How (not) to use TLS between 3 parties

Ioana Boureanu (Univ. of Surrey, UK)

Talk at Cryptacus Workshop, Nijmegen

18/11/2017

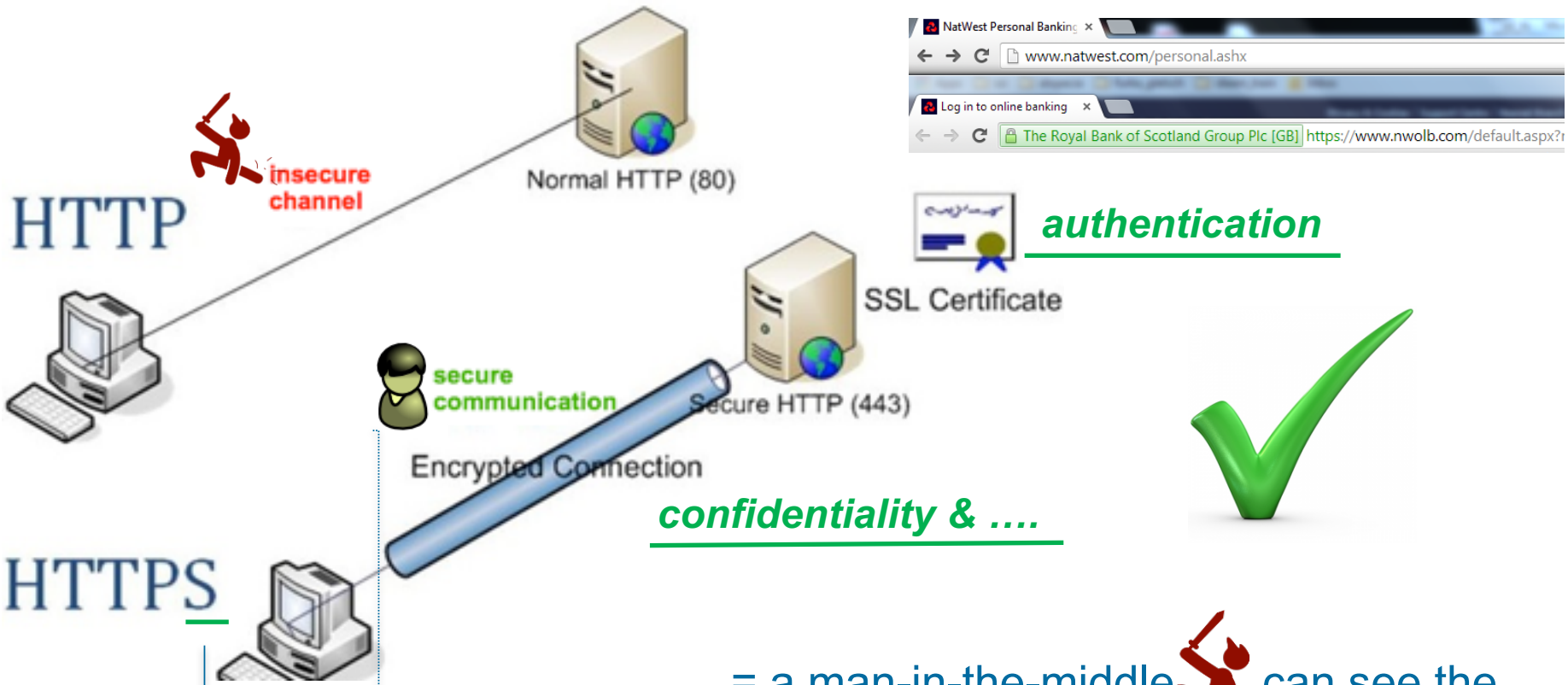
a collaboration with K. Bhargavan (Inria, Paris, France),
P. A. Fouque (UR1, Rennes, France),
C. Onete (U. of Limoges, France),
B. Richard (Orange, France)

Based on a paper at Euro S&P 2017

PRESENTATION OUTLINE

1. Context/Aim: achieve secure communication over insecure channels
2. Attacks on Cloudflare's Keyless SSL: How 2ACCE-security breaks with 3 parties
3. A Provably Secure Keyless SSL Variant
4. Food for Thought

