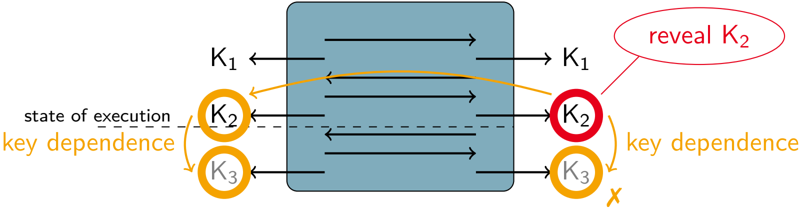
[FG14], [DFGS15], [DFGS16], [FG17], [G18]

Multi-Stage Key Exchange

Extended Properties

(In)Dependence of Session Keys

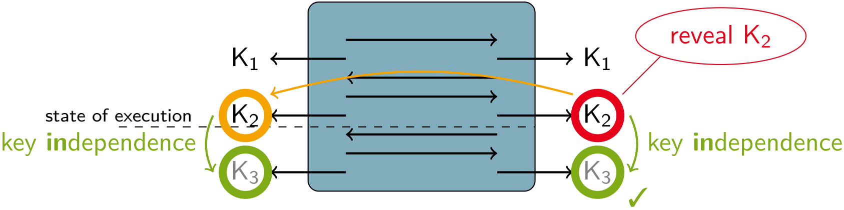
- multi-stage ⇒ derived keys might build upon each other
- **key-dependent**: reveal K_i before K_{i+1} accepted *may compromise* K_{i+1}



(In)Dependence of Session Keys

- multi-stage ⇒ derived keys might build upon each other

- **key-dependent**: reveal K_i before K_{i+1} accepted *may compromise* K_{i+1}
- **key-independent**: reveal of any K_i *never harms* any other K_{i+1}



Multi-Stage Key Exchange

Extended Properties

Forward Secrecy

- multi-stage ⇒ forward secrecy might kick in only at some stage j
- take this into account when handling corruptions

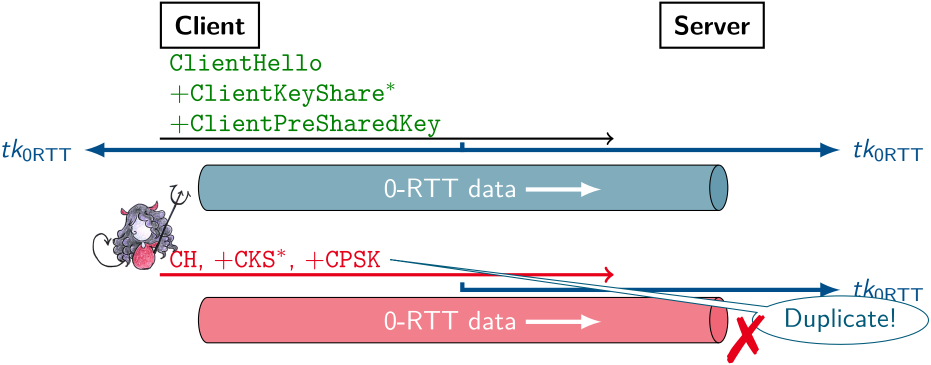
- **non-forward-secret**: all session keys compromised by corruption
- **stage-*j*-forward-secret**: accepted keys at stages $i \geq j$ remain secure

Levels of Authentication

- different stages/keys may hold different authentication properties
  - **unauthenticated** (no-one)
  - **unilateral** authentication (server-only)
  - **mutual** authentication (both)

- different types may run concurrently (TLS: adaptive client authentication)

0-RTT and Replays

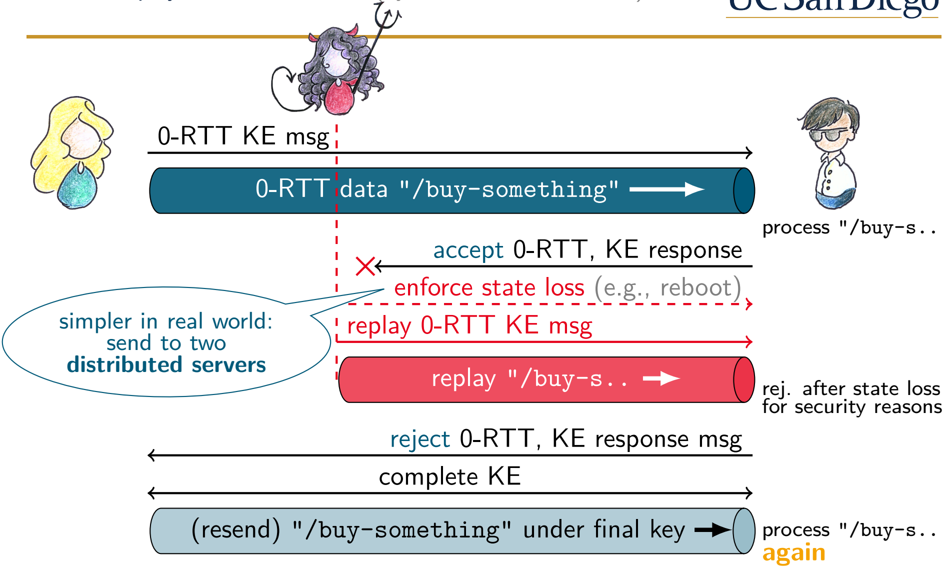


- allows client to send data without waiting for server reply
- but without server input, how does server know the request is fresh?

- adversary can replay `ClientHello` together with 0-RTT data
- idea: remember `ClientHello` identifier and reject duplicates

0-RTT and Replays

Generic Replay Attack on 0-RTT (by Daniel Kahn Gillmore)



0-RTT KE msg

0-RTT data "/buy-something"

process "/buy-s..

accept 0-RTT, KE response

enforce state loss (e.g., reboot)

replay 0-RTT KE msg

replay "/buy-s..

rej. after state loss
for security reasons

simpler in real world:
send to two
distributed servers

reject 0-RTT, KE response msg

complete KE

(resend) "/buy-something" under final key

process "/buy-s..
again

0-RTT and Replays

TLS 1.3's Take on Replays

TLS does not provide inherent replay protection for 0-RTT data.

[Simple duplicates] can be prevented by sharing state to guarantee that the 0-RTT data is accepted at most once.

