Enabling Security Analysis of IoT Device-to-Cloud Traffic

Eda Zhou
Department of Computer Science
Worcester Polytechnic Institute
Worcester, MA, USA
ezhou@wpi.edu

Joseph Turcotte
Department of Computer Science
Worcester Polytechnic Institute
Worcester, MA, USA
jaturcotte@wpi.edu

Lorenzo De Carli
Department of Computer Science
Worcester Polytechic Institute
Worcester, MA, USA
ldecarli@wpi.edu

***Abstract*—End-to-end encryption is now ubiquitous on the internet. By securing network communications with TLS, parties can insure that in-transit data remains inaccessible to collection and analysis. In the IoT domain however, end-to-end encryption can paradoxically *decrease* user privacy, as many IoT devices establish encrypted communications with the manufacturer's cloud backend. The content of these communications remains opaque to the user and in several occasions IoT devices have been discovered to exfiltrate private information (e.g., voice recordings) without user authorization.**

In this paper, we propose Inspection-Friendly TLS (IF-TLS), an IoT-oriented, TLS-based middleware protocol that preserves the encryption offered by TLS while allowing traffic analysis by middleboxes under the user's control. Differently from related efforts, IF-TLS is designed from the ground up for the IoT world, adding limited complexity on top of TLS and being fully controllable by the residential gateway. At the same time it provides flexibility, enabling the user to offload traffic analysis to either the gateway itself, or cloud-based middleboxes. We implemented a stable, Python-based prototype IF-TLS library; preliminary results show that performance overhead is limited and unlikely to affect quality-of-experience.

I. Introduction

The total number of Internet-connected devices is expected to reach 20 billion by 2020 [1]. Much of this growth is driven by Internet-of-Things (IoT) devices: smart, network-connected sensors and actuators such as cameras, thermostats, locks, and the such. These are deployed in residential and commercial settings to simplify and automate various everyday tasks.

Besides quality-of-life improvements, however, IoT devices also carry privacy risks. Personally identifiable information—such as usernames, passwords, and sensor streams–is not only stored on the devices themselves, but also in the network data these devices transmit. Fortunately, modern IoT devices tend to secure network connections using Transport Layer Security (TLS). This assurance of data privacy is especially important for IoT streaming devices used in households, such as smart home cameras and baby monitors, that transmit audio and video of a user's home with the expectation that unauthorized parties cannot access the streams.

In some cases, however, the cryptographic mechanisms intended to protect user privacy can have the opposite effect. Although end-to-end encryption protects data while in transit, this information cannot being inspected by any entities other than the two communicating endpoints—not even the device owner. In several cases, manufacturers have been suspected or determined to capture sensitive data beyond those necessary for devices to perform their function, and to stream these data to affiliate cloud endpoints. For example, it has been shown that streaming services utilize device tracking for advertising [2], and that home security systems contain hidden microphones [3]. Unfortunately, end-to-end encryption makes it impossible for the device user to detect unwanted leaks of private information from device-to-cloud backend. Furthermore, the fact that device communications remain opaque complicates the detection of security breaches where an attacker compromises and commandeers IoT devices (e.g., [4], [5]).

Currently, the commonly accepted approach is to sacrifice potential user privacy and security violations in favor of data security. Some approaches [6], [7] provide middlebox decryption capabilities in a secure manner but introduce impractical overhead for IoT devices. Others (e.g., [8]) use risky workarounds to give middleboxes decryption privileges; these approaches often introduce new security risks and challenges that outweigh the benefits of traffic inspection [9]. Given the significant limitations of current approaches, we take the alternative approach of a lightweight extension to TLS to specify middleboxes as secure entities in IoT communication.

In this paper, we describe Inspection-Friendly TLS (IF-TLS): a TLS-based protocol that preserves the secure data transit property from TLS, but also allows authenticated middleboxes to inspect the data while it is in transit. A challenge in providing this capability is to cover a broad range of user cases without excessively increasing communication overhead and penalizing quality-of-experience. IF-TLS' simple, streamlined design allows the user to retain control over which middleboxes have the ability to inspect traffic from each of their devices. Furthermore, these middleboxes can be located anywhere, including the local area network and the cloud. Finally, performance analysis shows that latency overhead remains limited and compatible with common use cases in the IoT world. **Overall, our work makes the following contributions:**

1) We describe the design of IF-TLS, an interception-friendly encrypted communication library designed for

