Trust but Verify: Auditing the Secure Internet of Things

Judson Wilson Riad S. Wahby Henry Corrigan-Gibbs
Dan Boneh Philip Levis Keith Winstein
{judsonw, rsw, henrycg, dabo, pal, keithw}@cs.stanford.edu
Stanford University

ABSTRACT

Internet-of-Things devices often collect and transmit sensitive information like camera footage, health monitoring data, or whether someone is home. These devices protect data in transit with end-to-end encryption, typically using TLS connections between devices and associated cloud services.

But these TLS connections also prevent device owners from observing what their own devices are saying about them. Unlike in traditional Internet applications, where the end user controls one end of a connection (e.g., their web browser) and can observe its communication, Internet-of-Things vendors typically control the software in both the device and the cloud. As a result, owners have no way to audit the behavior of their own devices, leaving them little choice but to hope that these devices are transmitting only what they should.

This paper presents TLS–Rotate and Release (TLS-RaR), a system that allows device owners (e.g., consumers, security researchers, and consumer watchdogs) to authorize devices, called auditors, to decrypt and verify recent TLS traffic without compromising future traffic. Unlike prior work, TLS-RaR requires no changes to TLS's wire format or cipher suites, and it allows the device's owner to conduct a surprise inspection of recent traffic, without prior notice to the device that its communications will be audited.

1. INTRODUCTION

The Internet of Things (IoT) is notoriously insecure [34,65]. Recent exploits have shown a drone flying by a building to take control of its lights [56], and tens of thousands of compromised webcams were behind recent denial-of-service attacks against PayPal, Twitter, Netflix, and other prominent sites [14]. A recent study by HP Labs found that cleartext data, insecure firmware updates, and poor cryptographic practices mean that a substantial majority of devices had exploitable security flaws [38].

One well-accepted way to improve IoT security is to use Transport Layer Security (TLS). Products from Nest [3] and Samsung SmartThings, for example, use TLS to connect to their respective cloud services. TLS provides useful guarantees: message integrity and confidentiality, and mutual authentication of devices and servers.

However, the use of strong encryption on a locked-down consumer device has a worrisome effect for privacy: you, the device owner, cannot tell what your own devices are reporting about you. For example, if you install a Nest thermostat or camera in your home, you cannot observe the contents of its traffic to verify that it's only sending data of the kind the vendor has promised.

Internet-of-Things applications pose new security and privacy concerns because *both* ends of a secure connection are controlled by a single party: the vendor. While a Nest thermostat runs Linux, the owner cannot log in to it or otherwise control its operation. Because you cannot modify the thermostat's firmware, trace its applications, or intercept its unencrypted traffic, you cannot see what a device is reporting about you and your home.

The IoT therefore marks a break from the tradition, essentially as old as the Internet, of end users being able to inspect their own communications. Web browsers generally have a developer interface that lets end users see the contents of network traffic. And with local control of the operating system, one can look inside TLS streams by installing the public key of a trusted middlebox as a root certificate. IoT devices generally do not allow these.

While it may be rare that any given consumer will want to inspect the contents of their apps' encrypted traffic, the *ability* to do so has allowed consumer watchdogs and security researchers to uncover undisclosed exposures of personal information [5,15,25,34,67], and by reporting or publicizing these exposures, cause manufacturers to fix them.

At the same time, IoT manufacturers are understandably reluctant to provide a means to weaken a device's security by installing a new root certificate, which allows the holder to act as a man-in-the-middle between the device and the cloud, modifying traffic or impersonating one to the other. Manufacturers have also shown little interest in proposals, such as mcTLS [45], that diverge from the TLS protocol by adding additional keys and message-authentication codes to allow "read-only" middleboxes. These modified protocols are not supported by TLS terminators and load balancers in the cloud, which are often out of a vendor's control.

In this paper, we present a mechanism, TLS-Rotate and Release (TLS-RaR), which allows device owners to audit their devices' traffic without compromising application integrity, and without modifying the TLS format on the wire. Using

MobiSys '17, June 19–23, 2017, Niagara Falls, NY, USA.
© 2017 Copyright held by the owner/author(s). Publication rights licensed to ACM. ISBN 978-1-4503-4928-4/17/06. . . $15.00
DOI: http://dx.doi.org/10.1145/3081333.3081342

TLS-RaR, TLS connections between an IoT device and the cloud retain their end-to-end integrity and remain encrypted as before, but an owner can optionally install any number of read-only, in-home **auditors**, low-cost devices on the owner's home network that passively sniff the ciphertext of TLS sessions.

TLS-RaR's key idea is that at any moment, the user can log in to the cloud service and instruct the device to *rotate* a TLS connection's symmetric encryption keys. Once the device is certain that both it and the server will no longer use the old keys, it securely *releases* the superseded keys to the user or the user's designated auditor. Using these keys, the auditor decrypts and verifies its cache of past ciphertext. As a result, the auditor receives delayed access to the same session keys as the TLS endpoints, and therefore the same integrity-protected plaintext, even from before the endpoints were informed about the auditor's existence; but the auditor cannot forge data or masquerade as either party.

Compared with approaches like mcTLS, TLS-RaR is simpler and more deployable. TLS-RaR requires no modifications to TLS's wire format, state machines, or the server-side TLS endpoint. This last point is beneficial because large-scale TLS deployments often offload server-side TLS termination to middleboxes or specialized hardware. Changing TLS software or hardware on the server side is difficult, and existing systems may be architected such that the TLS and application layers are independent. We found (§5.3) that TLS-RaR can already be used today with most TLS servers in the wild.

TLS-RaR differs in other respects from previous proposals to audit encrypted communications. It works with passive auditors that do not have to act as men-in-the-middle. Auditors are guaranteed to see the same payloads and perform the same integrity checks as the TLS endpoints. Endpoints cannot send a different stream to auditors than they do to each other, in order to conceal malicious traffic. Multiple auditors can observe encrypted streams in parallel, without being able to confuse one another.

TLS-RaR also carries major limitations. First, auditors only learn the plaintext after some delay. TLS-RaR cannot be used to detect and prevent compromise in real time. Second, TLS-RaR simply allows an auditor to see inside the encrypted stream provided by TLS. TLS-RaR does not, by itself, detect or defeat additional layers of encryption or covert channels within a TLS stream. Finally, TLS-RaR of course requires the vendor's cooperation. Manufacturers who are not interested in subjecting their devices' communications to independent audits are unlikely to implement TLS-RaR.