Why SSL/TLS ? … e.g., HTTP vs. HTTPS

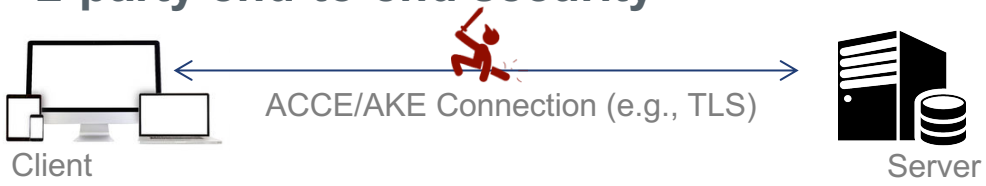


= a man-in-the-middle can see the msgs pass, but he/she/it cannot (for a start) understand them

via **TLS** → **Aim: Secure communications over insecure channels**

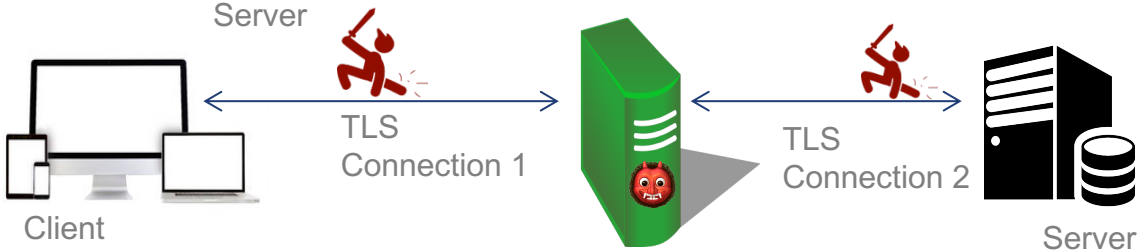
TLS & Authenticated and Confidential Channel Establishment (ACCE)

ACCE or AKE (authenticated key exchange) security models, and protocols implementing them (provably or otherwise) were conceived for **2-party end-to-end security**

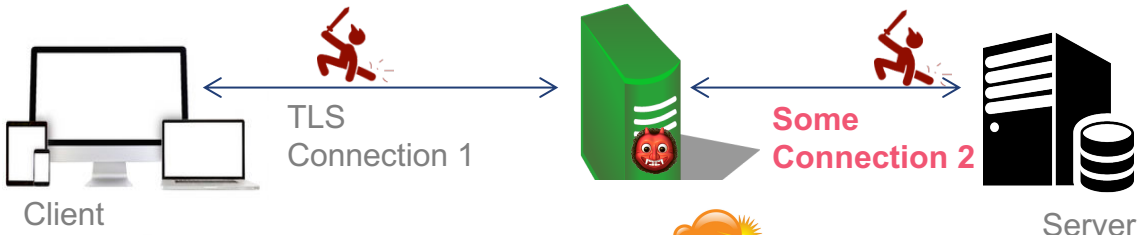


!!

What guarantees do/should/could we have for 3-party ACCE??



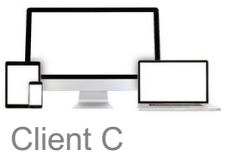
Content Filter/
*Reverse Caching Proxy or **Content Delivery Network** (CDN)/*
(Web) Application Firewall/
Corporate Proxy/…



CDN called CLOUDFLARE

!! product called **Keyless SSL** **!!**

Preamble: The TLS/SSL Handshake (RSA-mode)



$\mathrm{sk}_S, \mathrm{pk}_S, \mathrm{Cert}_S$

N_C, ConfigList, ExtList →

← N_S, Config, Ext

← $\mathrm{KE}_S = (pk_S, \mathrm{Cert}_S)$

Choose **pmk**
Set: $\boldsymbol{msk} = \mathrm{HMAC}_{pmk}(N_C, N_S)$
$KE_C = \mathrm{Enc}_{pk_S}(\mathrm{pmk})$
$(K_C, K_S) = \mathrm{HMAC}_{msk}(N_C, N_S)$
$\mathrm{CFin} = \mathrm{HMAC}_{msk}(H[N_C \ldots KE_C])$