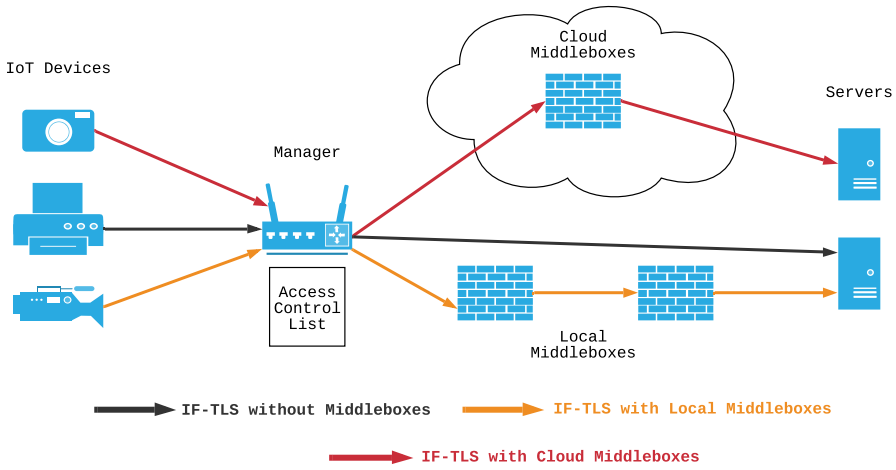

Fig. 1. High-level overview of IF-TLS

IoTs in residential contexts.

2) We implement a stable, proof-of-concept realization of IF-TLS which we plan to open-source to the public to foster further development and experimentation.
3) We analyze the performance of the IF-TLS prototype, highlighting that IF-TLS overhead is limited and unlikely to affect user experience.
4) We discuss potential limitations of our current design, and possible directions to extend it.

II. Background and Motivation

A. TLS Design

Transport Layer Security (TLS) is the standard protocol for creating an end-to-end secure connection using encryption. In August 2018, RFC 8446 specified the latest version of TLS, known as TLS 1.3 [10]. TLS 1.3 introduced a number of changes; the most notable is the removal of algorithms that do not provide forward secrecy (static RSA and Diffie-Hellman cipher suites).

These changes were intended to create a more secure Internet by removing algorithms that undermined the end-to-end nature of encryption. In particular, the removed cipher suites allowed a device to pre-share its private key with application-level firewalls that relied on inspecting un-encrypted data [11]. Scenarios which involve security analysis of payloads become impossible to handle since only the endpoints have access to plaintext.

Privacy advocates recognize that securing user communications from interception outweighs the reduced ability to perform security inspection. In the IoT domain however, the mapping between end-to-end encryption and user privacy is less straightforward. In many cases, end-to-end encryption enables device firmware to establish communication channels which remain opaque to the user.

B. IoT Privacy and Security

Many IoT devices can be characterized as "always-on" sensors that transmit information to the Internet. Researchers have raised concerns about their potential to violate user privacy. Ren et al. [12] found that some devices expose their device IDs, locations, and names. Moghaddam et al. [2] determined that many streaming services employ device tracking for advertising purposes, and that unique identifiers are often collected and transmitted unencrypted. Finally, Google revealed in February 2019 that its Nest home security system was equipped with a microphone; this fact had previously not been made explicit, raising concerns [3]. Since IoT devices regularly communicate to cloud backends to perform their legitimate function, and encrypt such communications, determining whether a device over-shares private information is difficult.

Many IoT devices also remain highly vulnerable to cyberattacks. Manufacturers have little economic incentive to perform proper security design [13], and may lack in-house expertise to do so. As a results, vulnerabilities are common [4], [14], some of which may result in the formation of large IoT botnets [15]. While traffic analysis can be deployed to identify misbehaving devices, its effectiveness is generally reduced by the inability to inspect encrypted traffic.

C. Threat Model

Our work contemplates the categories of threats above ("nosy manufacturer" and attackers commandeering IoT devices). Our goal is to enable controlled decryption of IoT flows by user-approved appliances, to improve the ability to detect such behaviors. We assume that the middleboxes to which the user delegates analysis are trusted not to willingly leak private data or share decryption keys. We furthermore assume that the appropriate response to suspicious network data is to drop the flow containing the data. Therefore, our approach enables third-parties to analyze packet payloads in unencrypted form, but not to modify data in-transit.

III. System Design

At a high level, IF-TLS works by sharing the IoT device and server session keys with authorized and trusted middleboxes

defined in an access control list (ACL). These middleboxes are permitted to inspect all traffic from a device. The user first provides an ACL that designates which middleboxes will be able to inspect traffic from their IoT devices to a dedicated component that we call the *IF-TLS manager*. Each IoT device using IF-TLS shares its client-server session key with the IF-TLS manager over a TLS connection. The IF-TLS manager then shares the device's session key with each authorized middlebox through TLS connections. Finally, all traffic from the IoT device is routed through the middleboxes by the manager. The overall functioning of IF-TLS is depicted in Figure 1.

