Key Confirmation in Key Exchange

A Formal Treatment and Implications for TLS 1.3

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Felix Günther

Technische Universität Darmstadt, Germany

joint work with Marc Fischlin, Benedikt Schmidt, and Bogdan Warinschi

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Cryptoplexity
Cryptography & Complexity Theory
Technische Universität Darmstadt
www.cryptoplexity.de

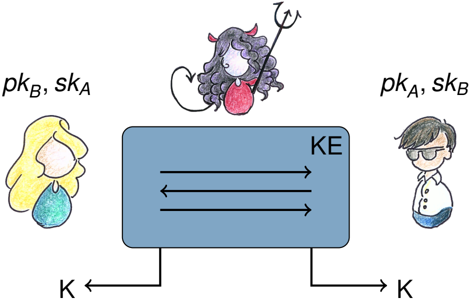
CROSSING

institute
imdea
software

University of
BRISTOL

Key Exchange

Security Goals (à la Bellare–Rogaway 1993)



key secrecy

“the key looks random (to Eve)”

(implicit) authentication

“only/at most Bob can hold the key”

drawings by *Giorgia Azzurra Marson*

Key Confirmation

The informal understanding

> ***Key confirmation*** *is the property whereby one party is assured that a second (possibly unidentified) party actually has possession of a particular secret key.*
>
> Handbook of Applied Cryptography, Definition 12.7

- ensuring Bob actually holds the key

- often mentioned in scientific papers on key exchange
- but no formal treatment so far

Key Confirmation Matters

TECHNISCHE
UNIVERSITÄT
DARMSTADT

- crypto: no security problem if no one can decrypt
- seems “clear”: use the key and you get key confirmation

?

- caution: unmitigated use of session key destroys BR’93 key secrecy

!

- folklore transform: derive separate key & send a MAC

- discussions around TLS 1.3: is key confirmation needed?
- specified as goal by NIST standards and for TLS 1.3

?!

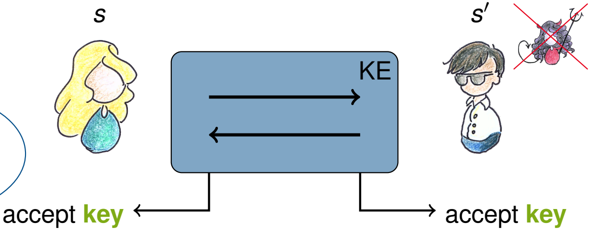
Our Results

- formalize key confirmation to enable a well-founded discussion
- revisit the “refresh-then-MAC” protocol transform
- analyze TLS 1.3 (draft-10) for key confirmation

Formal Model

Full Key Confirmation

- **intuition:** when a session accepts, another session already holds the key



s

s'

KE

accept **key**

accept **key**

e.g.,
$\mathcal{S}$ = clients,
$\mathcal{S}'$ = servers

$$\forall s \in \mathcal{S} :: [s.\text{status} = \texttt{accept} \wedge s.\text{peer} \notin \text{Corr} \cup \{*\}]$$
$$\implies \exists s' \in \mathcal{S}' :: (s'.\text{status} = \texttt{accept} \wedge s.\text{sid} = s'.\text{sid} \wedge s.\textbf{key} = s'.\textbf{key})$$

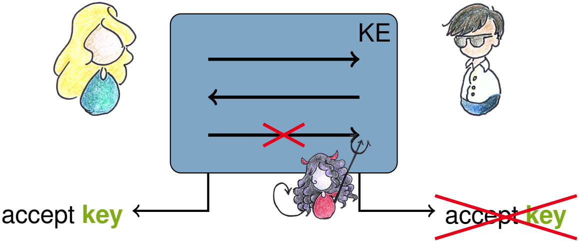
modular

- formalization exposes duality to classical authentication notion:
  - (implicit) authentication: there exists *at most one* session that holds the key
  - (full) key confirmation: there exists *at least one* session that holds the key