Servers SHOULD provide that level of replay safety by implementing one of the methods described in this section [. . .] [RFC 8446, Section 8]

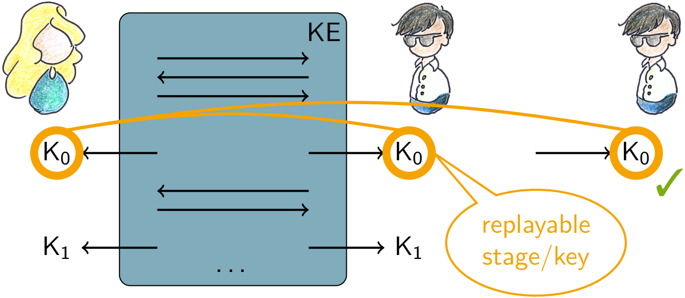
- **suggested mechanisms**
  - single-use tickets: allow each RMS to be used only once (simplest)
  - `ClientHello` recording: reject by unique identifier
  - freshness checks: reject based on `ClientHello` time
- "SHOULD" → treat 0-RTT keys generally as **replayable in analysis**
  - so, what security remains?

Multi-Stage Key Exchange

Extended Properties

Replays

- some stages' keys may be **replayable**
- may be **accepted multiple times**, this shouldn't count as an attack
- but should **still remain secret** from adversary even if replayed



The TLS 1.3 Handshake

draft-14 PSK-(EC)DHE 0-RTT

UC San Diego

Client **Server**

CH: $r_c \leftarrow_{\$} \{0,1\}^{256}$, g^x, psk_id →

HKDF(PSK, H(CH)) tk_{0RTT}

← SH: $r_s \leftarrow_{\$} \{0,1\}^{256}$, g^y, psk_id

DHE ← g^{xy}

HS ← HKDF(PSK, DHE)

HKDF(HS, H(CH‖SH)) tk_{hs}

{EncryptedExtensions}

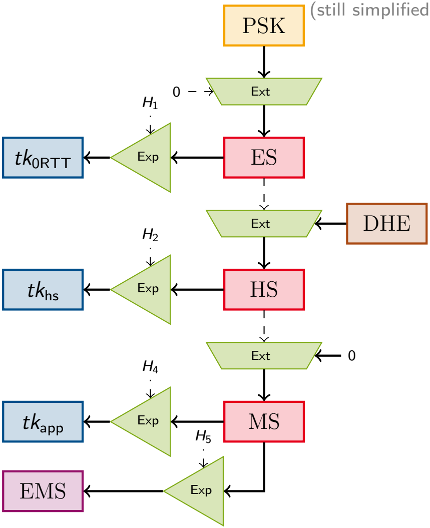
← {SF}: HMAC($\mathrm{FS_S}$, H(CH‖SH‖EE))

{CF}: HMAC($\mathrm{FS_C}$, H(CH‖ ... ‖SF)) →

HKDF(MS, H(CH‖ ... ‖SF)) tk_{app}

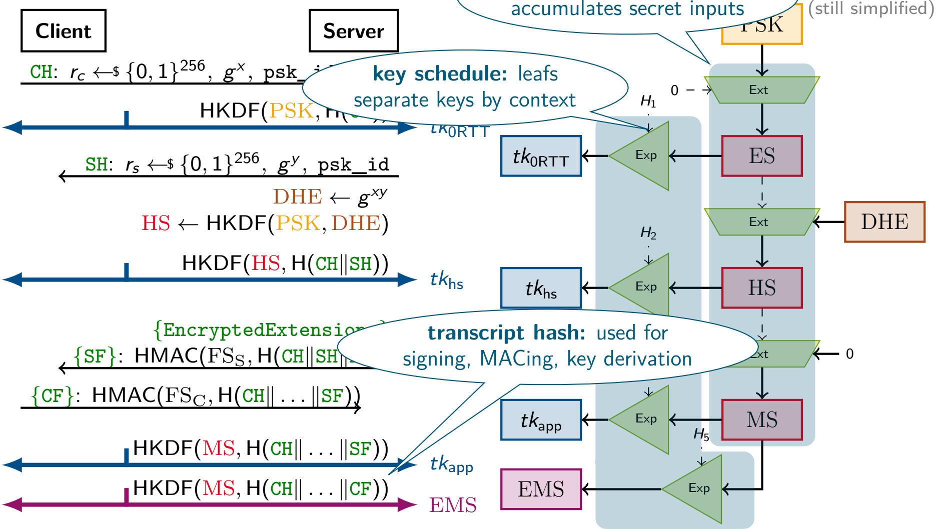
HKDF(MS, H(CH‖ ... ‖CF)) EMS

(still simplified)



The TLS 1.3 Handshake

draft-14 PSK-(EC)DHE 0-RTT


Client
Server
CH: $r_c \leftarrow_{\$} \{0,1\}^{256}$, g^x, psk_i
HKDF(PSK, H(C
tk_{0RTT}
SH: $r_s \leftarrow_{\$} \{0,1\}^{256}$, g^y, psk_id
DHE $\leftarrow g^{xy}$
HS $\leftarrow$ HKDF(PSK, DHE)
HKDF(HS, H(CH$\|$SH))
tk_{hs}
{EncryptedExtensio
{SF}: HMAC($\mathrm{FS_S}$, H(CH$\|$SH$\|$
{CF}: HMAC($\mathrm{FS_C}$, H(CH$\|\ldots\|$SF))
HKDF(MS, H(CH$\|\ldots\|$SF))
tk_{app}
HKDF(MS, H(CH$\|\ldots\|$CF))
EMS
(still simplified)
PSK
0
Ext
H_1
ES
Exp
tk_{0RTT}
Ext
DHE
H_2
HS
Exp
tk_{hs}
Ext
0
Exp
tk_{app}
MS
H_5
Exp
EMS

key schedule: core accumulates secret inputs

key schedule: leafs separate keys by context

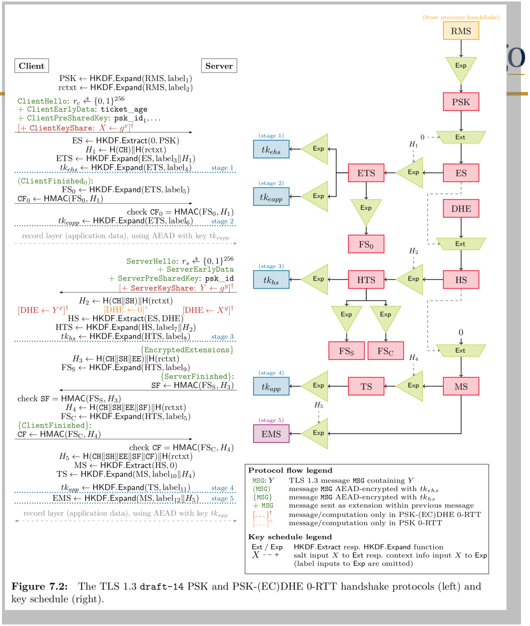
transcript hash: used for signing, MACing, key derivation

The TLS 1.3 Handshake

draft-14 PSK-(EC)DHE 0-RTT

The full details. . .

