MCeTLS: Enabling Secure In-Network Functionality in TLS

David Naylor* Kyle Schomp* Matteo Varvello* Ilias Leontiadis* Jeremy Blackburn*
Diego Lopez* Dina Papagiannaki* Pablo Rodriguez Rodriguez* Peter Steenkiste*

Motivation
Encryption is blinding middleboxes.

Observation 1
The use of encryption online is increasing rapidly.

Observation 2
Middleboxes are frequently used to add functionality or enhance performance.

Can we just use TLS?
Using middleboxes with TLS is broken:
1. Company installs root cert on client
2. Firewall fabricates a cert for foo.com
3. Client accepts fake cert because it's signed by company's root cert
4. Firewall opens separate TLS connection to foo.com

Main Idea
Encryption contexts for fine-grained access control.

Why access control?
Most middleboxes don't need full read/write access to all data.

HTTP Request HTTP Response

<table>
<thead>
<tr>
<th>Parental Filter</th>
<th>Packet Pacer</th>
<th>IDS</th>
<th>WAN Optimizer</th>
<th>Caching</th>
<th>Compression</th>
</tr>
</thead>
<tbody>
<tr>
<td>Headers</td>
<td>Body</td>
<td>Headers</td>
<td>Body</td>
<td></td>
<td></td>
</tr>
<tr>
<td>read only</td>
<td>read/write</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

What are encryption contexts?
An encryption context is a tag associated with a set of middlebox permissions. Applications specify a context for each piece of data.

send(data, context)

Context 1: “Request Headers”
- Read Only:
  - Read/Write:

Context 2: “Request Body”
- Read Only:
  - Read/Write:

Context 3: “Response Headers”
- Read Only:
  - Read/Write:

Context 4: “Response Body”
- Read Only:
  - Read/Write:

How do they work?
Each context has two symmetric keys:

READ KEY
- Given to each middlebox with read or write access to that context. Used to encrypt/decrypt and to generate a MAC for detecting third party changes.

WRITE KEY
- Given to each middlebox with write access to that context. Used to generate a MAC for detecting reader changes.

MCeTLS Record
Each record in MCeTLS carries three MACs. Read and write keys are per-context; the endpoint key is shared across all contexts.

Performance
MCeTLS adds functionality to TLS. Does it add overhead?

- Handshake Size
  MCeTLS introduces minimal data overhead.
  Handshake size increases with the number of contexts and middleboxes.

- Server Load
  MCeTLS introduces moderate CPU load.
  The server can opt out of much of this extra computation.

- Handshake Time
  MCeTLS introduces no time overhead.
  Just like TLS, the MCeTLS handshake is 2 RTTs.

www
Implementation, documentation, and research paper available online: mctls.org