A. Communicating Parties

Before we detail the protocol, we need to establish the communicating parties:

1) IoT devices: The smart devices whose traffic needs to be examined. We will refer to these as the clients who will share their session keys with the manager.
2) Manager: The device that runs the IF-TLS manager, responsible for configuring the middleboxes and the client. Most commonly, we expect the manager application to execute on the local network gateway. However, this is not required (e.g., it could reside on a specialized hardware device).
3) Middleboxes: The trusted middleboxes to which the user delegates inspection of IoT device traffic. While we expect the middleboxes to detect attacks and private information leaks, their specific function—beyond being able to decrypt data—is outside the scope of our project.

The rest of this section expands on the three steps in IF-TLS: initialization, key sharing, and data sending.

B. Initialization

The first step in IF-TLS is providing the access rules that determine which middleboxes will be able to decrypt which IoT devices' traffic. This information is supplied as an Access Control List (ACL) by the user. The ACL is a dictionary where the keys are IoT MAC addresses and the values are a list of middlebox IPs. The device traffic is sent to the middleboxes in the same order as they appear in the list. This means that IF-TLS enables offsourced traffic analysis at the granularity of individual devices.

We use IP addresses as unique identifiers for middleboxes because we need network layer addresses for forwarding packets to middleboxes outside the local network. These addresses need to be updated when they are changed. We expect middleboxes to be constantly running on a cloud instance or within the user's network, and thus to be less prone to being assigned new IP addresses.

IoT devices are identified in the ACL by their MAC addresses. MACs are locally unique to each device and we assume that they are immutable. A MAC-based access rule can be set for a device even before it is installed. This enables the user to have all traffic from a device, including setup, inspected.

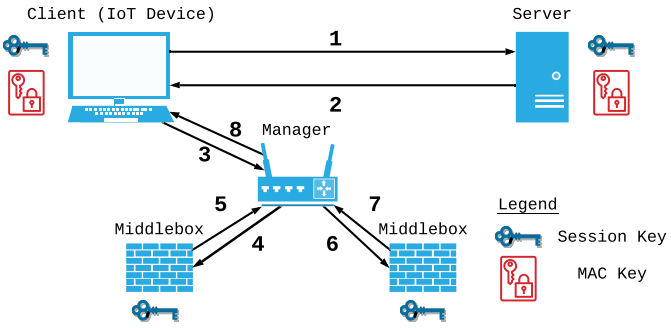

Fig. 2. IF-TLS session establishment details

The initialization process requires that the user defines an ACL for their configuration. In practice, we envision that traffic analysis will be provided by commercial entities, and such entities will provide pre-configured ACLs, and a mechanism to load them into the system, to users.

C. Session Establishment

Once IF-TLS is initialized, each new connection attempt by a local IoT device will initiate key sharing. There are three initializations that need to occur: the client/server, client/manager, and manager/middlebox initializations (summarized in Figure 2). The client/server initialization establishes the cipher suites and keys for the IF-TLS session. Then, during the client/manager initialization, the client shares its session key *pre-master secret* with the manager. Finally, in the manager/middlebox initialization the pre-master secret is distributed to the middleboxes.

1) Client/Server Initialization: An IoT device (client), begins by establishing a TCP connection with the server. Then, the following IF-TLS negotiation—similar to TLS, and abridged for brevity's sake—takes place (steps **1** and **2** in Figure 2):

1) **Cipher negotiation:** the client agrees with the server on the cipher suite to use, similar to TLS, and verifies the server.
2) **Pre-master exchange:** the client creates the pre-master secret, encrypts it with the server's public key, and sends it to the server. The server receives the pre-master secret, and computes the pre-master message from the client to create the session and MAC[1] keys.
3) **Acknowledgments:** The server sends an acknowledgment (ACK) message encrypted using the session key and MAC key to the client. The server is now ready to send and receive data through IF-TLS. The client computes the session and MAC keys from the pre-master and receives the ACK message from the server. Then, the client decrypts the message and verifies the message.

[1]In this subsection, we use the "MAC" acronym to refer to a **M**essage **A**uthentication **C**ode, and not to layer-II MAC addresses as in the rest of the paper.