- more intermediate keys (e.g., deriving MAC keys)
- a fifth key tk_{0hs} for 0-RTT handshake encryption (got dropped again later)
- and more. . .



Client **Server**

$\mathrm{PSK} \leftarrow \mathsf{HKDF.Expand}(\mathrm{RMS}, \mathrm{label}_1)$
$\mathrm{rctxt} \leftarrow \mathsf{HKDF.Expand}(\mathrm{RMS}, \mathrm{label}_2)$

`ClientHello`: $r_c \xleftarrow{\$} \{0,1\}^{256}$
+ `ClientEarlyData`: `ticket_age`
+ `ClientPreSharedKey`: `psk_id`$_1, \ldots$
[+ `ClientKeyShare`: $X \leftarrow g^x]^\dagger$

$\mathrm{ES} \leftarrow \mathsf{HKDF.Extract}(0, \mathrm{PSK})$
$H_1 \leftarrow \mathsf{H}(\mathtt{CH}) \| \mathsf{H}(\mathrm{rctxt})$
$\mathrm{ETS} \leftarrow \mathsf{HKDF.Expand}(\mathrm{ES}, \mathrm{label}_3 \| H_1)$
$tk_{ehs} \leftarrow \mathsf{HKDF.Expand}(\mathrm{ETS}, \mathrm{label}_4)$ stage 1

(`ClientFinished`$_0$):
$\mathrm{FS}_0 \leftarrow \mathsf{HKDF.Expand}(\mathrm{ETS}, \mathrm{label}_5)$
$\mathtt{CF}_0 \leftarrow \mathsf{HMAC}(\mathrm{FS}_0, H_1)$

check $\mathtt{CF}_0 = \mathsf{HMAC}(\mathrm{FS}_0, H_1)$
$tk_{eapp} \leftarrow \mathsf{HKDF.Expand}(\mathrm{ETS}, \mathrm{label}_6)$ stage 2

record layer (application data), using AEAD with key tk_{eapp}

`ServerHello`: $r_s \xleftarrow{\$} \{0,1\}^{256}$
+ `ServerEarlyData`
+ `ServerPreSharedKey`: `psk_id`
[+ `ServerKeyShare`: $Y \leftarrow g^y]^\dagger$

$H_2 \leftarrow \mathsf{H}(\mathtt{CH}\|\mathtt{SH}) \| \mathsf{H}(\mathrm{rctxt})$
$[\mathrm{DHE} \leftarrow Y^x]^\dagger$ $[\mathrm{DHE} \leftarrow 0]^\diamond$ $[\mathrm{DHE} \leftarrow X^y]^\dagger$
$\mathrm{HS} \leftarrow \mathsf{HKDF.Extract}(\mathrm{ES}, \mathrm{DHE})$
$\mathrm{HTS} \leftarrow \mathsf{HKDF.Expand}(\mathrm{HS}, \mathrm{label}_7 \| H_2)$
$tk_{hs} \leftarrow \mathsf{HKDF.Expand}(\mathrm{HTS}, \mathrm{label}_8)$ stage 3

{`EncryptedExtensions`}
$H_3 \leftarrow \mathsf{H}(\mathtt{CH}\|\mathtt{SH}\|\mathtt{EE}) \| \mathsf{H}(\mathrm{rctxt})$
$\mathrm{FS_S} \leftarrow \mathsf{HKDF.Expand}(\mathrm{HTS}, \mathrm{label}_9)$
{`ServerFinished`}:
$\mathtt{SF} \leftarrow \mathsf{HMAC}(\mathrm{FS_S}, H_3)$

check $\mathtt{SF} = \mathsf{HMAC}(\mathrm{FS_S}, H_3)$
$H_4 \leftarrow \mathsf{H}(\mathtt{CH}\|\mathtt{SH}\|\mathtt{EE}\|\mathtt{SF}) \| \mathsf{H}(\mathrm{rctxt})$
$\mathrm{FS_C} \leftarrow \mathsf{HKDF.Expand}(\mathrm{HTS}, \mathrm{label}_5)$
{`ClientFinished`}:
$\mathtt{CF} \leftarrow \mathsf{HMAC}(\mathrm{FS_C}, H_4)$

check $\mathtt{CF} = \mathsf{HMAC}(\mathrm{FS_C}, H_4)$
$H_5 \leftarrow \mathsf{H}(\mathtt{CH}\|\mathtt{SH}\|\mathtt{EE}\|\mathtt{SF}\|\mathtt{CF}) \| \mathsf{H}(\mathrm{rctxt})$
$\mathrm{MS} \leftarrow \mathsf{HKDF.Extract}(\mathrm{HS}, 0)$
$\mathrm{TS} \leftarrow \mathsf{HKDF.Expand}(\mathrm{MS}, \mathrm{label}_{10} \| H_4)$

$tk_{app} \leftarrow \mathsf{HKDF.Expand}(\mathrm{TS}, \mathrm{label}_{11})$ stage 4
$\mathrm{EMS} \leftarrow \mathsf{HKDF.Expand}(\mathrm{MS}, \mathrm{label}_{12} \| H_5)$ stage 5

record layer (application data), using AEAD with key tk_{app}

Protocol flow legend

`MSG`: Y	TLS 1.3 message `MSG` containing Y
(`MSG`)	message `MSG` AEAD-encrypted with tk_{ehs}
{`MSG`}	message `MSG` AEAD-encrypted with tk_{hs}
\+ `MSG`	message sent as extension within previous message
$[\ldots]^\dagger$	message/computation only in PSK-(EC)DHE 0-RTT
$[\ldots]^\diamond$	message/computation only in PSK 0-RTT

Key schedule legend

Ext / Exp	HKDF.Extract resp. HKDF.Expand function
X - - →	salt input X to Ext resp. context info input X to Exp (label inputs to Exp are omitted)

Figure 7.2: The TLS 1.3 `draft-14` PSK and PSK-(EC)DHE 0-RTT handshake protocols (left) and key schedule (right).