Nonetheless, we believe the benefits of IoT audits would be considerable. We envision independent watchdogs (e.g., Consumer Reporters, Underwriters Laboratories, security firms, independent researchers, and academics) purchasing a random sample of popular devices off the shelf, and using TLS-RaR to verify that they communicate as they should.

For example, Amazon states that its Alexa devices listen to audio all the time, but only transmit recordings that are made while the light is on (and shortly before the light turned on) [1]. Is this statement true? Can it be independently verified, and monitored, as the software on these devices is updated over time? We believe that IoT manufacturers ought to welcome the idea of independent audits to verify that their devices are communicating as advertised, just as they allow financial auditors to inspect their warehouses and financial statements. In both cases, such audits build confidence and allow consumers or shareholders to separate good products—and companies—from bad ones.

This paper makes these contributions:

1. The design of TLS-RaR, including secure protocols for key rotation and for key release.
2. Measurements of TLS-RaR's existing support across a survey of popular websites.
3. Measurements of the communications of 5 IoT devices.
4. An evaluation of TLS-RaR's effect on application traffic using an implementation of TLS-RaR based on OpenSSL 1.0.2 running on embedded hardware, and an auditor based on tshark.

The rest of this paper is structured as follows. Section 2 describes requirements for IoT auditing. Section 3 gives an overview of TLS-RaR's operating model and approach. Sections 4.1 and 4.2 present the TLS-RaR protocols for rotation and release, respectively. Section 5 evaluates TLS-RaR's compatibility with existing servers, the performance of a prototype implementation, and the behavior of existing IoT devices. An auditor may have significant ethical and surveillance implications; we discuss these in Section 6, and limitations of the approach in Section 7. Section 8 surveys related work.

2. AUDITING IOT DEVICES

TLS-RaR's goal is to enable *auditing*—after-the-fact inspection of traffic—without modifying TLS or giving auditors man-in-the-middle access to encrypted channels. To that end, this section defines four requirements for a secure and practical IoT auditing system: past auditability, present-moment integrity, audit robustness, and compatibility with TLS. It also describes the threat model against which the security notions are defined, and discusses limitations.

We use the following notation for a TLS session when describing these requirements. A TLS session uses a set of symmetric *session keys* for its ciphers; the number and size of keys is determined by the TLS cipher suite, which is chosen by the the client and server endpoints independent of the auditing system. We refer to keys and their lifetime in terms of *epochs*. In an epoch t, a client and server communicate using session keys k_t. TLS allows the client and server to generate new keys, whereupon epoch $t+1$ begins. In practice, the client and server can generate new keys asynchronously, so the endpoints' notions of the current epoch may differ transiently. This means that epochs are defined with respect to a sequence of TLS records rather than with respect to wall-clock time.

2.1 Requirements

In this section, we outline four requirements for an IoT auditing system. The first three, past auditability, present-moment integrity, and audit robustness, are security properties; we define the corresponding threat model in Section 2.2. The fourth requirement, TLS compatibility, responds to the practical realities of deploying new protocols using existing infrastructure.

Past Auditability. The primary goal of an IoT auditing system is allowing the owner of an IoT device to empower one or more *auditors* to decrypt and verify past traffic to and from that IoT device. Informally, auditors should be able to examine the plaintext sent through a TLS connection after some delay, provided that the auditor captured the encrypted traffic from the network.

More formally, if a TLS session is in epoch t, an auditor with all of the encrypted records in each of the epochs $T \subseteq \{0, \ldots, t-1\}$ will either recover the exact sequence of plaintext bytes sent for all epochs $t' \in T$ (after some delay; §2.3), or the auditor will output "failure."

Present-Moment Integrity. An auditor should not violate the integrity of communication between device and server: the auditor must not add, drop, or modify TLS traffic without detection. This ensures that an auditor cannot mount attacks against the device or the server. This protection serves two purposes. First, it prevents the owner of an IoT device from disrupting the device's proper functioning (for example, by falsifying water metering information). Second, it means that a compromised auditor cannot be used to attack devices, their owners, or their manufacturers (e.g., by spoofing an unlock command to a smart door lock).

Formally, if a device believes the current epoch to be t_d and the server believes it to be t_s, then an auditor cannot produce a valid TLS record for any epoch t' such that $t' \geq \min\{t_c, t_s\}$. Note that auditors can forge records for past epochs, but the device and server will ignore these.

Audit Robustness. The data produced by an audit whose output is not "failure" should be correct: no adversary can cause an auditor to output plaintext that differs from what was sent between the device and server. This ensures that malicious third parties, including compromised auditors, cannot falsely implicate IoT devices, and that endpoints cannot persuade auditors to accept incorrect plaintexts.

Formally, if an auditor has the all of the keys released by the device for epochs $T \subseteq \{0 \ldots t\}$ (but not necessarily the corresponding TLS records), a third party cannot produce any TLS record r such that the auditor falsely believes that the device or server sent r in epoch $t' \in T$ when the device or server did not do so.

TLS Compatibility. To be practically deployable, an auditing system should be deployable using existing infrastructure. This means that the system should not require changes to the TLS protocol or wire format: the former risks introducing security vulnerabilities, while the latter might cause incompatibility with middleboxes. More generally, the system should not require changes to the server's implementation, because many deployments use special-purpose hardware to terminate the server's end of TLS connections (say, to accelerate expensive cryptographic operations). On the other hand, the client is an IoT device controlled by the manufacturer, so changes to its software (provided they comply with the above) are acceptable.

Finally, the auditing system must maintain the forward-secrecy properties [39] of the TLS cipher suite that the client and server negotiate. (Roughly speaking, a session has forward secrecy if future key compromises do not reveal past plaintext.)

Non-Requirement: Real-Time Auditing. As a consequence of satisfying these four requirements, TLS-RaR does not give auditors the ability to decrypt and verify traffic in real time. Doing so using unmodified TLS ciphersuites (required for TLS compatibility) would require giving an auditor the current traffic keys (which would violate present-moment integrity). Compared with real-time monitoring, past auditability is weaker, but still sufficient to detect improper behavior. The auditor can choose when—and how often—to ask the device to rotate keys and release them to the auditor, but even if the auditor asks for a rotation and receives keys every second, it will always be slightly behind.