Formal Model

Full Key Confirmation

- **note:** cannot have mutual full key confirmation

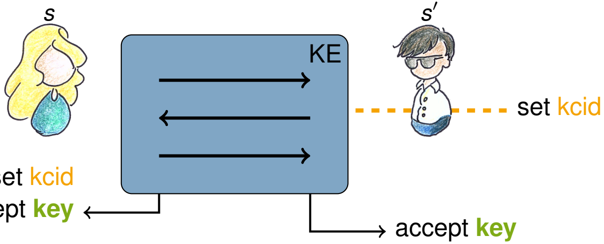


- adversary can always drop the last message
- i.e., sender of last message can only get weaker guarantees

Formal Model

Almost-Full Key Confirmation

- **intuition:** when a session accepts, another session exists that, *if it accepts*, will hold the same key
- introduce **key-confirmation identifier** kcid



set kcid

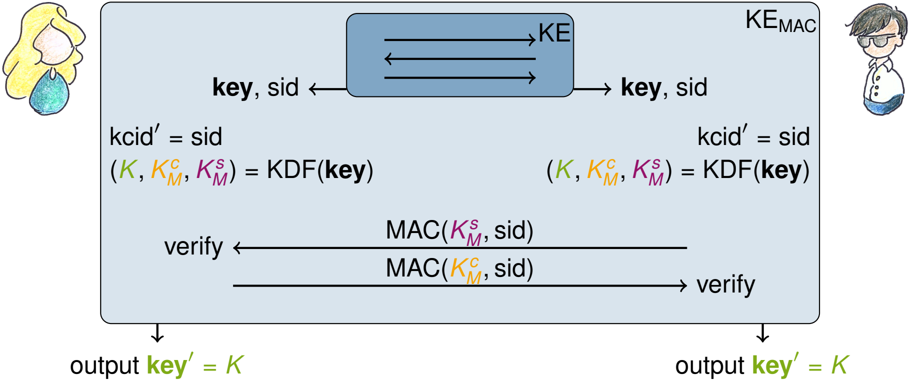
accept **key**

set kcid

accept **key**

$$\forall s \in \mathcal{S} :: [s.\text{status} = \texttt{accept} \wedge s.\text{peer} \notin \text{Corr} \cup \{*\}]$$
$$\implies \exists s' \in \mathcal{S}' :: [s.\text{kcid} = s'.\text{kcid} \wedge (s'.\text{status} = \texttt{accept} \implies s.\textbf{key} = s'.\textbf{key})]$$

“Refresh-then-MAC” Protocol Transform



$\mathsf{KE}_{\mathsf{MAC}}$

KE

key, sid ← → **key**, sid

$\mathrm{kcid}' = \mathrm{sid}$ $\qquad$ $\mathrm{kcid}' = \mathrm{sid}$

$(K, K_M^c, K_M^s) = \mathrm{KDF}(\mathbf{key})$ $\qquad$ $(K, K_M^c, K_M^s) = \mathrm{KDF}(\mathbf{key})$

verify $\xleftarrow{\mathrm{MAC}(K_M^s, \mathrm{sid})}$

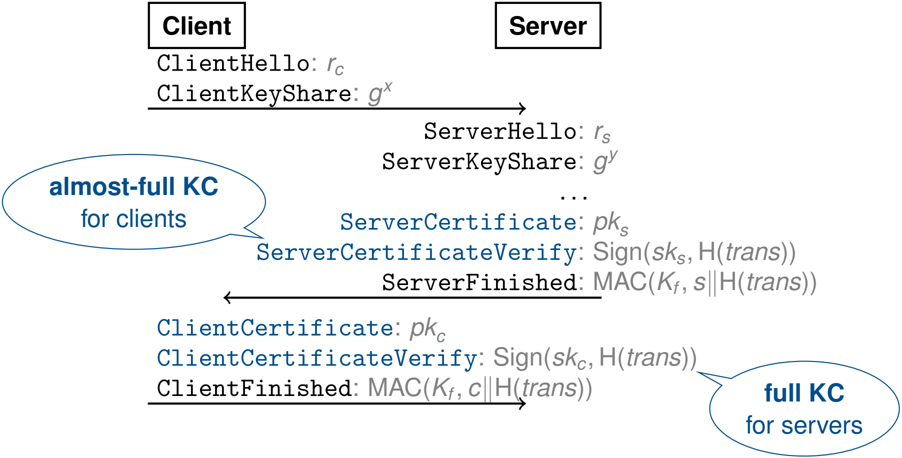
$\xrightarrow{\mathrm{MAC}(K_M^c, \mathrm{sid})}$ verify

output $\mathbf{key}' = K$ $\qquad$ output $\mathbf{key}' = K$

- key secrecy and authentication is preserved
- full KC for receiver, almost-full KC for sender of final message

Key Confirmation in TLS 1.3

draft-10 Full (EC)DHE Handshake



Client

Server

`ClientHello`: r_c

`ClientKeyShare`: g^x

`ServerHello`: r_s

`ServerKeyShare`: g^y

...