TLS 1.3 Handshake Security

draft-14 PSK-(EC)DHE 0-RTT as Multi-Stage KE [FG17]

The **TLS 1.3 PSK-(EC)DHE 0-RTT** handshake provides

- random-looking secret keys (tk_{0hs}, tk_{0RTT}, tk_{hs}, tk_{app}, EMS)
- forward secrecy for non–0-RTT keys
- mutual authentication wrt. PSK
- key independence
- replayable 0-RTT keys

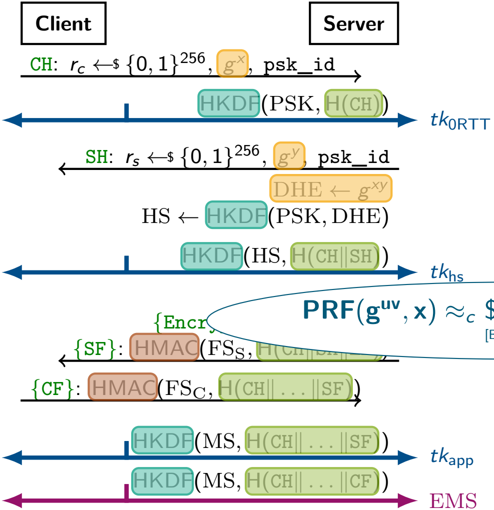
assuming . . .

Theorem 7.4. *The TLS 1.3* `draft-14` *PSK-(EC)DHE 0-RTT handshake is* **Multi-Stage-secure** *in a* ***key-independent*** *and* ***stage-3-forward-secret*** *manner with properties* (M, **AUTH**, USE, **REPLAY**).

$$
\begin{aligned}
\mathsf{Adv}^{\mathsf{Multi\text{-}Stage},\mathcal{D}}_{\mathtt{draft\text{-}14\text{-}PSK\text{-}(EC)DHE\text{-}0RTT},\mathcal{A}} \leq 5n_s \cdot \Bigg(& \mathsf{Adv}^{\mathsf{COLL}}_{\mathsf{H},\mathcal{B}_1} \\
+ n_p \cdot \Big(& \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathtt{HKDF.Expand},\mathcal{B}_2} + \mathsf{Adv}^{\mathsf{HMAC(0,\$)\text{-}\$}}_{\mathsf{HMAC},\mathcal{B}_3} \\
& + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HMAC},\mathcal{B}_4} + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathtt{HKDF.Expand},\mathcal{B}_5} \Big) \\
+ n_s \cdot n_p \cdot \Big(& \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathtt{HKDF.Expand},\mathcal{B}_6} + \mathsf{Adv}^{\mathsf{HMAC(0,\$)\text{-}\$}}_{\mathsf{HMAC},\mathcal{B}_7} \\
& + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HMAC},\mathcal{B}_8} + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HMAC},\mathcal{B}_9} \\
& + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathtt{HKDF.Expand},\mathcal{B}_{10}} + \mathsf{Adv}^{\mathsf{EUF\text{-}CMA}}_{\mathsf{HMAC},\mathcal{B}_{11}} \Big) \\
+ n_s \cdot n_p \cdot \Big(& \mathsf{Adv}^{\mathsf{snPRF\text{-}ODH}}_{\mathtt{HKDF.Extract},\mathbb{G},\mathcal{B}_{12}} + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HMAC},\mathcal{B}_{13}} \\
& + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathtt{HKDF.Expand},\mathcal{B}_{14}} + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathtt{HKDF.Expand},\mathcal{B}_{15}} \\
& + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathtt{HKDF.Expand},\mathcal{B}_{16}} \Big) \Bigg).
\end{aligned}
$$

TLS 1.3 Handshake Security

draft-14 PSK-(EC)DHE 0-RTT as Multi-Stage KE [FG17]


Client
Server
CH: $r_c \leftarrow_\$ \{0,1\}^{256}$, g^x, psk_id
HKDF(PSK, H(CH))
tk_{0RTT}
SH: $r_s \leftarrow_\$ \{0,1\}^{256}$, g^y, psk_id
DHE ← g^{xy}
HS ← HKDF(PSK, DHE)
HKDF(HS, H(CH‖SH))
tk_{hs}
{Encr
{SF}: HMAC(FS$_S$, H(CH‖SH‖
{CF}: HMAC(FS$_C$, H(CH‖ . . . ‖SF))
HKDF(MS, H(CH‖ . . . ‖SF))
tk_{app}
HKDF(MS, H(CH‖ . . . ‖CF))
EMS

Theorem 7.4. *The TLS 1.3* draft-14 *PSK-(EC)DHE 0-RTT handshake is* **Multi-Stage-secure** *in a* ***key-independent*** *and* ***stage-3-forward-secret*** *manner with properties* (M, **AUTH**, USE, **REPLAY**).

$$\mathsf{Adv}^{\mathsf{Multi\text{-}Stage},\mathcal{D}}_{\mathtt{draft\text{-}14\text{-}PSK\text{-}(EC)DHE\text{-}0RTT},\mathcal{A}} \leq 5n_s \cdot \Big(\mathsf{Adv}^{\mathsf{COLL}}_{\mathsf{H},\mathcal{B}_1}$$

$$+ n_p \cdot \Big(\mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Expand},\mathcal{B}_2} + \mathsf{Adv}^{\mathsf{HMAC(0,\$)\text{-}\$}}_{\mathsf{HMAC},\mathcal{B}_3} + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HMAC},\mathcal{B}_4} + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Expand},\mathcal{B}_5} \Big)$$

$$+ \mathsf{Adv}^{\mathsf{HMAC(0,\$)\text{-}\$}}_{\mathsf{HMAC},\mathcal{B}_7} \quad \mathsf{F\text{-}sec}_{\mathsf{HMAC},\mathcal{B}_9}$$

$$+ \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Expand},\mathcal{B}_{10}} + \mathsf{Adv}^{\mathsf{EUF\text{-}CMA}}_{\mathsf{HMAC},\mathcal{B}_{11}} \Big)$$

$$+ n_s \cdot n_p \cdot \Big(\mathsf{Adv}^{\mathsf{snPRF\text{-}ODH}}_{\mathsf{HKDF.Extract},\mathbb{G},\mathcal{B}_{12}} + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HMAC},\mathcal{B}_{13}} + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Expand},\mathcal{B}_{14}} + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Expand},\mathcal{B}_{15}} + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Expand},\mathcal{B}_{16}} \Big) \Big).$$