2) Client/Manager Initialization: After the client/server initialization, the IoT device proceeds to a client/manager initialization in which the client shares its session key over TLS with the manager (this step is similar to LOCKS [16]). Since a device needs to resend its session key to the manager with every new generation, this connection may remain open between IF-TLS sessions.

In order to keep the packet length consistent, the client sends a part of the pre-master secret to the manager, rather than the computed key itself (step **3**). In particular, the part sent to the manager is the one used to generate the session key. The remaining part of the pre-master secret is used to generate the client/server MAC key and is not shared because it would give manager and middleboxes the ability to modify packets in transit. Once the manager finishes sharing the pre-master session key with the specified middleboxes, the manager sends an ACK to the client (step **8**).

3) Manager/Middlebox Initialization: The remaining initialization is between the manager and the middleboxes. The manager consults the ACL to determine which middleboxes need a device's session key. Then, the manager initializes each middlebox by creating a TLS session with that middlebox and sharing the device's information (MAC address, cipher used) and pre-master session key. The middlebox computes the session key and respond with an ACK message if the calculation is successful (steps **4-7**).

D. Data Sending

Once the session key has been shared, packets can be redirected through the middleboxes (Figure 1). The specific redirection mechanism is orthogonal to IF-TLS, and a variety have been proposed (e.g., [17], [18]) that can be adapted to our use case. For evaluation purposes, we deploy simple tunneling. When a packet arrives at the manager running on the local router, the manager matches the packet's source MAC against the ACL. If there is no match, the packet is forwarded normally. If there is a match, the list of middleboxes responsible for analyzing the packet is retrieved and the packet is forwarded to the first one (we assume that each middlebox in the chain is correctly configured to send data to the next one).

The packets that are routed through the middleboxes contain a message authentication code to ensure integrity. While the middleboxes are able to decrypt the payloads, they are unable to modify in-transit data without being detected by the receiving end. The intended use of IF-TLS is for middleboxes to block traffic containing excessive user information. Therefore, we do not allow packet content manipulation, but we do allow flows to be dropped in transit.

IV. Discussion

a) Access granularity: The IF-TLS design implicitly defines two access levels: no access (for any middlebox that does not appear in an ACL entry), or full decryption access (for every middlebox in an ACL entry). We chose this design because it allows maximum flexibility to middleboxes in deciding how to analyze traffic. If we were to enable partial decryption, as in BlindBox [6] we would have to pre-process a device's traffic to determine what is suitable for a middlebox to inspect, introducing an additional layer of complexity.

b) Analysis scope: While the server's traffic to the client is also sent over IF-TLS, it is not routed through middleboxes. This is intentional because our purpose is to enable users to view data emitted from their IoT devices. Incoming traffic is less likely to be sensitive. Furthermore, redirecting this traffic would require cooperation from the manufacturer's backend, a significant practical hurdle.

c) IP spoofing: The current design purposely does not perform middlebox authentication. This choice is reasonable if the middleboxes are either virtualized and running on the gateway itself, or on a remote device/cloud to which a secure channel exists. If this holds, further per-middlebox authentication would provide limited security gains and significantly increase overhead. However, we acknowledge that these assumptions may not hold true in other deployment scenarios where the attacker can inject arbitrary traffic to the gateway. For these scenarios, IF-TLS would need to be extended with a middlebox authentication capability, and a more robust notion of middlebox identity beyond IP addresses. We leave this extension as future work.

d) MAC spoofing: An attacker may perform MAC address spoofing. Doing so to imitate a known device gives no additional leverage, as it would simply cause the attacker's traffic to be routed through the middleboxes for analysis (and the IF-TLS manager never sends any confidential information to the IoT device). An attacker may also perform MAC spoofing in order to *evade* IF-TLS. Similar to middlebox authentication, device authentication would prevent this attack, but also add complexity and latency. An alternative is to employ device fingerprinting on the gateway[19] to the detect situations where a known device suddenly changes layer-II/III identifiers.

e) Denial of Service: We focus on a DoS against the IF-TLS manager, which we assume runs on a local router. While it is also possible to launch a DoS on a middlebox or server, those entities exist independently of IF-TLS and thus are orthogonal to our work. Should an attacker successfully disable the router, all devices connected to the router would be unusable. However, using IF-TLS will not make a router more vulnerable to conventional DoS attacks than it already is. Since the IF-TLS manager only performs lightweight dictionary look-ups to the ACL, the vulnerability of a router to DoS remains roughly the same regardless of whether it runs the IF-TLS manager or not.