2.2 Threat and Network Model

As in a standard TLS session, there are two endpoints (an IoT device and a server). We assume that the endpoints cooperate to realize the same security as TLS against all parties other than auditors, which can audit past epochs as specified above; we discuss auditors below. Endpoints may attempt to undermine past auditability, but we consider some attacks out of scope (see §2.3).

We make standard cryptographic assumptions, and assume all of the standard threats against TLS. The network, may drop, delay, or reorder packets, even adversarially. Attackers may use passive or active attacks.

Auditors distrust the TLS endpoints they are auditing, and they distrust other auditors—either may attempt to undermine audit robustness. Other than allowing auditors to observe past epochs' plaintext, TLS endpoints distrust auditors. Auditors may attempt to undermine present-moment integrity or audit robustness, except that we do not consider denial of service attacks. Device owners and auditor manufacturers may attempt to disrupt or corrupt IoT applications via auditors (i.e. by attempting to undermine present-moment integrity), except that auditors divulging plaintext from past epochs is out of scope.

2.3 Limitations of the Auditing Model

Auditing is a useful means of detecting incorrect, insecure, or undesirable operation. For example, devices may be compromised; being able to audit their traffic provides a limited but useful check on their behavior. And while auditing does not protect devices from compromise, it may indirectly improve their security by obviating man-in-the-middle proxies.

On the other hand, auditing is not a panacea. Being able to audit TLS traffic does not ensure that a device's owner can see all data sent by that device, or that a device's manufacturer acts in good faith. As examples, a device's manufacturer could use steganography to hide data in innocent-looking exchanges or even install a hidden wireless modem in the device. These *covert channels* are a broad and complementary issue; we leave defending against covert-channel attacks [11, 31, 36, 40, 47, 60] out of scope. Likewise, a device might refuse to allow auditing—though an auditor can detect (and should report) missed audits.

Finally, as mentioned in Section 2.2, users trust auditors with their private information. Auditors might divulge this information or use it for mischief. For example, an auditor might spoof a device's application-layer credentials if those credentials are not resistant to replay attacks. Applications can address this issue, e.g., by using TLS client certificates.

3. SYSTEM OVERVIEW

This section considers three straw man proposals for providing the four properties described in Section 2 and describes a high-level overview of TLS-RaR, including relevant parts of the TLS key exchange protocol.

3.1 Straw Man: TLS Inside TLS

One straw man approach is to tunnel one TLS session inside another. In this arrangement, the outer TLS session provides both confidentiality and integrity, while the inner session uses an integrity-only cipher suite (i.e., anyone can read the plaintext, but it cannot be modified without a key). Then, an auditor can act as a man in the middle for the outer TLS session, but not the inner one. Since the auditor is acting as a man in the middle for the outer session, it can read the plaintext of the inner session, but because only the client and server have the keys to the inner session, the auditor cannot forge or modify data.

This approach has three major flaws. First, it imposes a significant overhead in message size and connection setup, which is a major concern for some IoT devices. More significantly, integrity-only cipher suites are being removed from TLS 1.3, and they have limited support even today. Supporting this approach in the future would require modifications to the TLS specification. Finally, TLS-inside-TLS may break compatibility with some servers and middleboxes, e.g., TLS terminators.

3.2 Straw Man: Application Layer MAC

A second possibility is simply to add application-layer authentication in the form of a MAC. An auditor acting as a man-in-the-middle at the TLS layer can decrypt and inspect traffic, but because it does does not have the application-layer MAC key, it cannot modify this traffic.

This approach has the major issue that safe auditing requires modifications to *all* application-layer protocols in use—each one essentially reimplementing a subset of TLS (key exchange, etc.), in the process inheriting the attendant difficulty of correctly designing and implementing such protocols.

3.3 Straw Man: Separate TLS Channel

A third straw man approach is for clients to log everything they send and receive to each auditor over a separate TLS channel. This approach also has two major problems. First, it requires each device to send copies of all traffic to every auditor; for low-power or embedded IoT devices, this increase in traffic might present an unacceptable energy cost. Second, in this scheme, a device can equivocate by exfiltrating private data to the manufacturer while sending a benign-looking stream of traffic to the auditors. Barring careless mistakes (e.g., data size differences) on the part of the device, auditors would have no way of detecting this misbehavior.

3.4 TLS-RaR

Our approach, which we call TLS-RaR, is illustrated in Figure 1. A person owns one or more networked *devices* that use a local-area network under that person's control, where they can install auditors that passively observe all traffic. An owner cannot modify the software or security credentials of devices. Each device communicates with its manufacturer or other *servers* outside the LAN. These communication channels, which we call *main channels*, are secured using TLS with standard cipher suites.

In addition to communicating with servers, a device allows *auditors* to establish *key release channels*. Key release channels can either be directly with the device, or with an agreed-upon third party, such as the application's cloud service. Key release channels require that an auditor authenticate that it is allowed to decrypt traffic. The choice of

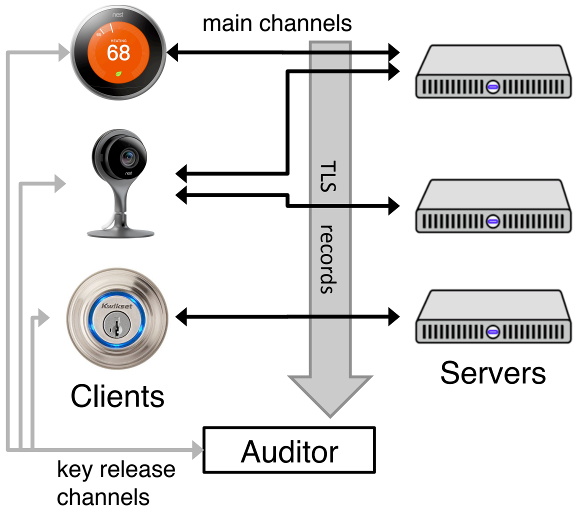

Figure 1: TLS-RaR architecture. Both the main client-server channels and the auxiliary key-release channels use TLS. An auditor passively collects TLS records from the client-server channel. After a key rotation has been fully consummated, clients share the old session key with the auditors. Several independent auditors may operate in tandem.

authentication mechanism is outside the scope of this paper; possibilities include passwords, public-key infrastructure, or whatever the user already uses to authenticate to the cloud service's website.