$\mathrm{KE}_C, \{\mathrm{CFin}\}_{K_C}$ →

Decrypt KE_C to get pmk
Compute msk, (K_C, K_S)
Decrypt $\{\mathrm{CFin}\}_{K_C}$ and check $CFin$
$\mathrm{SFin} = \mathrm{HMAC}_{msk}(H[N_C \ldots \mathrm{CFin}])$

← $\{\mathrm{SFin}\}_{K_S}$

- TLS 1.2 in specific modes with specific primitives/assumptions → provably secure
- session resumption: shortcutting a **new handshake**, from **old msk**, **no sk_S** being used at the current run

Preamble: Classical TLS over CDNs vs. Cloudflare's Keyless SSL

Client

Classical CDN, Edge-Server

$sk_S^{Impersonation} = sk_S$
$pk_S^{Impersonation} = pk_S$
$Cert_S^{Impersonation} = Cert_S$

Origin Server **S**

$sk_S^{Origin}, pk_S^{Origin}, Cert_S^{Origin}$

Client

Cloudflare's Keyless Edge-ServerCDN

$pk_S^{Impersonation} = pk_S$
$Cert_S^{Impersonation} = Cert_S$

Origin Server **S**

$sk_S^{Origin}, pk_S^{Origin}, Cert_S^{Origin}$

$\boxed{sk_S}, pk_S, Cert_S$

Cloudflare's Keyless SSL (RSA mode)

N_C, ConfigList, ExtList →

← N_S, Config, Ext

← $KE_S = (pk_S, Cert_S)$

Choose pmk
Set: $msk = HMAC_{pmk}(N_C, N_S)$
$KE_C = Enc_{pk_S}(pmk)$
$(K_C, K_S) = HMAC_{msk}(N_C, N_S)$
$CFin = HMAC_{msk}(H[N_C \ldots KE_C])$

$KE_C, \{CFin\}_{K_C}$ →

KE_C →

← **pmk**

! uses his sk to produce a **channel key for C—MW, to which he has no access**

Get **msk**, (K_C, K_S), check CFin
$SFin = HMAC_{msk}(H[N_C \ldots CFin])$

! gets **pmk**

← $\{SFin\}_{K_S}$

Cloudflare's Keyless SSL -- security goals (informally proven)

"The adversary cannot read or insert messages on the authenticated encryption channel between the client and edge server, provided that the client and edge server's session keys were not compromised, and the long-term private key of the origin server that the client thinks it is talking to was not com- promised, and no edge server's private key was compromised between the time when the client sent its first message and accepts."

=

confidentiality and integrity of a proxied TLS connection

- if the server's private key is kept secret,
- if *all middleboxes' private keys are secret until the end of the handshake*.

Does Keyless SSL achieve this?

Are these goals enough?

Cloudflare's Keyless SSL -- why do we care?

Many reasons...

+

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: December 30, 2016

D. Migault
K. Ma
Ericsson
R. Rich
Akamai
S. Mishra
Verizon Communications
O. Gonzales de Dios
Telefonica
June 28, 2016

LURK TLS/DTLS Use Cases
draft-mglt-lurk-tls-use-cases-02

Abstract

TLS as been designed to setup and authenticate transport layer between a TLS Client and a TLS Server. In most cases, the TLS Server both terminates the TLS Connection and owns the authentication credentials necessary to authenticate the TLS Connection.

This document provides use cases where these two functions are split into different entities, i.e. the TLS Connection is terminated on an Edge Server, while authentication credentials are generated by a Key Server, that owns the Private Key.