f) Component Security: The security of IF-TLS depends heavily on the security of its components: the IoT device, the manager/router, the middleboxes, and the server. By virtue of enabling middleboxes to access traffic, IF-TLS introduces inherent risks related to information disclosure that do not exist when using TLS 1.3. Adding middleboxes to the path increases the attack space that adversaries can exploit. However, the risk

of compromising a well-established cloud middlebox, such as one running on AWS or Google Cloud, is low.

g) Incentives: As IoT devices become commonplace, we expect that purchasers of IoT devices will become more conscious of related privacy issues. Thus, devices implementing features such as IF-TLS will become more attractive to consumers, incentivizing manufacturers to implement these features. We also note that IoT data collection has, in many cases, legitimate and useful applications. IF-TLS does not prevent it. Instead, it encourages manufacturers to limit the scope of data collection to the strictly necessary, and to be transparent about the nature of collected data.

V. Evaluation

A. Implementation, Testbed, and Traces

We implemented our IF-TLS prototype as a Python library consisting of 800 lines of code. The user-facing API consists of a single Python class that can be used to instantiate an IF-TLS session on either the client- or server-side (we omit details)[2]. We chose Python for its ease of use and availability of support libraries, as our goal is to evaluate the viability of IF-TLS. However, we acknowledge that many IoT devices are programmed in C/C++. We estimate the engineering efforts to translate the Python module to C++ to be fairly limited.

In our testbed, we implemented four communication scenarios: TLS (i.e., no IF-TLS), IF-TLS w/o middleboxes, IF-TLS w/ local middleboxes, and IF-TLS w/ cloud-based middleboxes. IF-TLS w/ local middleboxes is representative of a scenario where middlebox processing executes on the local network, while IF-TLS w/ cloud middleboxes represents a scenario where traffic processing is offloaded to the cloud. The local network was simulated by implementing IoT device (client), manager/gateway and a local middlebox as TinyCore Linux VMs running on an 8GB 2015 Apple Macbook Laptop. The server was implemented on a TinyCore VM executing on a different home network and running on an 8GB 2015 HP Notebook Laptop. A cloud-based middlebox was implemented as an Amazon EC2 t2.micro instance based on the Amazon Linux AMI template. To avoid making our results dependent on a specific type of middlebox analysis, our test middlebox implements a passthrough appliance that forwards all received packets.

For our RTT measurements we replayed a set of three traces captured in an affiliated institution's IoT lab, summarized in Table I. Each trace represents a full TCP session by an IoT devices. We picked these traces as they are heterogeneous both in terms of originating devices, and length/number of packets.

B. IF-TLS Initialization Overhead

Our first performance result highlights the total initialization overhead for IF-TLS; this includes the client-server, client-manager, and manager-middlebox initialization times. The initialization overhead is defined as the time elapsed between the client initiating the establishment of an IF-TLS session, and the establishment of the session as signaled by the reception of an acknowledgment from the server (all events are timestamped on the client). We consider three IF-TLS scenarios: IF-TLS w/o middleboxes in the path and IF-TLS with a middlebox located respectively on the local gateway and in the cloud. For comparison, we also measure the TLS session initialization time between the same machines. Each measure was repeated 30 times (we report averages).

Device	Capture Size [# packets]	Session Length [s]
Google Home Mini 1	24	0.5
Arlo Q Camera	1070	39.5
Arlo Q Camera	2052	12.1

TABLE I
Traces for RTT evaluation

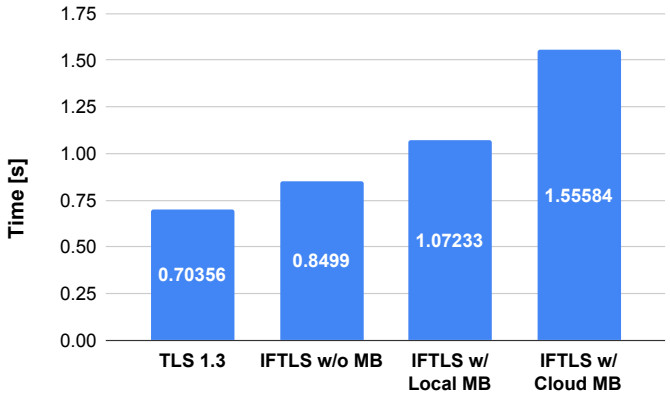

Fig. 3. Average initialization time for the test scenarios

Figure 3 shows the average amount of time it took for the initialization procedure to complete under TLS and the three IF-TLS scenarios. On average, IF-TLS without a middlebox adds 21 percent additional delay to the initialization procedure compared to TLS 1.3. The percentage increases for IF-TLS with a local middlebox and cloud-based middlebox are 52% and 121%, respectively.