Auditors buffer encrypted TLS records to and from devices. Initially, the auditors do not have the keys used to encrypt these records and so cannot view the underlying cleartext. Upon a request by the user or auditor, the device *rotates* the TLS connections of their main channels to use new keys. This request may come as an impromptu "surprise inspection"—the cloud service and device may not know of the presence of the auditor until receiving this request.

Once a device is certain that both it and the server have retired the old keys, it uses its key release channel to *release* the old keys to auditors. The auditors use these keys to authenticate and decrypt the saved ciphertext.

Withholding keys from auditors while they passively monitor and buffer encrypted TLS records to and from devices meets the present-moment integrity requirement, while releasing only expired keys meets the past auditability requirement. Meeting the audit robustness and requiring no TLS changes depends on exactly how the key rotation and release protocols are implemented, which the next two sections describe.

4. KEY ROTATION AND RELEASE

TLS-RaR has two protocols, rotation and release. The *rotation* protocol advances a TLS session to a new epoch by using new symmetric keys and retiring the old ones. The *release* protocol gives auditors the keys from a prior epoch which it is sure have been safely retired.

4.1 Key Rotation

To rotate keys, TLS-RaR generates new keying material for the session and *retires* the old keys, meaning that the device and server will discard any further TLS records using

TLS mechanism	*auth-ack?*	*RTT*	*version*
KeyUpdate	✗	1	1.3
KeyUpdate + Heartbeat	✓	1	1.3
KeyUpdate + `OPTIONS`	✓	1	1.3
Resume	✗	1	≤ 1.2
Resume + Heartbeat	✓	1	≤ 1.2
Resume + `OPTIONS`	✓	1	≤ 1.2
Renegotiate	✓	2	≤ 1.2
Reconnect	*?*	4	any

Table 1: Mechanisms for rotating keys (§4.1).

the old keys. If the old keys are not retired, an attacker with the key can violate present-moment integrity. For example, suppose a device releases the TLS session keys immediately after closing the TCP stream and without verifiably shutting down the TLS session. An attacker who prevents the device's `FIN` from reaching the server and spoofs the server's `FIN/ACK` response could use the released keys to continue the TLS session and masquerade as the device. This broken straw man protocol illustrates the following necessary condition for secure key rotation:

Authenticated Key-Retirement ACK (auth-ack): For a device to securely rotate keys, it must first receive an authenticated acknowledgment that the server has retired the keys.

The TLS standards define several mechanisms for a TLS session to choose new symmetric keys, but most require an extra step to have an authenticated key-retirement ACK. Table 1 lists these mechanisms; below, we briefly describe each.

KeyUpdate. The TLS 1.3 draft standard [53] introduces a mechanism called KeyUpdate that causes both TLS endpoints to generate new session keys. KeyUpdate is asynchronous, may be initiated by either side of the connection, and imposes no blocking and almost no overhead.

Unfortunately, KeyUpdate is not sufficient for an auth-ack because it is possible for either endpoint to send KeyUpdate on its own initiative. As a result, a device that sends and then receives a KeyUpdate message does not know if the server was responding to the device's KeyUpdate or if the server initiated a KeyUpdate on its own. To distinguish these two cases, a device must send a message after initiating KeyUpdate and then wait for the server to respond to that message.

The simplest mechanism for the required request-response exchange is a TLS heartbeat [52]. In this approach, the device sends a heartbeat request after initiating KeyUpdate; the corresponding heartbeat response serves as authenticated acknowledgment that the server has received and acted upon the request. This mechanism is efficient: the device can send this heartbeat message immediately after sending a KeyUpdate. The entire rotation takes one round trip and doesn't block traffic at any point.

Because of security concerns in the wake of the Heartbleed bug [19], TLS heartbeat is an optional feature that is often disabled. When heartbeat is not available, a device can instead send an application-layer request. For example, devices communicating with an HTTP server can send an `OPTIONS` request [28], which elicits a short response and has no effect on the HTTP session.

Resume. TLS standards prior to 1.3 [22,23,24] allow clients to resume a prior session with a server using a session ID or session ticket [57]. While this causes a key rotation, the device does not immediately learn that the server has changed its keys: because the device speaks last in a session resumption, it does not know that its final message arrived until it receives more traffic from the server. Like KeyUpdate, pairing Resume with a heartbeat or an application-layer request-response gives an auth-ack.

Renegotiate. TLS standards prior to 1.3 allow endpoints to renegotiate an existing TLS session, meaning that they repeat the TLS handshake over the existing connection and then continue communicating using new keys. The server sends the final message in a renegotiation exchange, and this message serves as an auth-ack. The major drawback of renegotiation is its cost: it requires computationally expensive public key cryptographic operations.

Reconnect. The final rotation approach is for a device to disconnect a TLS session by sending a TLS `close_notify` message and waiting for a corresponding TLS `close_notify` from the server. After receiving the `close_notify`, the device knows the server has retired the session's keys. The device starts the next epoch by initiating a new connection. Like KeyUpdate, disconnection can be initiated asynchronously by either side, but TLS implementations can nevertheless give an auth-ack by discarding all incoming traffic after sending or receiving a `close_notify` message. Implementations that close a session's underlying TCP connection immediately upon receiving a `close_notify` without sending a response do not provide an auth-ack.

TLS Session Termination. Except for reconnection, the above protocols obtain an auth-ack by assuming that a device will continue communicating after rotation: this post-rotation traffic provides authenticated acknowledgment of an epoch's end because it uses new keys. When a TLS session terminates, however, there are no new keys or messages that serve as an auth-ack, meaning that a device cannot safely release the session's final keys and auditors cannot examine the session's final plaintext.

The reconnection protocol above can provide an auth-ack at session termination when paired with a TLS implementation that discards all incoming traffic after sending or receiving a `close_notify`. Alternatively, the application can use a session tracking mechanism at the application layer and invalidate that session before terminating the TLS connection. If an attacker attempts to append data, the server will ignore it because the application session has terminated.

4.2 Key Release

Once a TLS-RaR device has safely rotated keys, it releases the old keys to auditors. The release protocol must maintain the auditability and robustness properties of Section 2. In particular, it must prevent falsifying the plaintext of an epoch, and it must prevent devices from equivocating about that content. For example, a device that broadcasts keys to several auditors does not give audit robustness: if one auditor fails to receive the keys, an attacker with the keys can falsify additional TLS records. When that auditor finally receives the keys, it will be convinced that the forged traffic is real.