PRESENTATION OUTLINE

1. Context/Aim: achieve secure communication over insecure channels
2. Attacks on Cloudflare's Keyless SSL: How 2ACCE-security breaks with 3 parties
3. A Provably Secure Keyless SSL Variant
4. Food for Thought

Keyless SSL's/CDNs' Issue1: Lack of Accountability

Client C

pk_S, $Cert_S$

Keyless SSL Edge-Server MW

sk_S, pk_S, $Cert_S$

Origin Server S

N_C, ConfigList, ExtList →

↔

…

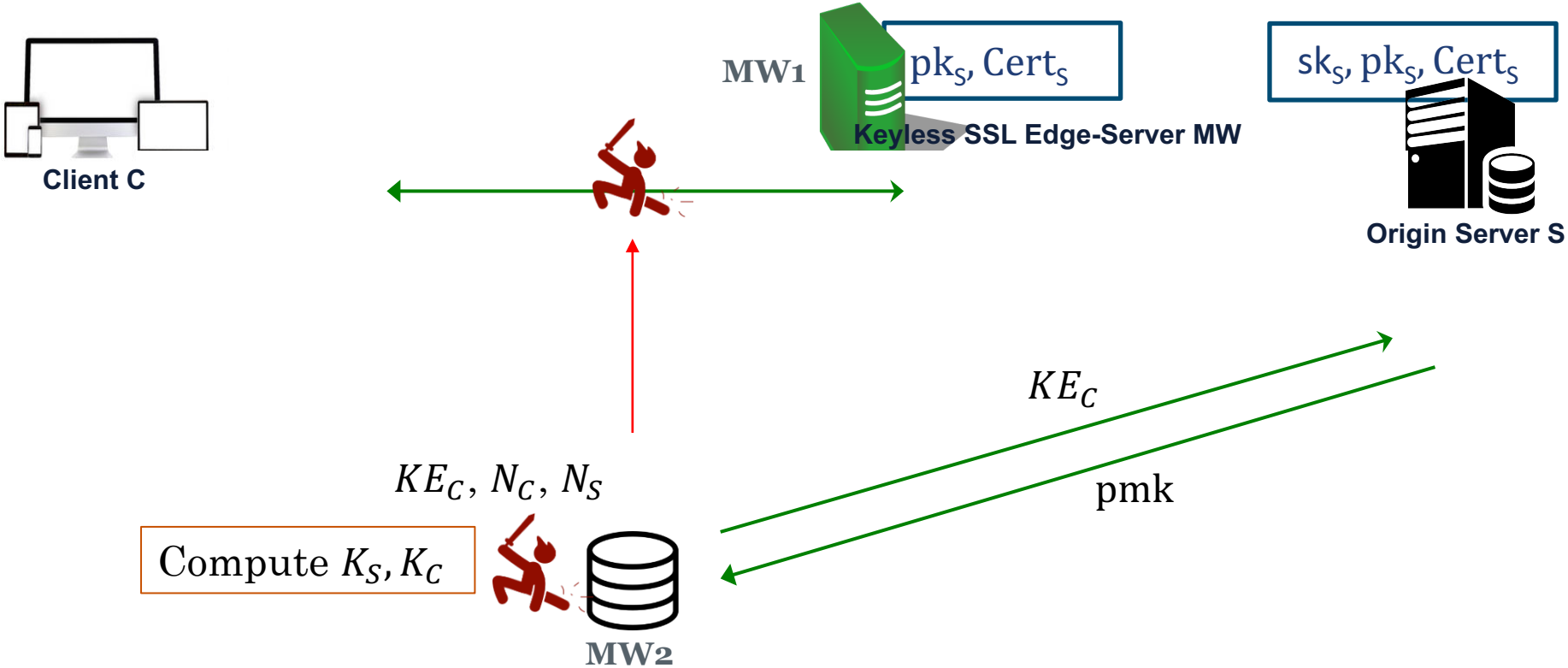
$\{SFin\}_{K_S}$

← Malware

The CDN can send what it pleases, fully on behalf of S, without S's knowledge or any possibility of (future) audit!

If the server S knew the channel-key, the S could "audit" the CDN's behaviour!