C. Initialization Overhead Breakdown

In addition to analyzing the total initialization time before data sending, we also measured the individual components of the initialization across the three IF-TLS scenarios. Figure 4 shows each component's proportion of the total initialization time for these scenarios; from left to right, the client-server initialization makes up 91, 76, and 75 percent of the initialization procedure[3]. The absence of a middlebox in the first IF-TLS scenario implies that the client-manager initialization consists of a simple lookup in the access control list, followed by an acknowledgment to the client that all middleboxes (in this case, 0) have computed the session key.

The latter two measurements are representative of having one middlebox in the path; thus, the client-manager proportion

[2]The code can be accessed at: https://www.dropbox.com/sh/dhk67rc7jdwtdhh/AACo0a8xT-19v32ehkpYZ4wca

[3]The components do not exactly sum to the totals of Figure 3, as those also account for a few auxiliary operations that are not part of either initialization.

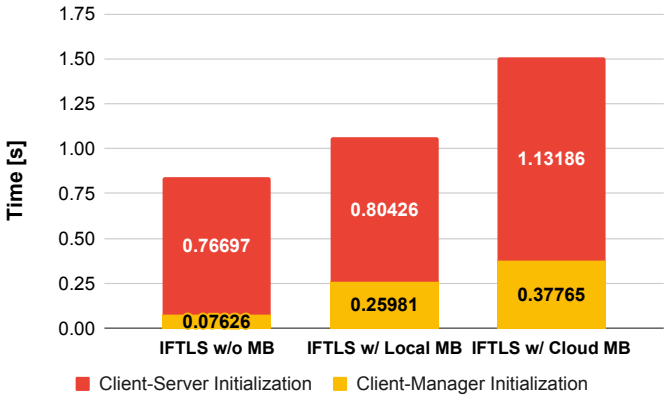

Fig. 4. Breakdown of initialization times in IF-TLS

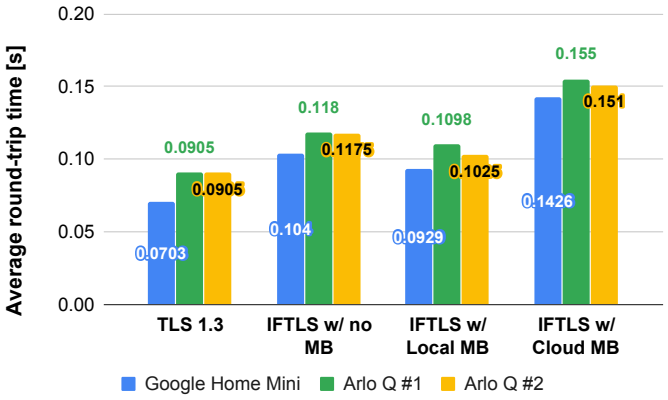

Fig. 5. RTT comparison

Middlebox Location	Client-Manager init time [s]	Manager-Middlebox init time [s]
Local	0.260	0.182
Cloud	0.377	0.297

TABLE II
CLIENT-MANAGER INITIALIZATION DETAILS

will increase by a roughly constant factor as more middleboxes are added to the path. Table II shows the overhead we obtained by timing the manager-middlebox initialization time on the manager. This data shows that the manager-middlebox initialization, which is a sub-component of the client-manager initialization, takes up a significant percentage of the latter. The additional overhead for a middlebox in the local network and the cloud is roughly 0.2s and 0.3s, respectively. The cloud-related overhead is expected to vary slightly depending on the physical location of the cloud server.

D. RTT analysis

Beyond initialization, it is important to characterize the impact of IF-TLS on the performance of an ongoing data transfer. By introducing additional forwarding steps, IF-TLS may increase a connection's round-trip time (RTT) with possible impact on both congestion control and the perceived QoE of interactive traffic.

Figure 5 shows a comparison of the average RTTs for the four scenarios. Compared to TLS 1.3, the averaged RTT increases amount to (left to right) 36%, 22% and 80%. IF-TLS with a local middlebox is generally comparable to IF-TLS without middleboxes; as expected the effect of routing through a local appliance is negligible.

The RTT will increase with more middleboxes in the connection. Assuming that middleboxes performs passive monitoring (i.e. they do not hold packets for processing), the only factors contributing to the overhead are packet forwarding operations between middleboxes. This overhead is negligible if the middleboxes are co-located, but may be high if the middleboxes are geographically distributed. Commercial services deploying cloud-based analysis may need to perform optimization of their network topology to minimize this overhead.