`ServerCertificate`: pk_s

`ServerCertificateVerify`: $\mathsf{Sign}(sk_s, \mathsf{H}(trans))$

`ServerFinished`: $\mathsf{MAC}(K_f, s \| \mathsf{H}(trans))$

`ClientCertificate`: pk_c

`ClientCertificateVerify`: $\mathsf{Sign}(sk_c, \mathsf{H}(trans))$

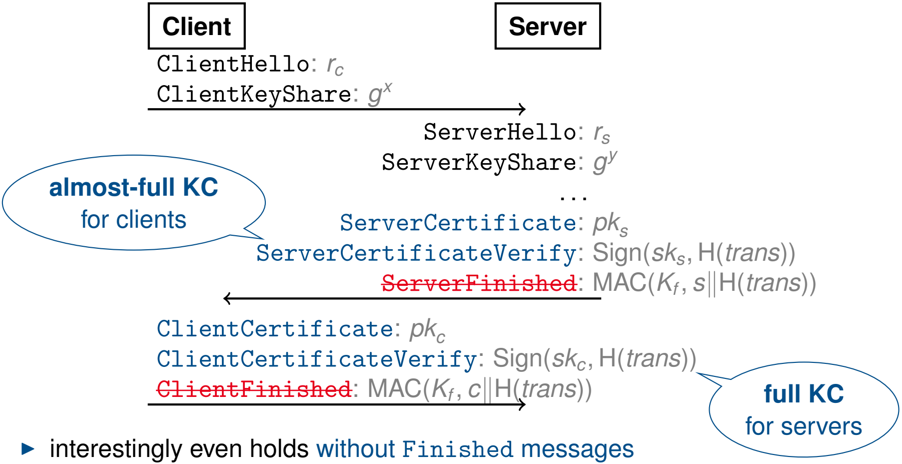
`ClientFinished`: $\mathsf{MAC}(K_f, c \| \mathsf{H}(trans))$

almost-full KC
for clients

full KC
for servers

Key Confirmation in TLS 1.3

draft-10 Full (EC)DHE Handshake



Client

Server

`ClientHello`: r_c

`ClientKeyShare`: g^x

`ServerHello`: r_s

`ServerKeyShare`: g^y

...

`ServerCertificate`: pk_s

`ServerCertificateVerify`: $\mathrm{Sign}(sk_s, \mathrm{H}(trans))$

~~`ServerFinished`~~: $\mathrm{MAC}(K_f, s \| \mathrm{H}(trans))$

`ClientCertificate`: pk_c

`ClientCertificateVerify`: $\mathrm{Sign}(sk_c, \mathrm{H}(trans))$

~~`ClientFinished`~~: $\mathrm{MAC}(K_f, c \| \mathrm{H}(trans))$

almost-full KC
for clients

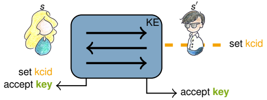
full KC
for servers

- interestingly even holds without `Finished` messages
- in the paper: (similar) results on unilateral authentication case

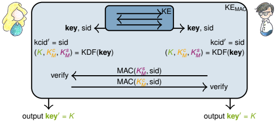
Summary

We

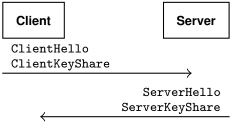
- formalize key confirmation
  in a game-based model
  - full key confirmation for receiver,
  - almost-full key confirmation for sender of last message



- confirm that the “refresh-then-MAC” transform generically adds key confirmation



- show that TLS 1.3 provides key confirmation
  (even without `Finished` messages)



Thank You!