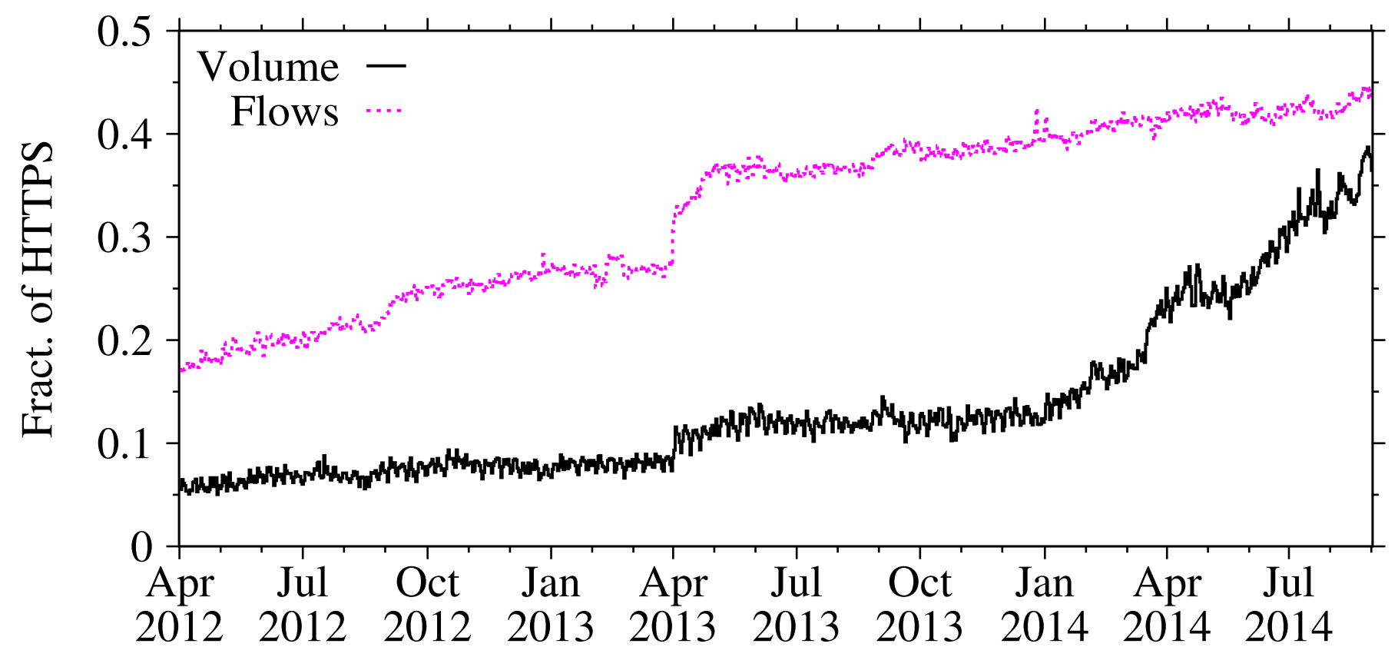
mcTLS:
Enabling Secure In-Network Functionality in TLS

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.



▲ *Encrypted Web Traffic*

We studied the amount of Web traffic using HTTPS in a residential ISP in Europe.