E. Discussion of Results

One of our goals in our performance testing was to determine whether the additional delay introduced by IF-TLS would still allow for an acceptable user experience with an IoT device, even when placing a middlebox in the cloud. One accepted metric for how long users are willing to wait for loading a web page is 2 seconds [20]; this is analogous to the IF-TLS initialization time, which does not exceed 2 seconds in the case of a single cloud-based middlebox. Furthermore, the majority of data transmitted using IoT devices is sensor or streaming data, and does not require frequent interaction, such as clicking between web pages, with the user. Other studies [21] [22] have shown that the frequency of interaction with IoT devices is not high enough for a user to notice a few seconds of initialization overhead, which thus does not negatively impact the quality of the user experience. Based on the initialization and round-trip time data we obtained, we conclude that IF-TLS can still perform reasonably with a series of cloud-based middleboxes.

Our second goal was to determine the performance impacts of placing a middlebox in the local network versus placing it in the cloud. In the previous three sections, we observed that placing a middlebox in the cloud resulted in higher initialization and data sending times; thus, utilizing the cloud presents a performance trade-off between cloud-based benefits, such as scalability and reduced IT costs, and a faster connection.

VI. Related Work

Naylor et al. [7] propose Multi-Context TLS (mcTLS) to provide decryption capabilities to middleboxes without requiring the client to install a root certificate. The notion of an encryption "context", or a set of symmetric encryption and message authentication code (MAC) keys, functions as an access control mechanism that allows for flexible configuration of middlebox decryption capabilities. Endpoints can limit the kinds of data to which a middlebox has decryption access, define per-flow decryption rights, and restrict permissions to read-only for the purpose of preventing illegal data modifications. By specifically restricting its scope to the IoT domain, IF-TLS can define a much simpler design. For a cryptographic

session layer, there is strong evidence that keeping complexity to a minimum is crucial to ensure correct implementation and simplifying audits [23]. Furthermore, we believe that the capabilities we provide are sufficient for practical use cases.

Bierma et al. [16] introduce Locally Operated Cooperative Key Sharing (LOCKS), a mechanism that allows clients to share TLS keys with a trusted agent, such as a security monitoring system, inside an enterprise network. While our design shares some aspects with LOCKS, their approach specifically targets enterprise environments and would not directly work with middleboxes in the cloud.

Sherry et al. [6] propose BlindBox, a system that allows DPI on encrypted traffic. BlindBox utilizes searchable encryption to inspect encrypted traffic using keywords or according to predefined rules. This mechanism restricts the capabilities of middleboxes to inspect and analyze user data. However, the connection setup happens on the order of minutes for large IDS systems with thousands of search rules, and thus is impractical for use on smart devices.

VII. Conclusion

While IoT-originated network communications can benefit from encryption, it is important to guarantee that users retain the ability to inspect the traffic that their devices generate. This paper put forward an initial proposal to do so. We presented IF-TLS, a simple session-layer protocol to encrypt in-transit data while ensuring user-controlled middleboxes remain able to inspect it. IF-TLS is designed to be simple and generate very limited overhead compared to conventional TLS. While additional features will be necessary to support all use cases (e.g. middlebox authentication), we hope IF-TLS can act as a "conversation starter" for the design of encrypted protocols that do not make IoT communications opaque to the user.

References

[1] R. van der Meulen. (2017) Gartner says 8.4 billion connected "things" will be in use in 2017, up 31 percent from 2016. [Online]. Available: https://www.gartner.com/en/newsroom/press-releases/2017-02-07-gartner-says-8-billion-connected-things-will-be-in-use-in-2017-up-20-31-percent-from-2016

[2] H. M. Moghaddam, G. Acar, B. Burgess, A. Mathur, D. Y. Huang, N. Feamster, E. W. Felten, P. Mittal, and A. Narayanan, "Watching you watch: The tracking ecosystem of over-the-top tv streaming devices," in *CCS*, 2019.

[3] S. Fussell. (2019, February) The microphones that may be hidden in your home. [Online]. Available: https://www.theatlantic.com/technology/archive/2019/02/googles-home-security-devices-had-hidden-microphones/583387/

[4] S. Nichols, "Don't panic, but your baby monitor can be hacked into a spycam," Jun. 2018. [Online]. Available: https://www.theregister.co.uk/2018/06/22/baby_monitor_hacked/

[5] C. Cimpanu, "Hacker leaks passwords for more than 500,000 servers, routers, and IoT devices," Jan. 2020. [Online]. Available: https://www.zdnet.com/article/hacker-leaks-passwords-for-more-than-500000-servers-routers-and-iot-devices/

[6] J. Sherry, C. Lan, R. A. Popa, and S. Ratnasamy, "Blindbox: Deep packet inspection over encrypted traffic," *SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 4, pp. 213–226, Aug. 2015.