$\mathbf{PRF}(\mathbf{g}^{\mathbf{uv}}, \mathbf{x}) \approx_c \$$, given oracle $\mathbf{PRF}(\cdot^{\mathbf{u}}, \cdot)$

[BFGJ17]

TLS 1.3 Handshake Security

In perspective

UC San Diego

- **cryptographic design** of TLS 1.3 handshake is **sound**
- strong security results for main keys (both full and PSK handshakes)
- replays and lacking forward secrecy for 0-RTT are a (recognized) downside

- recall: focus on handshake modes in isolation, for draft-14 (and earlier)

- further analyses:
  - other computational analyses of sub-parts (e.g., post-handshake client auth)
  - tool-based/symbolic analyses up to full protocol and on multiple drafts
  - work-in-progress verified implementation

- jointly, these analyses give rise to confidence in TLS 1.3 handshake design
- still, doesn't mean there won't be any attacks (bets are on 0-RTT. . .)

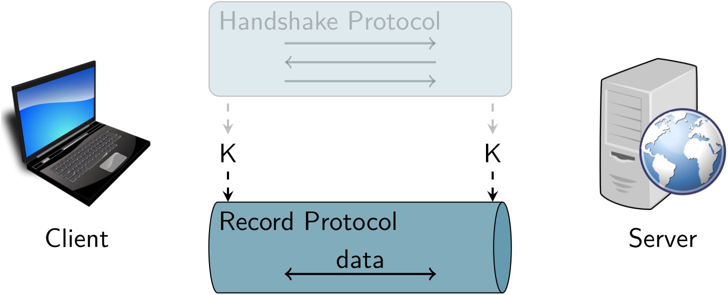
TLS 1.3
Record Protocol & Some Analysis

The TLS Protocol

So. . . what about the Record Protocol?

Handshake Protocol:

- negotiate security parameters (“cipher suite”)
- authenticate peers
- establish key material for data protection



Handshake Protocol

K

K

Record Protocol

data

Client

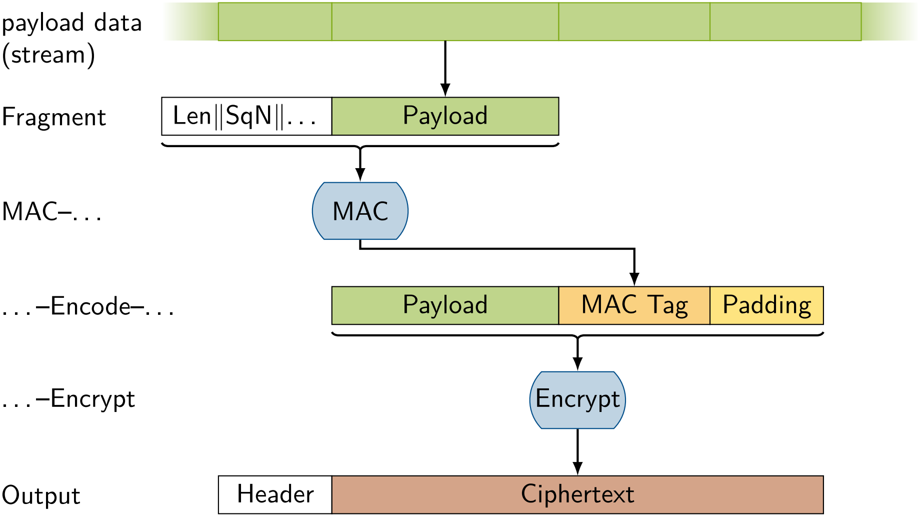
Server

Record Protocol:

- protect data using key material from handshake
- ensuring confidentiality and integrity