To safely release keys, TLS-RaR uses a *Sealed-History Release* protocol. Devices compute cryptographic hashes of all TLS records sent and received. After a device rotates keys and is ready to release the keys k_t for epoch t, it signs and discloses to the auditors a message:

$$\Big(\texttt{release}, t, k_t, \mathrm{Hash}(r_1^d, \ldots, r_n^d), \mathrm{Hash}(r_1^s, \ldots, r_n^s)\Big)$$

where $(r_1^d, \ldots, r_n^d)$ and $(r_1^s, \ldots, r_n^s)$ are the TLS records sent by device and server, respectively, in epoch t.

Providing a sealed history prevents falsified content. While an attacker can delay a key release message's arrival at another auditor, it cannot forge a new one (because devices and auditors communicate over an authenticated key release channel). Meanwhile, for a given key-release message, an attacker cannot forge new records because doing so requires finding a hash collision. To participate, the device must compute a cryptographic hash of the messages sent and received during each key epoch. This may be computed incrementally.

A remaining concern is that a malicious device might attempt to evade audit using *deniable encryption* [12], allowing the device to equivocate between an inappropriate plaintext (sent to the server) and an innocuous one (decrypted by the auditor). Specifically, the device could release a key $\hat{k}_t$ that seems to decrypt and authenticate data correctly, but the device and server know of another key k_t that decrypts and authenticates a different, valid plaintext TLS transcript. Sealed-History Release prevents deniable encryption from producing valid plaintext by requiring auditors to check that the structure of every decrypted TLS record is valid. Under reasonable assumptions about the security of modern AEAD ciphers (such as `AES_GCM` [43]), finding two different keys that correctly authenticate this header content but yield different records is infeasible. With overwhelming probability, no such pair of keys exists.

5. EVALUATION

To evaluate how the key rotation protocols affect performance, we implemented TLS-RaR on our own IoT device and web service, along with a prototype auditor.

Our IoT device is a temperature sensor built on the Raspberry Pi 3 platform with 4 × 1.2 GHz ARM Cortex-A53 cores. The web service consists of web servers built on the Python Twisted Web library. The device periodically uploads JSON objects encoding timestamped temperature readings. The device owner can view the readings from a web interface.

We implemented a "surprise" auditing interface on the website, where a user can click a button to request keys for auditing. Upon clicking the button, the server requests a key rotation and release from the device. The device then initiates a key rotation by TLS Renegotiate or Resume on its TLS connections. After a key rotation is complete, the device signs the superseded keys and traffic hashes and submits them to the web server, where the user can view and download them.

Because TLS 1.3 is not fully specified or implemented in current software and libraries our implementation communicates using TLS 1.2 with help from the OpenSSL 1.0.2 library. Implementing the TLS-RaR client required making only one modification to OpenSSL: we added a callback that releases the TLS session keys to the application after a rotation.

We also implemented a prototype auditor, built on `libpcap` and `tshark`, that captures TLS streams. After a rotation-and-release, our auditor reads the signed key-release message, verifies the signature and hashes, and decrypts the captured ciphertext.

	(CPU seconds / GiB)
Baseline TLS (encrypt/send)	9.8
TLS-RaR (encrypt/hash/send)	11.9
Mean CPU overhead per byte:	22%

Table 2: Ongoing CPU overhead. The extra SHA-256 hashing required by TLS-RaR imposes about 22% CPU overhead on an ARM Cortex-A53 processor, versus baseline TLS using AES-128.

	(CPU milliseconds)
Rotate by Renegotiation	46
Rotate by Resume + heartbeat	1.2
Release	0.7

Table 3: Rotation/release overhead. It takes less than two milliseconds of total CPU time on the ARM Cortex-A53 for TLS-RaR to rotate keys (using the Resume + heartbeat method) and prepare a signed release.

	(transport bytes)
Rotate by Renegotiation	3004
Rotate by Resume + heartbeat	573
Release	900

Table 4: Network overhead. Each time the user requests a key rotation-and-release, TLS-RaR will need to send roughly 1.5 kilobytes over the wire (using the Resume + heartbeat method).

5.1 Performance

We used our IoT temperature sensor as a platform for analyzing the costs of TLS-RaR. Tables 2, 3, and 4 show the ongoing CPU overhead, the CPU cost of each rotate/release event, and the IO cost of each rotate/release event, as measured on the Cortex-A53 processor using standard TLS 1.2 with the AES-128 cipher.

In summary, TLS-RaR imposes an ongoing CPU overhead of roughly 22%, compared with baseline TLS. This is attributable to the ongoing hashing required to be able to prepare a Sealed-History Release if the device is eventually audited. This ongoing overhead would decrease on IoT devices with hardware support for SHA-256 or another secure hash algorithm. By contrast, the CPU and network costs associated with rotating and releasing keys are minimal.

The values from the tables can be used to estimate the cost of rotation in relation to TLS traffic encryption. Rotation by Renegotiation, the most computationally expensive method, reaches parity with the traffic encryption and hashing cost at approximately 4 MiB of plaintext per epoch, and it reduces to approximately 10% of the overall CPU cost of TLS at approximately 36 MiB of plaintext per epoch. At 36 MiB per epoch, Rotation by Renegotiation increases the amount of bytes transferred by less than 0.01%, but data transfer will be interrupted by 2 round trips.

Rotation by Resume and Heartbeat is approximately 1/20 of the CPU cost of Rotation by Renegotiation. At approximately 1.5 MiB per epoch, rotations consume 10% of overall TLS CPU time while placing less than 0.04% extra bytes on the wire, and interrupting transfer for 1 round trip.

Rotation by a TLS 1.3 KeyUpdate will consume an amount of CPU similar to a Resume, and will not interrupt data transfer at all.

5.2 IoT Communications Survey

To put TLS-RaR in the context of real-world IoT systems, we recorded and inspected the traffic from commercially available devices: a Nest Protect smoke detector, a Nest Cam, a Samsung SmartThings hub with a SmarthThings Outlet, and a Vizio TV.