[7] D. Naylor, K. Schomp, M. Varvello, I. Leontiadis, J. Blackburn, D. R. López, K. Papagiannaki, P. Rodriguez Rodriguez, and P. Steenkiste, "Multi-context tls (mctls): Enabling secure in-network functionality in tls," *ACM SIGCOMM Computer Communication Review*, vol. 45, no. 4, pp. 199–212, 2015.

[8] V. Zakharevich and M. Rakhmanov, "Intercepting ssl and https traffic with mitmproxy and sslsplit," Apr 2016. [Online]. Available: https://www.trustwave.com/en-us/resources/blogs/spiderlabs-blog/intercepting-ssl-and-https-traffic-with-mitmproxy-and-sslsplit/

[9] Z. Durumeric, Z. Ma, D. Springall, R. Barnes, N. Sullivan, E. Bursztein, M. Bailey, J. A. Halderman, and V. Paxson, "The security impact of https interception." in *NDSS*, 2017.

[10] E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.3," Internet Requests for Comments, RFC Editor, RFC 8446, August 2018. [Online]. Available: https://tools.ietf.org/rfc/rfc8446.txt

[11] F. Andreasen, N. Cam-Winget, and E. Wang, "Tls 1.3 impact on network-based security," Working Draft, IETF Secretariat, Internet-Draft draft-camwinget-tls-use-cases-00, October 2017, http://www.ietf.org/internet-drafts/draft-camwinget-tls-use-cases-00.txt. [Online]. Available: http://www.ietf.org/internet-drafts/draft-camwinget-tls-use-cases-00.txt

[12] J. Ren, D. J. Dubois, D. Choffnes, A. M. Mandalari, R. Kolcun, and H. Haddadi, "Information exposure from consumer iot devices," in *IMC*, 2019.

[13] B. Schneier. (2016) Security economics of the internet of things. [Online]. Available: https://www.schneier.com/blog/archives/2016/10/security_econom_1.html

[14] G. Acar, D. Y. Huang, F. Li, A. Narayanan, and N. Feamster, "Web-based Attacks to Discover and Control Local IoT Devices," in *IoT S&P*, 2018.

[15] M. Antonakakis, T. April, M. Bailey, M. Bernhard, E. Bursztein, J. Cochran, Z. Durumeric, J. A. Halderman, L. Invernizzi, M. Kallitsis, D. Kumar, C. Lever, Z. Ma, J. Mason, D. Menscher, C. Seaman, N. Sullivan, K. Thomas, and Y. Zhou, "Understanding the Mirai Botnet," in *USENIX Security Symposium*, 2019.

[16] M. Bierma, A. Brown, T. DeLano, T. M. Kroeger, and H. Poston, "Locally operated cooperative key sharing (locks)," in *ICNC*, 2017.

[17] J. Sherry, S. Hasan, C. Scott, A. Krishnamurthy, S. Ratnasamy, and V. Sekar, "Making middleboxes someone else's problem: network processing as a cloud service," *ACM SIGCOMM Computer Communication Review*, vol. 42, no. 4, pp. 13–24, 2012.

[18] C. R. Taylor, T. Guo, C. A. Shue, and M. E. Najd, "On the feasibility of cloud-based sdn controllers for residential networks," in *2017 IEEE Conference on Network Function Virtualization and Software Defined Networks*, 2017.

[19] B. Bezawada, M. Bachani, J. Peterson, H. Shirazi, I. Ray, and I. Ray, "Behavioral Fingerprinting of IoT Devices," in *ASHES*, New York, NY, USA, 2018.

[20] F. F.-H. Nah, "A study on tolerable waiting time: how long are web users willing to wait?" *Behaviour & Information Technology*, vol. 23, no. 3, pp. 153–163, 2004.

[21] C. Axel, G. Ravindra, and O. W. Tsang, "Towards characterizing users' interaction with zoomable video," in *SAPMIA*, 2010.

[22] N. Q. M. Khiem, G. Ravindra, and W. T. Ooi, "Towards understanding user tolerance to network latency in zoomable video streaming," in *Proceedings of the 19th ACM International Conference on Multimedia*, 2011.

[23] S. Schmidt, "Introducing s2n, a New Open Source TLS Implementation," Jun. 2015. [Online]. Available: https://aws.amazon.com/blogs/security/introducing-s2n-a-new-open-source-tls-implementation/