Keyless SSL's Pb2: One Malicious CDN => Compromise Any/All



A "post-handshake" corruption of a MW2 can compromise sessions of another MW1!
(contradicts the "Keyless-SSL" paper…as per its sec. goals)

If the server S got more than just an out-of-context KE_C (e.g., C-side session-data), then this attack would become harder to mount by malicious CDNs

Keyless SSL's Main Pbs: Recall

Client C

pk_S, $Cert_S$

Keyless SSL Edge-Server MW

sk_S, pk_S, $Cert_S$

Origin Server S

N_C, ConfigList, ExtList →

← N_S, Config, Ext

← $KE_S = (pk_S, Cert_S)$

Choose pmk
Set: $msk = HMAC_{pmk}(N_C, N_S)$
$KE_C = Enc_{pk_S}(pmk)$
$(K_C, K_S) = HMAC_{msk}(N_C, N_S)$
$CFin = HMAC_{msk}(H[N_C \ldots KE_C])$

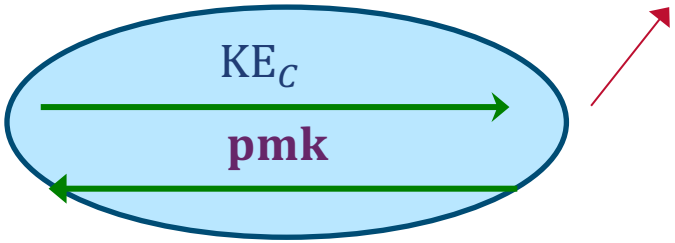
KE_C, $\{CFin\}_{K_C}$ →

Get **msk**, (K_C, K_S), check CFin
$SFin = HMAC_{msk}(H[N_C \ldots CFin])$

← $\{SFin\}_{K_S}$

1. Lack of accountability by S on C-MW session intrinsic to sk_s

2. No C-session data to S =>
   - channel security pb's by malicious MW



PRESENTATION OUTLINE

1. Context/Aim: achieve secure communication over insecure channels
2. Attacks on Cloudflare's Keyless SSL: How 2ACCE-security breaks with 3 parties
3. A Provably Secure Keyless SSL Variant
4. Food for Thought

Keyless SSL's Fix: First-Attempt

pk_S, $Cert_S$

sk_S, pk_S, $Cert_S$

N_C, ConfigList, ExtList →

← N_S, Config, Ext

← $\mathrm{KE}_S = (pk_S, \mathrm{Cert}_S)$

Choose pmk
Set: $msk = \mathrm{HMAC}_{pmk}(N_C, N_S)$
$KE_C = \mathrm{Enc}_{pk_S}(\mathrm{pmk})$
$(K_C, K_S) = \mathrm{HMAC}_{msk}(N_C, N_S)$
$\mathrm{CFin} = \mathrm{HMAC}_{msk}(H[N_C \ldots KE_C])$

$\mathrm{KE}_C, \{\mathrm{CFin}\}_{K_C}$ →

$[N_C, \ldots, \{\mathrm{CFin}\}_{K_C}]$ →

Check validity

+ gains "audit" powers

← **pmk**

Get msk, (K_C, K_S), check CFin
$\mathrm{SFin} = \mathrm{HMAC}_{msk}(H[N_C \ldots \mathrm{CFin}])$

← $\{\mathrm{SFin}\}_{K_S}$

Keyless SSL's Fix: First Attempt + Resumption vs Accountability

msk

(K_C, K_S)

N_C, ConfigList, **Resume** →

msk

(K_C, K_S)

(K_C, K_S)

← N_S, Config, Ext

(K_C^*, K_S^*)

(K_C^*, K_S^*)

???

.....

← $\{SFin\}_{K_S}$

Resumption: run a shorter handshake => computation of session-keys related to a previous, full handshake

Given msk, MW just needs a new tuple of nonces for new session-keys

Accountability??

Keyless SSL's Fix: Second Attempt + Cross-protocol Attack

Client C

pk_S, $Cert_S$

Keyless SSL Edge-Server MW

sk_S, pk_S, $Cert_S$

Origin Server S

N_C, ConfigList, ExtList

N_S, Config, Ext

$KE_S = (pk_S, Cert_S)$

Choose pmk

Set: $msk = HMAC_{pmk}(N_C, N_S)$

$KE_C = Enc_{pk_S}(pmk)$

$(K_C, K_S) = HMAC_{msk}(N_C, N_S)$

$CFin = HMAC_{msk}(H[N_C \ldots KE_C])$

MW chooses all parameters → Cross-protocol, Auth. attacks

$KE_C, \{CFin\}_{K_C}$

$[N_C, \ldots, \{CFin\}_{K_C}]$

Check validity

$(K_C, K_S, SFin)$

$\{SFin\}_{K_S}$

Fixed Keyless TLS 1.2

Client C

pk_S, $Cert_S$

Keyless SSL Edge-Server MW

sk_S, pk_S, $Cert_S$

Origin Server S

N_C, ConfigList, ExtList

N_S, Config, Ext

$KE_S = (pk_S, Cert_S)$

Request

N_S

Unavoidable

S runs entire protocol!