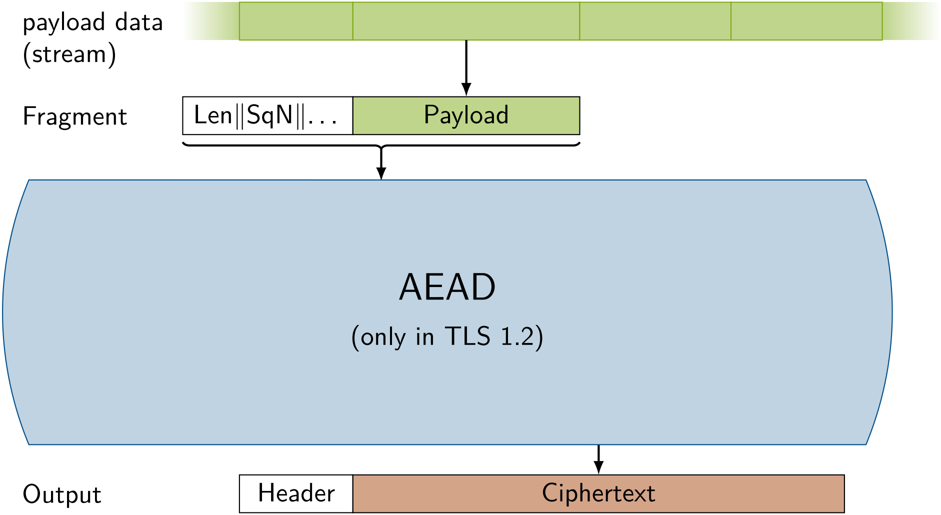
The TLS Protocol

Recap: Record Protocol Structure up to TLS 1.2

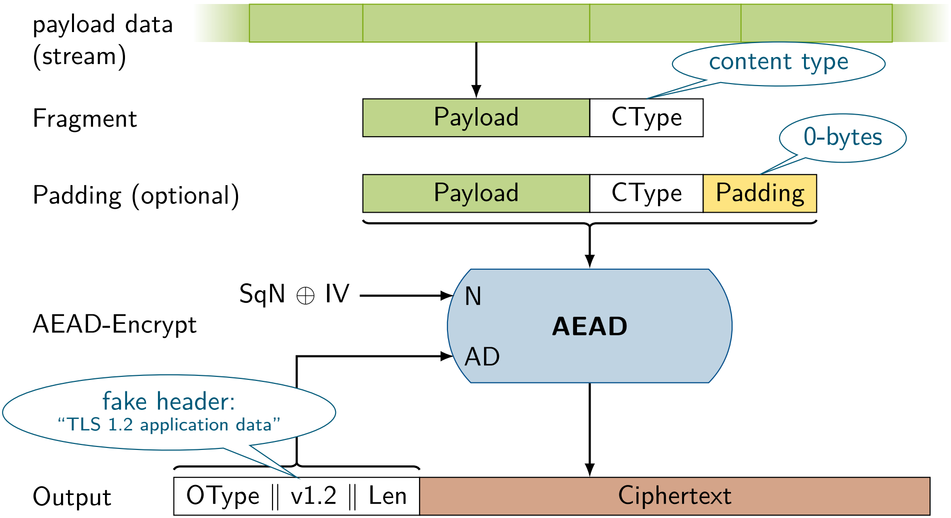


The TLS Protocol

Recap: Record Protocol Structure up to TLS 1.2

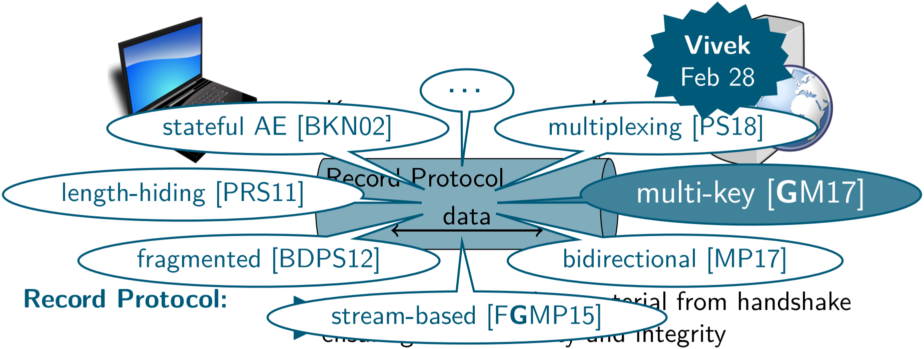


The TLS 1.3 Record Protocol



TLS 1.3 Record Protocol Security

- AEAD-based design looks sound...
- but the crypto community hasn't really conclusively ventilated the question: **What is a secure channel protocol?**

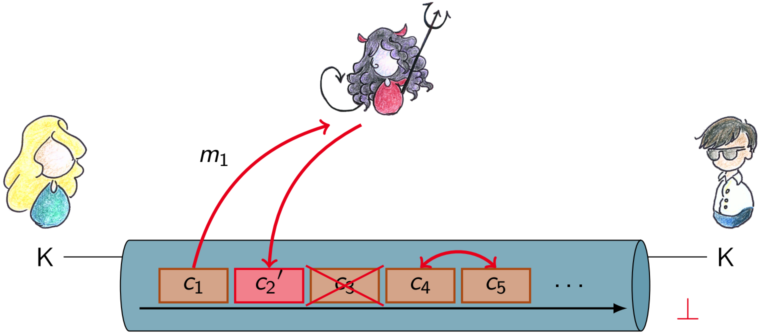


Record Protocol:

- terial from handshake
- ensu integrity

Channel Security

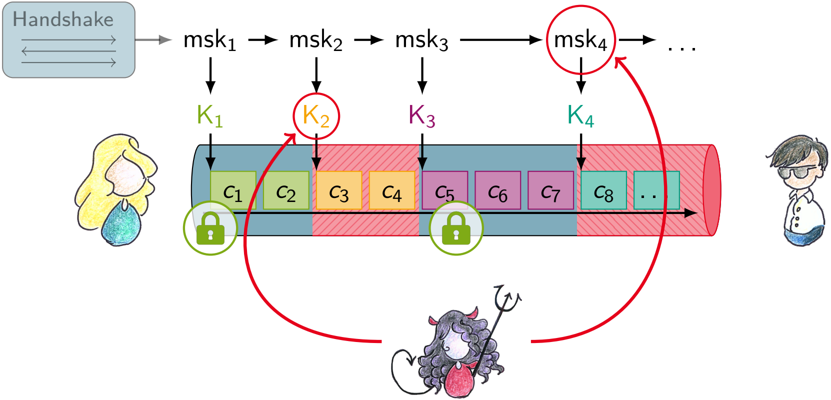
Recap: Bellare, Kohno, Namprempre 2002 [BKN02]



IND-sfCPA	(passive confidentiality)	INT-sfPTXT	(plaintext integrity)
IND-sfCCA	(active confidentiality)	INT-sfCTXT	(ciphertext integrity)

Multi-key Channels

- keys updated during channel operation (e.g., TLS 1.3, Signal, . . .)



(phase-)key insulation

forward security

[GM17]