We connected the devices to the Internet through a Wi-Fi base station that itself was connected through a desktop PC with two Ethernet interfaces. The PC recorded the IoT devices' traffic using tcpdump for several weeks. We observed the following communication characteristics:

1. The Nest Cam makes a single long-lived HTTPS connection to subdomains of dropcam.com that resolve to Amazon EC2 hosts, and uploads video continuously at a rate of approximately 200 kilobits/s.

2. The Nest Protect makes a TCP connection to an Amazon EC2 server about every six hours. The traffic doesn't appear to be TLS; we believe it is using Nest's Weave protocol. The TCP connections last between 3 and 12 seconds, and the device sends between 1 kilobyte and 45 kilobytes, and receives between 1 kilobyte and 6 kilobytes, each time. (This is with the device idle and no fires happening.)

3. The Samsung SmartThings Hub, with a SmartThings Outlet connected to it, maintains one long-lived HTTPS connection to an Amazon EC2 host. It exchanges about 50 bytes every 30 seconds. Every five minutes, the device makes an HTTPS connection to another Amazon EC2 host, sending approximately 550 bytes and receiving about 2000 bytes each time.

4. Every 24 hours, the Vizio TV made a series of brief HTTPS and HTTP connections to hosts on Amazon EC2, uploading tens of kilobytes and receiving up to a megabyte each time.

These flow lengths and durations, combined with the performance measurements from the previous section, suggest that the overheads of deploying TLS-RaR in the context of these IoT devices would be reasonable—even if the user wished to audit their devices' traffic every minute. Even at that frequency, the overall cost would be dominated by the ongoing cost of computing the hashed history, not of rotating and releasing keys.

5.3 Survey: Server Compatibility

TLS-RaR requires no changes to the TLS protocol, but it does require a server that supports certain TLS features. To test the compatibility of TLS-RaR with real-world servers, we surveyed the subset of the Alexa Top 1,000,000 sites [2] that support long-lived HTTPS connections [29]. Table 5 gives an overview of the results.

	Fraction of Servers
Rotation by Reconnect	54.2%
Rotation by Renegotiate	12.2%
Rotation by Resume	0.5%
TLS 1.3 KeyUpdate	*0% at time of survey*

Table 5: Survey of the top 1,000,000 websites, conducted in early 2016. Most existing server-side TLS implementations support at least one method of key rotation and could be used with TLS-RaR today. As TLS 1.3 is deployed, we expect close to 100% of updated servers to support rotation by KeyUpdate.

Most of the servers in our sample are compatible with Rotate by Reconnection (assuming they stop receiving data before sending `close_notify`; §4.1). The remaining servers are incompatible because they do not reply to `close_notify` correctly; incompatible servers simply close the underlying TCP connection without sending a `close_notify`.

Full reconnection of a TLS session, however, is an expensive way to rotate a key. Of the lighter-weight methods, about 12% of servers in our sample support Rotate by Renegotiation, while only a small fraction (0.5%) support Rotate by Resume.

Because the TLS 1.3 specification is still in draft form, we were unable to test Rotation by KeyUpdate, but expect support to become ubiquitous when TLS 1.3 is deployed because KeyUpdate is a mandatory part of the draft specification.

6. ETHICAL CONSIDERATIONS

This paper arose out of our personal concerns about the privacy implications of IoT devices. As everyday users, we would like to know exactly what our devices are saying about us. Though the motivation for this work is to empower the consumer, there is a risk that businesses or governments could deploy TLS-RaR in a way that would harm the privacy or agency of end users. Recent disclosures of vast and sophisticated surveillance by state-sponsored actors [13,26,30,33,44,49,54,58,62,63] have brought new focus to the ethical implications of computer-security systems [55]. Understanding the ethical status of TLS-RaR requires a weighing of the potential social risks and rewards of the scheme.

In terms of risks, it is possible that the comparably benign nature of TLS-RaR—in that it allows only read-only auditing—might permit an employer or government to more easily compel its use. Even so, we expect that corporations or governments with the power to force their employees or citizens to use TLS-RaR would almost certainly prefer to deploy a full man-in-the-middle proxy to skirt the limitations on the auditor that TLS-RaR imposes. Indeed, employers already exert considerable control over their employees' devices, and workplace policies requiring man-in-the-middle proxies are already widespread [37]. We find it unlikely that the existence of TLS-RaR would change the calculus of those engaging in network surveillance in a meaningful way.

One of the design goals of TLS-RaR was to enable auditing of TLS traffic even for devices that are cryptographically bound to communicate only with their manufacturers' servers (e.g., via certificate pinning [27]). This raises another potential ethical qualm: by providing a means to audit IoT device traffic, TLS-RaR could make it more acceptable for manufacturers to sell “locked down” devices. We appreciate that some users value the ability to redirect their devices’ traffic to a user-controlled server, but business realities mean that the sale of locked devices will likely continue. With TLS-RaR, manufacturers will be able to produce locked devices that still allow their owners to audit the traffic the devices send.

With these potential risks in mind, we now recall the social benefits of TLS-RaR. We find the fact that today’s IoT devices are unaccountable to their owners troubling, and we view it as socially unwise to grant manufacturers unbridled control over devices placed inside private homes—a place “strongly guarded by every holy feeling” [16]. From the perspective of those who wish to combat inappropriate surveillance, the fact that TLS-RaR enables device owners and security researchers to learn what information their devices are sending back to the manufacturer has considerable social value.

While TLS-RaR allows anyone to audit their IoT devices, its benefits come even if only a handful of people do. Watchdog groups will be able to verify manufacturers’ claims about their devices’ behaviors (as they can for Web browser [32] and ISP behaviors [35]) and consumers will gain confidence in manufacturers’ privacy polices, or avoid products that behave unfavorably.

7. LIMITATIONS AND DISCUSSION

TLS-RaR allows device owners (including consumer watchdogs and security researchers) to audit the traffic of their IoT devices. It achieves this without modifying the TLS protocol or compromising the integrity of the device-cloud connection. This model, while valuable, has some limitations, which we discuss here.

First, TLS-RaR assumes that IoT application vendors are willing to share their traffic with device owners. Our initial responses from numerous vendors across many application domains (home automation, insurance, automotive, industrial automation, smart appliances) has been unanimously positive. The most common concern we have heard is that owners may react irrationally to raw data. For example, the owner of a carbon monoxide sensor reporting a concentration of 2 parts per million might be alarmed even though this is well below the recommended safe limit, 9 ppm. On the whole, the vendors we have spoken with said that the benefits of transparency would likely outweigh its costs.