Choose pmk

Set: $msk = HMAC_{pmk}(N_C, N_S)$

$KE_C = Enc_{pk_S}(pmk)$

$(K_C, K_S) = HMAC_{msk}(N_C, N_S)$

$CFin = HMAC_{msk}(H[N_C \ldots KE_C])$

$KE_C, \{CFin\}_{K_C}$

$[N_C, \ldots, \{CFin\}_{K_C}]$

Check validity

$(K_C, K_S, SFin)$

$\{SFin\}_{K_S}$

Fixed Keyless TLS 1.2 (RSA Mode) -- Observations

Authentication and Secure Channel:

- N_S generated honestly
- KE_C given by MW to S in full context; it allows S (honest) to prevent channel-security attacks by MW corruption

Accountability (by S for the C—MW link):

- MW forwards the nonce of C, and S, encrypted CFin, and KE_C
  - S can verify the forwarded nonces are correct as per Cfin and KE_C
- S sends directly the session keys and encrypted SFin
  - No session resumption

Other features:

- Security w.r.t. to new, **formal** ACCE model for 3 parties
- More security guarantees, under some assumptions, such as *content soundness* (i.e., "MW can only deliver contracted material")

Fixing Keyless SSL – Some More Results & Discussions

UNIVERSITY OF SURREY

Original Keyless TLS 1.2. in DHE mode

-- It exhibits cross-protocol attack

-- It ensures no accountability & no content soundness… but that's OK-ish

-- Unfortunately, our Fixed Keyless TLS 1.2 in DHE mode has the same drawbacks as our Fixed Keyless TLS 1.2 in RSA mode (i.e., large PKI, heavy server-side computation)

Keyless TLS 1.3 (based on an older version of the TLS1.3 on-going specs)

-- It did not exist in CloudFlare's original proposal

-- In our paper, we propose a Keyless TLS 1.3 which

- did not allow resumption …
- is more efficient than Fixed Keyless TLS 1.2
- needs a lighter PKI

General Tradeoffs

-- accountability vs limited resumption

-- in-between solutions exist…

Surrey Centre for SCCS Cyber Security

PRESENTATION OUTLINE

1. Context/Aim: achieve secure communication over insecure channels
2. Attacks on Cloudflare's Keyless SSL: How 2ACCE-security breaks with 3 parties
3. A Provably Secure Keyless SSL Variant
4. Food for Thought

Keyless SSL and Other TLS "Compositions" over 3 Parties

UNIVERSITY OF SURREY

CDNs, Filtering, Proxy-ing

- Uses a "cache/process then deliver" strategy to improve efficiency in content delivery over HTTPS:// or to filter content, etc. Ses. resumption does help!
- Provide such services transparently to clients
- A single CDN/Proxy can serve many content-owners simultaneously

- TLS/SSL (1.2) was not designed to be run in 2+, i.e., be composably secure
- CDN services were not designed with client-side security in mind
- Bespoke solutions like Keyless SSL can be even worse than simple "TLS sequencing"…
- CDN services, if corrupted, can break AKE-security in bad ways!
- **CDN-driven security protocols are being IETF-standardised** (see LURK)!
- Clients cannot make informed decisions based on whether they communicate with a CDN/proxy, etc. or the server directly

Surrey Centre for SCCS Cyber Security

Maybe, if possible, some clients would choose security over efficiency…

THANK YOU!

QUESTIONS?

- The speaker thanks C. Onete for some of the illustrations on these slides.
- The copyright of certain figures/icons/pictures used within is not attributed to the presenter. These were employed for illustration purposes only and not to be re-used without further research into their copyright.