Observation 2

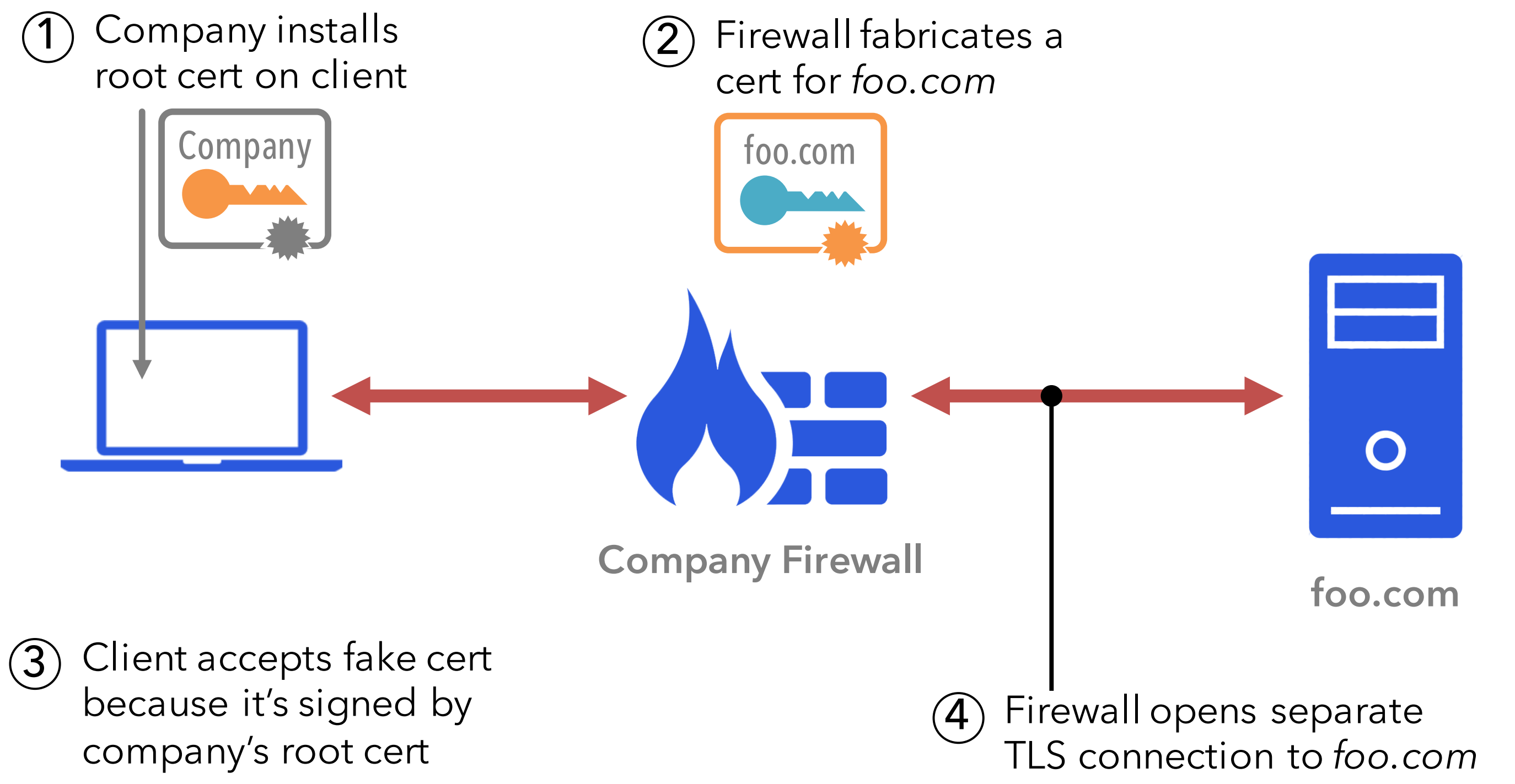
Middleboxes are frequently used to add functionality or enhance performance.

▲ *Example Application-Layer Middleboxes*

Parental filters, virus scanners, and intrusion detection systems add security functionality. Web proxies decrease page load time by caching and decrease data usage by compressing objects.

Can we just use TLS?

Using middleboxes with TLS is broken:


① Company installs root cert on client
Company
② Firewall fabricates a cert for *foo.com*
foo.com
Company Firewall
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.