Second, TLS-RaR provides auditors with delayed access to an encrypted stream and so does not support systems that require real-time access to encrypted traffic, such as intrusion-prevention systems that seek to interdict harmful traffic before it reaches an endpoint. To our knowledge this is a necessary tradeoff to simultaneously provide present-moment integrity and past auditability while maintaining backwards compatibility with existing TLS implementations and infrastructure.

Third, as we discuss in Section 2, TLS-RaR does not prevent a device from intentionally hiding communication from an auditor. A deceptive manufacturer—or a compromised device—can leak private data by encrypting it again (within the TLS stream) or by using steganography, a hidden wireless modem, or another covert channel. A complementary system might attempt to frustrate, or at least detect, likely covert channels.

Fourth, the primary benefits of auditing can be achieved even if only a small fraction of devices are ever audited. A device does not know it is being audited until *after* it has sent the traffic. Only a few audits by consumer watchdogs or security researchers will benefit everyone, encouraging device engineers to respect owner privacy and influencing them to fix flaws when discovered.

Finally, there are several alternative or complementary approaches to TLS-RaR; we discuss related work more fully in the next section.

8. RELATED WORK

The idea of allowing trusted third parties to inspect the contents of encrypted communications has a considerable literature. In this section, we compare TLS-RaR with related work and systems.

Man-in-the-Middle TLS Proxies. The standard way to give middleboxes access to the plaintext of a TLS stream is to install the middlebox’s public key as a root certificate on end-users’ devices [37, 59], allowing the middlebox to act as a man-in-the-middle. This technique carries risk. A recent survey [21] found that 75% of the man-in-the-middle TLS proxies tested exposed the client behind the proxy to MITM attacks by adversaries on the Internet (i.e., not the proxy). These middleboxes become another point of failure for the client’s security, and any vulnerability in the middlebox can become a vulnerability for the client (e.g. [18, 20]).

Multi-Context TLS [45] is a recent proposal that allows endpoints to grant middleboxes read-only or read-write access to parts of a TLS stream. In brief, mcTLS augments the TLS protocol with three keyed message-authentication codes that control access to encrypted data. Endpoints hold one key, the second is shared with middleboxes with read/write access, and the third is shared with middleboxes with read-only access.

Unlike mcTLS, TLS-RaR maintains compatibility with the TLS protocol (and with some existing endpoint implementations, §5.3). Avoiding changes and extensions to the TLS protocol is desirable both for ease of deployment and for security: TLS is a subtle protocol [7,8,9], and making changes risks introducing new vulnerabilities. On the server side, TLS is commonly implemented by hardware TLS terminators and load balancers, which the vendor may not be able to modify.

Further, TLS-RaR ensures that auditors see the same byte stream that the TLS recipient does (§2). In contrast, mcTLS’s use of separate keying material held by endpoints and by different kinds of middleboxes opens the door to mischief. Finally, TLS-RaR allows auditors to begin auditing “impromptu” in the middle of a long-running TLS session. In contrast, mcTLS requires participation by all middleboxes when the TLS session is being negotiated, meaning that existing connections cannot be audited, and devices might choose to behave honestly only when under an auditor’s gaze.

mcTLS has several capabilities that TLS-RaR does not. mcTLS allows for read-write middleboxes (e.g., a transparent Web cache), and for real-time read-only middleboxes (e.g., an intrusion-prevention system that immediately interdicts harmful traffic). TLS-RaR only supports applications that audit traffic after the fact.

BlindBox [61] is another proposed system for granting middleboxes restricted access to TLS traffic. BlindBox allows middleboxes to search for substrings in TLS-encrypted traffic. To do so, the BlindBox endpoints use a secondary connection to the middlebox, and on that connection, send the same traffic with a *searchable encryption* scheme [4, 6, 10, 17, 64] that allows the middlebox to determine if a particular substring appears in the TLS traffic without learning the full contents. BlindBox requires the endpoints to faithfully execute the protocol by providing an accurate version of their communications on the secondary stream, so that the middlebox can see if an intrusion-signature substring was sent. This differs from the goals of TLS-RaR: we wish to allow the auditor to see everything that the device transmits, and do not trust the vendor that controls both endpoints.

Proposed TLS Proxy Support. In parallel to the academic work on inspecting TLS traffic, a number of protocol designers have proposed extensions or modifications to TLS. One theme in this line of work is to enable a way for a TLS client to specify an explicit proxy with the privileges to decrypt and modify the TLS stream [41, 42, 46, 48]. TLS-RaR differs from these proposals in that it never allows the auditor to modify the TLS stream and does not require changes to TLS.

Another approach is to make changes at the application layer rather than at the transport layer. QoS2 enables HTTP caching middleboxes while providing TLS-strength integrity and authenticity guarantees [68]. When deployed in tandem with QoS2, TLS-RaR could allow a device owner to audit their device's QoS2 sessions.

Computing on Encrypted Data. A substantial literature has developed about systems that compute on encrypted data [50, 51, 61, 66]. In these systems, the client is trusted but the server isn't. From a privacy perspective, such protocols are preferable to ones that reveal sensitive information to the server, but the trust model differs from that of the IoT, where both the device and server are under the control of the same vendor and may not faithfully execute the protocol. We regard this work as complementary. A privacy-preserving IoT system might use both mechanisms in concert: the protocol would prevent sensitive information from being shared with the cloud, *and* auditors would verify that the device faithfully executed that protocol.

9. CONCLUSION

Owners of IoT devices should have the right to listen in on what their own devices are saying about them. Towards this goal, we have proposed TLS-RaR, a system that allows device owners, consumer watchdogs, and security researchers to inspect TLS streams with the cooperation of the device. TLS-RaR is compatible with some existing TLS servers, makes no changes to the TLS protocol, and does not allow the auditor to man-in-the-middle TLS connections. In this way, TLS-RaR reconciles the desire for end-to-end encryption with the ability to see what our devices are saying about us.

10. ACKNOWLEDGMENTS

We would like to thank our shepherd, Lin Zhong, and the anonymous reviewers for helpful feedback. We are grateful to Eric Rescorla, Adam Langley, and David Benjamin for discussions about this proposal. This work was supported in part by Intel/NSF CPS Security grant #1505728, with additional support from an NDSEG Fellowship, the Simons Foundation, VMware, Qualcomm, Analog Devices, Google, Amazon, and the Secure Internet of Things Project.