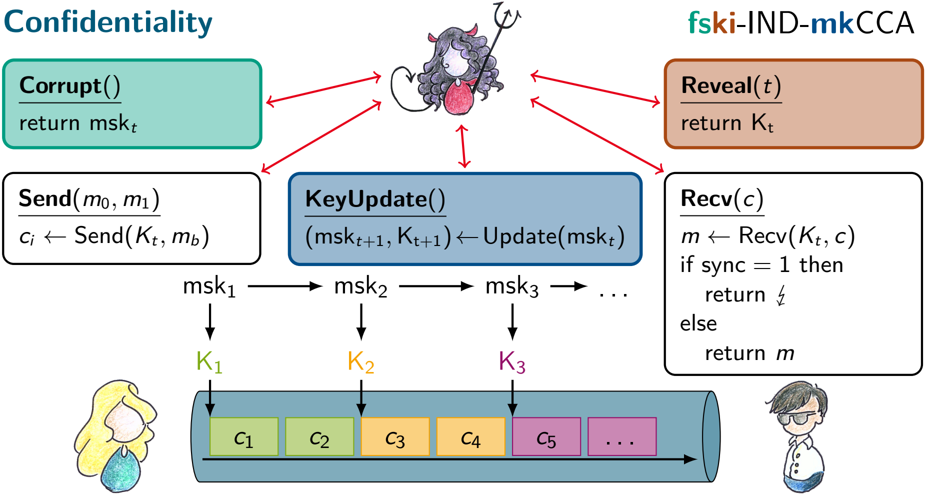
Multi-key Channels

Security Model

UC San Diego

Confidentiality

fski-IND-mkCCA



Corrupt()

```
return $\mathsf{msk}_t$
```

Reveal(t)

```
return $\mathsf{K}_\mathsf{t}$
```

Send(m_0, m_1)

```
$c_i \leftarrow \mathrm{Send}(K_t, m_b)$
```

KeyUpdate()

```
$(\mathsf{msk}_{t+1}, \mathsf{K}_{\mathsf{t+1}}) \leftarrow \mathrm{Update}(\mathsf{msk}_t)$
```

Recv(c)

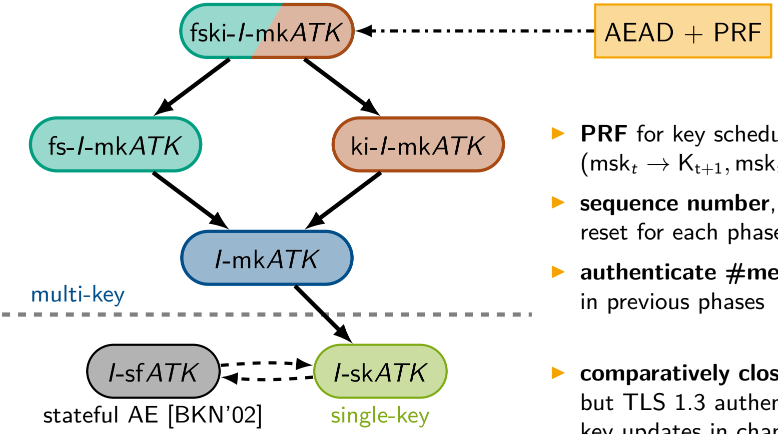
```
$m \leftarrow \mathrm{Recv}(K_t, c)$
if sync $= 1$ then
    return ↯
else
    return $m$
```

$\mathsf{msk}_1 \rightarrow \mathsf{msk}_2 \rightarrow \mathsf{msk}_3 \rightarrow \dots$
K_1
K_2
K_3
c_1
c_2
c_3
c_4
c_5
...

(simplified)

Multi-key Channels

Security Hierarchy and Instantiation



- **PRF** for key schedule ($\mathsf{msk}_t \to \mathsf{K}_{t+1}, \mathsf{msk}_{t+1}$)
- **sequence number**, reset for each phase
- **authenticate #messages** in previous phases
- **comparatively close to TLS 1.3**, but TLS 1.3 authenticates key updates in channel

$I = \mathrm{IND}, \quad ATK \in \{\mathrm{CPA}, \mathrm{CCA}\}$
$I = \mathrm{INT}, \quad ATK \in \{\mathrm{PTXT}, \mathrm{CTXT}\}$

Thank You!

TLS 1.3

Feb 21	**TLS 1.3** [TLS13] **& some security models** [FG17,GM17]	Felix
Feb 28	**Multiplexing channels** [PS18]	Vivek
Mar 7	**Symbolic Tamarin analysis** [CHH+17]	Baiyu
Mar 14	**Downgrade resilience** [BBF+16]	Ruth

References I

[BMM+15] C. Badertscher, C. Matt, U. Maurer, P. Rogaway, and B. Tackmann. "Augmented Secure Channels and the Goal of the TLS 1.3 Record Layer". In: *ProvSec 2015*. Ed. by M. H. Au and A. Miyaji. Vol. 9451. LNCS. Springer, Heidelberg, Nov. 2015, pp. 85–104.

[BKN02] M. Bellare, T. Kohno, and C. Namprempre. "Authenticated Encryption in SSH: Provably Fixing The SSH Binary Packet Protocol". In: *ACM CCS 02*. Ed. by V. Atluri. ACM Press, Nov. 2002, pp. 1–11.

[BR94] M. Bellare and P. Rogaway. "Entity Authentication and Key Distribution". In: *CRYPTO'93*. Ed. by D. R. Stinson. Vol. 773. LNCS. Springer, Heidelberg, Aug. 1994, pp. 232–249.

[Ben18] D. Benjamin. *TLS Ecosystem Woes: Why your Crypto isn't Real World yet*. Presented at the Real World Crypto Symposium 2018, `https://docs.google.com/presentation/d/1jqyTwZlTPD_xp4rTD4FmbsdKYWRHcUkN5lfMeGQZQ_o/`. 2018.

[BBDL+16] B. Beurdouche, K. Bhargavan, A. Delignat-Lavaud, C. Fournet, S. Ishtiaq, M. Kohlweiss, J. Protzenko, N. Swamy, S. Zanella-Béguelin, and J. K. Zinzindohoué. *Towards a Provably Secure Implementation of TLS 1.3*. Presented at the TRON Workshop at NDSS 2016. 2016.

[BBK17] K. Bhargavan, B. Blanchet, and N. Kobeissi. "Verified Models and Reference Implementations for the TLS 1.3 Standard Candidate". In: *2017 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, May 2017, pp. 483–502.

[BBF+16] K. Bhargavan, C. Brzuska, C. Fournet, M. Green, M. Kohlweiss, and S. Z. Béguelin. "Downgrade Resilience in Key-Exchange Protocols". In: *2016 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, May 2016, pp. 506–525.

References II

[BL16] K. Bhargavan and G. Leurent. "Transcript Collision Attacks: Breaking Authentication in TLS, IKE and SSH". In: *NDSS 2016*. The Internet Society, Feb. 2016.

[BDPS12] A. Boldyreva, J. P. Degabriele, K. G. Paterson, and M. Stam. "Security of Symmetric Encryption in the Presence of Ciphertext Fragmentation". In: *EUROCRYPT 2012*. Ed. by D. Pointcheval and T. Johansson. Vol. 7237. LNCS. Springer, Heidelberg, Apr. 2012, pp. 682–699.