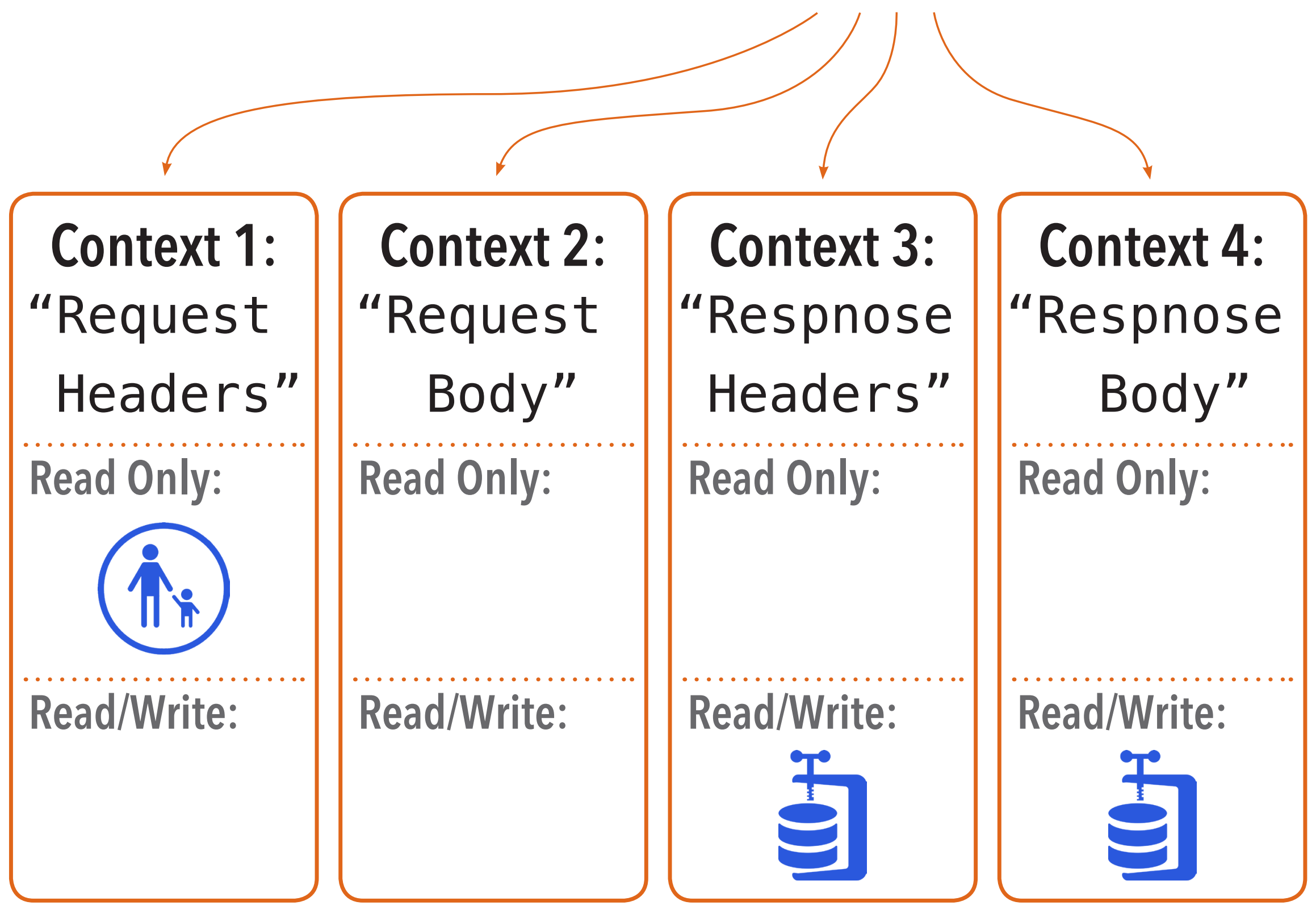
	HTTP Request		HTTP Response	
	Headers	*Body*	*Headers*	*Body*
Parental Filter	○			
Packet Pacer			○	
IDS	○	○	○	○
WAN Optimizer	○	○	○	○
Caching			●	●
Compression			●	●

○ read only ● read/write

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)
```

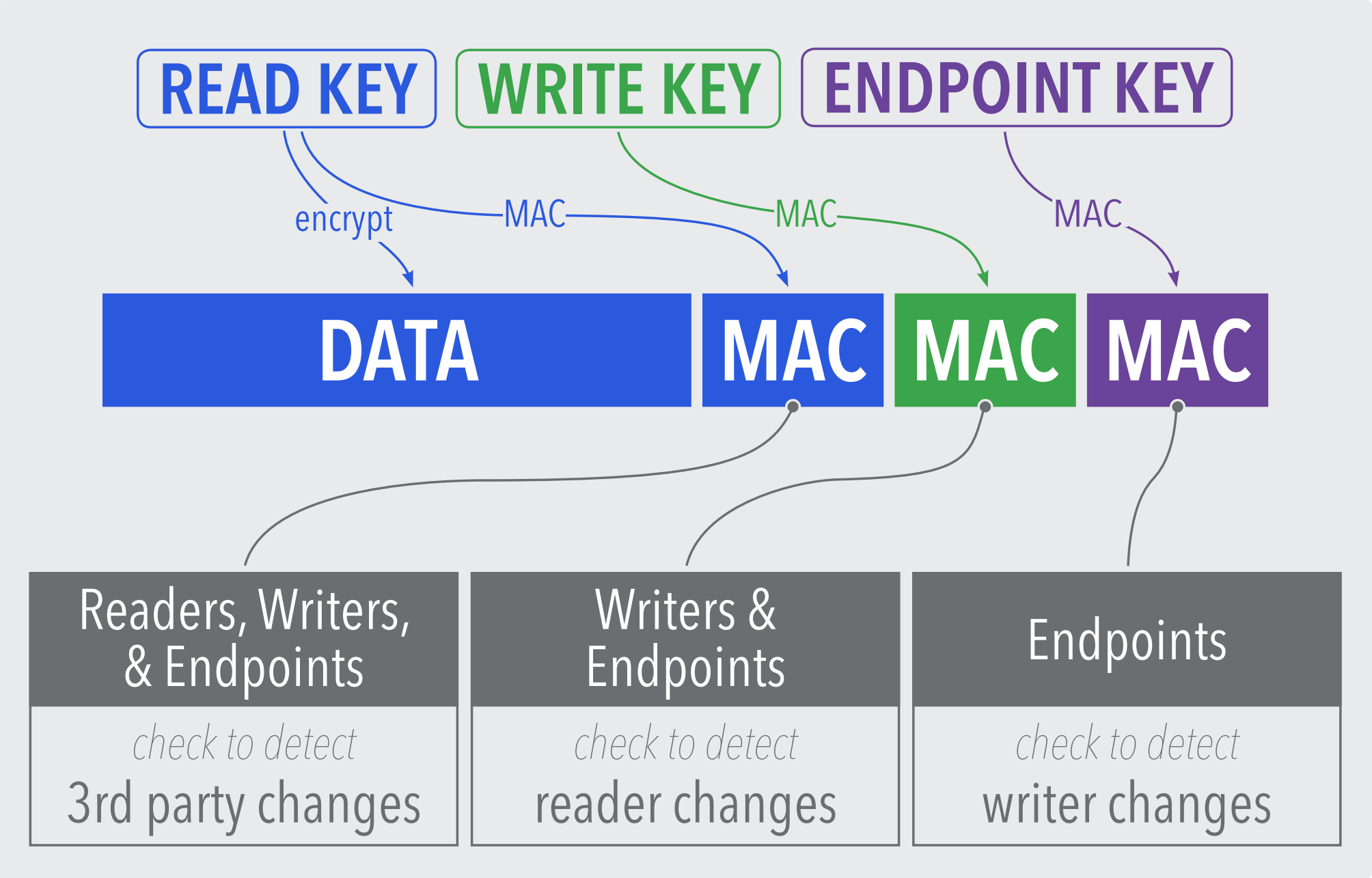


How do they work?

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.**

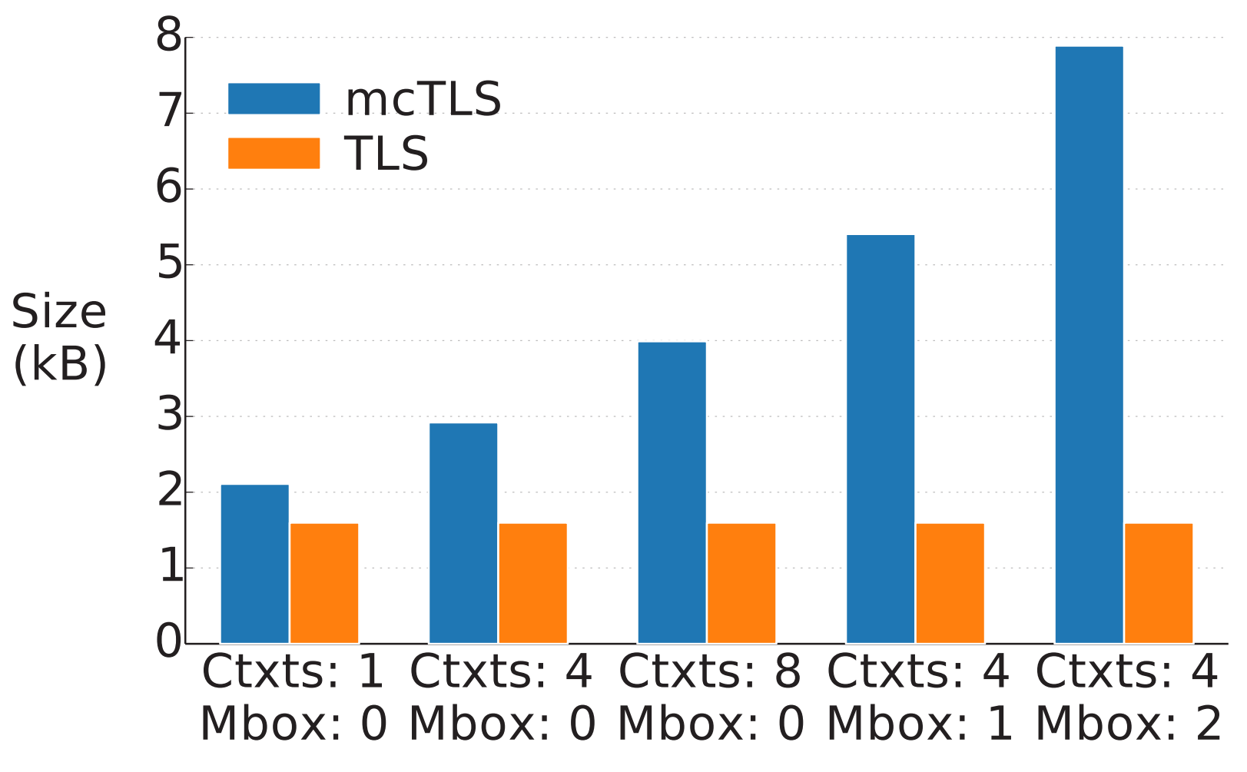


◀ *mcTLS Record*

Each record in mcTLS carries three MACs.

Read and write keys are per-context; the endpoint key is shared accross all contexts.