[BFGJ17] J. Brendel, M. Fischlin, F. Günther, and C. Janson. "PRF-ODH: Relations, Instantiations, and Impossibility Results". In: *CRYPTO 2017, Part III*. Ed. by J. Katz and H. Shacham. Vol. 10403. LNCS. Springer, Heidelberg, Aug. 2017, pp. 651–681.

[CHH+17] C. Cremers, M. Horvat, J. Hoyland, S. Scott, and T. van der Merwe. "A Comprehensive Symbolic Analysis of TLS 1.3". In: *ACM CCS 17*. Ed. by B. M. Thuraisingham, D. Evans, T. Malkin, and D. Xu. ACM Press, 2017, pp. 1773–1788.

[CHSM16] C. Cremers, M. Horvat, S. Scott, and T. van der Merwe. "Automated Analysis and Verification of TLS 1.3: 0-RTT, Resumption and Delayed Authentication". In: *2016 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, May 2016, pp. 470–485.

[DLFK+17] A. Delignat-Lavaud, C. Fournet, M. Kohlweiss, J. Protzenko, A. Rastogi, N. Swamy, S. Z. Béguelin, K. Bhargavan, J. Pan, and J. K. Zinzindohoue. "Implementing and Proving the TLS 1.3 Record Layer". In: *2017 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, May 2017, pp. 463–482.

References III

[DFGS15] B. Dowling, M. Fischlin, F. Günther, and D. Stebila. “A Cryptographic Analysis of the TLS 1.3 Handshake Protocol Candidates”. In: *ACM CCS 15*. Ed. by I. Ray, N. Li, and C. Kruegel: ACM Press, Oct. 2015, pp. 1197–1210.

[DFGS16] B. Dowling, M. Fischlin, F. Günther, and D. Stebila. *A Cryptographic Analysis of the TLS 1.3 draft-10 Full and Pre-shared Key Handshake Protocol*. Cryptology ePrint Archive, Report 2016/081. `http://eprint.iacr.org/2016/081`. 2016.

[FG14] M. Fischlin and F. Günther. “Multi-Stage Key Exchange and the Case of Google's QUIC Protocol”. In: *ACM CCS 14*. Ed. by G.-J. Ahn, M. Yung, and N. Li. ACM Press, Nov. 2014, pp. 1193–1204.

[FG17] M. Fischlin and F. Günther. “Replay Attacks on Zero Round-Trip Time: The Case of the TLS 1.3 Handshake Candidates”. In: *2017 IEEE European Symposium on Security and Privacy, EuroS&P 2017*. Paris, France: IEEE, 2017, pp. 60–75.

[FGMP15] M. Fischlin, F. Günther, G. A. Marson, and K. G. Paterson. “Data Is a Stream: Security of Stream-Based Channels”. In: *CRYPTO 2015, Part II*. Ed. by R. Gennaro and M. J. B. Robshaw. Vol. 9216. LNCS. Springer, Heidelberg, Aug. 2015, pp. 545–564.

[Gün18] F. Günther. “Modeling Advanced Security Aspects of Key Exchange and Secure Channel Protocols”. `http://tuprints.ulb.tu-darmstadt.de/7162/`. PhD thesis. Darmstadt, Germany: Technische Universität Darmstadt, 2018.

[GM17] F. Günther and S. Mazaheri. “A Formal Treatment of Multi-key Channels”. In: *CRYPTO 2017, Part III*. Ed. by J. Katz and H. Shacham. Vol. 10403. LNCS. Springer, Heidelberg, Aug. 2017, pp. 587–618.

References IV

[JSS15] T. Jager, J. Schwenk, and J. Somorovsky. "On the Security of TLS 1.3 and QUIC Against Weaknesses in PKCS#1 v1.5 Encryption". In: *ACM CCS 15*. Ed. by I. Ray, N. Li, and C. Kruegel: ACM Press, Oct. 2015, pp. 1185–1196.

[Kra16] H. Krawczyk. "A Unilateral-to-Mutual Authentication Compiler for Key Exchange (with Applications to Client Authentication in TLS 1.3)". In: *ACM CCS 16*. Ed. by E. R. Weippl, S. Katzenbeisser, C. Kruegel, A. C. Myers, and S. Halevi. ACM Press, Oct. 2016, pp. 1438–1450.

[KW16] H. Krawczyk and H. Wee. "The OPTLS Protocol and TLS 1.3". In: *2016 IEEE European Symposium on Security and Privacy, EuroS&P 2016*. Saarbrücken, Germany: IEEE, 2016, pp. 81–96.

[Mac17] C. MacCárthaigh. *Security Review of TLS 1.3 0-RTT*. `https://github.com/tlswg/tls13-spec/issues/1001`. 2017.

[MP17] G. A. Marson and B. Poettering. "Security Notions for Bidirectional Channels". In: *IACR Trans. Symm. Cryptol.* 2017.1 (2017), pp. 405–426.

[PM16] K. G. Paterson and T. van der Merwe. "Reactive and Proactive Standardisation of TLS". In: *Security Standardisation Research: Third International Conference (SSR 2016)*. Ed. by L. Chen, D. A. McGrew, and C. J. Mitchell. Vol. 10074. Lecture Notes in Computer Science. Gaithersburg, MD, USA: Springer, 2016, pp. 160–186.

[PRS11] K. G. Paterson, T. Ristenpart, and T. Shrimpton. "Tag Size Does Matter: Attacks and Proofs for the TLS Record Protocol". In: *ASIACRYPT 2011*. Ed. by D. H. Lee and X. Wang. Vol. 7073. LNCS. Springer, Heidelberg, Dec. 2011, pp. 372–389.

References V

[PS18] C. Patton and T. Shrimpton. "Partially Specified Channels: The TLS 1.3 Record Layer without Elision". In: *ACM CCS 18*. Ed. by D. Lie, M. Mannan, M. Backes, and X. Wang. ACM Press, Oct. 2018, pp. 1415–1428.

[TLS13] E. Rescorla. *The Transport Layer Security (TLS) Protocol Version 1.3*. RFC 8446 (Proposed Standard). RFC. Fremont, CA, USA: RFC Editor, Aug. 2018.