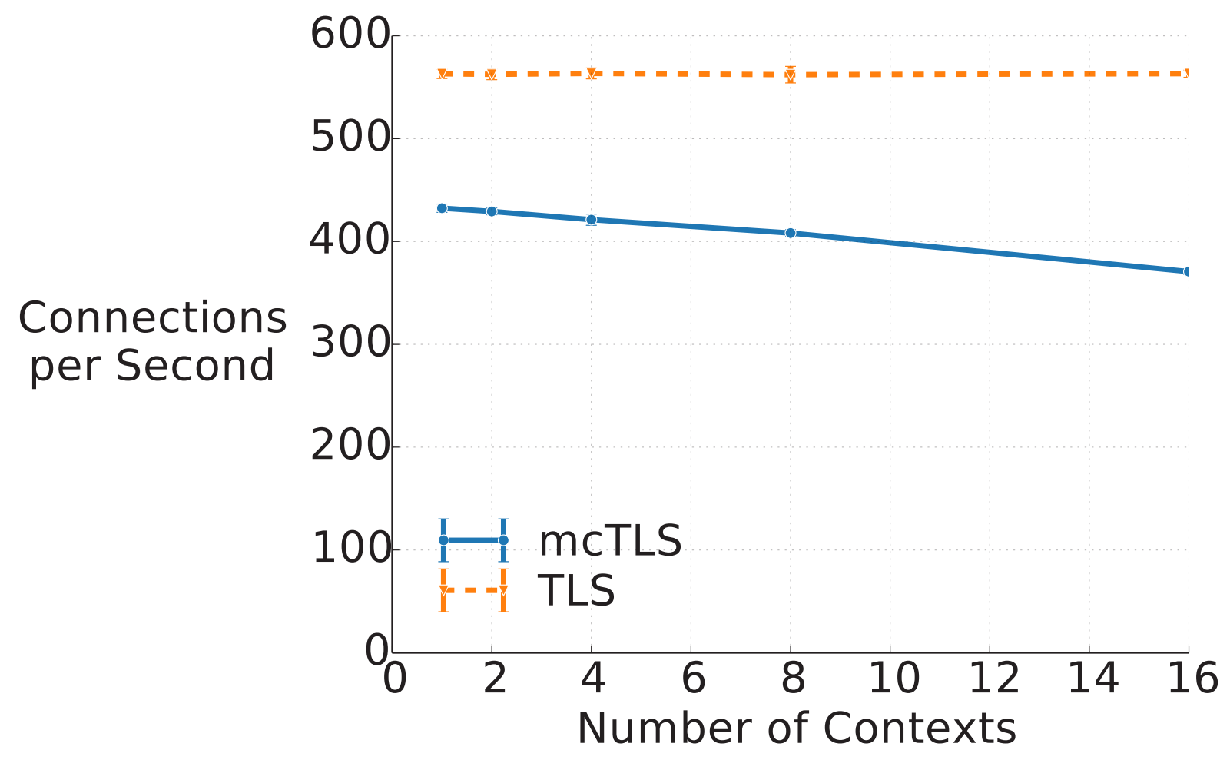
PERFORMANCE 〉 mcTLS adds functionality to TLS. Does it add overhead?



◀ *Handshake Size*

mcTLS introduces minimal data overhead.

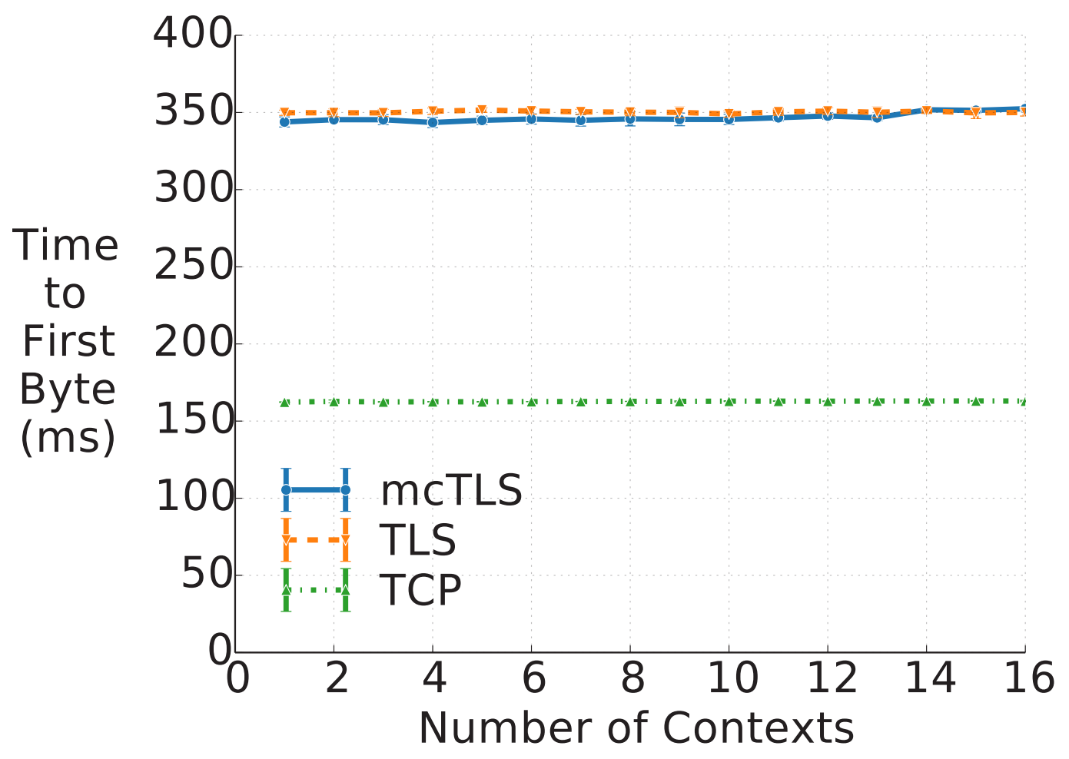
Handshake size increases with the number of contexts and middleboxes.



◀ *Server Load*

mcTLS introduces moderate CPU load.

The server can opt out of much of this extra computation.



◀ *Handshake Time*

mcTLS introduces no time overhead.

Just like TLS, the mcTLS handshake is 2 RTTs.

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

[✦] Carnegie Mellon University
[*] Case Western Reserve University
[•] Telefónica Research

Carnegie Mellon University Case Western Reserve University Telefónica