11. REFERENCES

[1] Alexa and Alexa device FAQs. Amazon.com. https://www.amazon.com/gp/help/customer/display.html?nodeId=201602230. Accessed: 2017-04-25.

[2] Top 1,000,000 sites (updated daily). Alexa Internet Inc., 2009-2016. http://s3.amazonaws.com/alexa-static/top-1m.csv.zip. Accessed: 2016-01-18.

[3] Keeping data safe at Nest. Nest Labs, Dec. 2016. https://nest.com/security/.

[4] M. Abdalla, M. Bellare, D. Catalano, E. Kiltz, T. Kohno, T. Lange, J. Malone-Lee, G. Neven, P. Paillier, and H. Shi. Searchable encryption revisited: Consistency properties, relation to anonymous IBE, and extensions. In *Advances in Cryptology – CRYPTO 2005: 25th Annual International Cryptology Conference, Santa Barbara, California, USA, August 14-18, 2005. Proceedings*, pages 205–222, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[5] J. Angwin and J. Valentino-Devries. Google's iPhone tracking: Web giant, others bypassed Apple browser settings for guarding privacy. The Wall Street Journal, Feb. 17, 2012. http://www.wsj.com/articles/SB10001424052970204880404577225380456599176.

[6] M. Bellare, A. Boldyreva, and A. O'Neill. Deterministic and efficiently searchable encryption. In *Proceedings of the 27th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO'07, pages 535–552. Springer-Verlag, Berlin, Heidelberg, 2007.

[7] B. Beurdouche, K. Bhargavan, A. Delignat-Lavaud, C. Fournet, M. Kohlweiss, A. Pironti, P.-Y. Strub, and J. K. Zinzindohoue. A messy state of the union: Taming the composite state machines of TLS. In *IEEE Symposium on Security & Privacy 2015*, San Jose, United States, May 2015. IEEE.

[8] K. Bhargavan, A. D. Lavaud, C. Fournet, A. Pironti, and P. Y. Strub. Triple handshakes and cookie cutters: Breaking and fixing authentication over TLS. In *Proceedings of the 2014 IEEE Symposium on Security and Privacy*, SP '14, pages 98–113, Washington, DC, USA, 2014. IEEE Computer Society.

[9] K. Bhargavan and G. Leurent. Transcript collision attacks: Breaking authentication in TLS, IKE, and SSH. In *23nd Annual Network and Distributed System Security Symposium 2016, NDSS 2016*, Feb. 2016.

[10] D. Boneh, G. Di Crescenzo, R. Ostrovsky, and G. Persiano. Public key encryption with keyword search. In *Advances in Cryptology - EUROCRYPT 2004: International Conference on the Theory and Applications of Cryptographic Techniques, Interlaken, Switzerland, May 2-6, 2004. Proceedings*, pages 506–522, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[11] S. Cabuk, C. E. Brodley, and C. Shields. IP covert timing channels: design and detection. In *Proceedings of the 11th ACM Conference on Computer and Communications Security*, CCS '04, pages 178–187, New York, NY, USA, 2004. ACM.

[12] R. Canetti, C. Dwork, M. Naor, and R. Ostrovsky. Deniable encryption. In *Annual International Cryptology Conference*, CRYPTO '97, Aug. 1997.

[13] L. Cauley. NSA has massive database of Americans' phone calls; 3 telecoms help government collect billions

of domestic records. USA Today, May 11, 2006. http://usatoday30.usatoday.com/news/washington/2006-05-10-nsa_x.htm.

[14] E. Chiel. Here are the sites you can't access because someone took the internet down. Fusion, October 20 2016. http://fusion.net/story/360952/which-sites-affected-ddos-attack/.

[15] R. Chirgwin. Internet of Things 'smart' devices are dumb by design. The Register, Jan. 19, 2016. https://www.theregister.co.uk/2016/01/19/iot_smart_devices_are_dumb/.

[16] Cicero. *Ad Pontifices*, XLI., 109, translated by Harbottle, Dictionary of Quotations (Classical) (Sonnenschein 1906).

[17] R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky. Searchable symmetric encryption: Improved definitions and efficient constructions. In *Proceedings of the 13th ACM Conference on Computer and Communications Security*, CCS '06, pages 79–88, New York, NY, USA, 2006. ACM.

[18] CVE-2012-3372: Vulnerability in Cyberoam DPI devices. Common Vulnerabilities and Exposures List, June 30, 2012. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2012-3372.

[19] CVE-2014-0160: OpenSSL 'Heartbleed' Vulnerability. Common Vulnerabilites and Exposures List, Apr. 2014. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-0160.

[20] CVE-2016-1280: Self-signed certificate with spoofed trusted issuer cn accepted as valid. Common Vulnerabilites and Exposures List, July 2016. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2016-1280.

[21] X. d. C. de Carnavalet and M. Mannan. Killed by proxy: Analyzing client-end tls interception software. In *23nd Annual Network and Distributed System Security Symposium 2016, NDSS 2016*, Feb. 2016.

[22] T. Dierks and C. Allen. The TLS protocol version 1.0. IETF, 1999. RFC 2246.

[23] T. Dierks and E. Rescorla. The transport-layer security (TLS) protocol, version 1.1. IETF, 2006. RFC 4346.

[24] T. Dierks and E. Rescorla. The transport layer security (TLS) protocol version 1.2. IETF, 2008. RFC 5246.

[25] B. Donohue. Dozens of popular Android apps leak sensitive user data. Kaspersky Lab official blog, Sept. 18, 2016. https://blog.kaspersky.com/privacy_holes_in_popular_android_apps/6047/.

[26] R. Ensafi, D. Fifield, P. Winter, N. Feamster, N. Weaver, and V. Paxson. Examining how the Great Firewall discovers hidden circumvention servers. In *Proceedings of the 2015 Internet Measurement Conference*, IMC '15, pages 445–458, New York, NY, USA, 2015. ACM.

[27] S. Fahl, M. Harbach, H. Perl, M. Koetter, and M. Smith. Rethinking SSL development in an appified world. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer and Communications Security*, CCS '13, pages 49–60, New York, NY, USA, 2013. ACM.

[28] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext transfer protocol—HTTP/1.1. IETF, 1999. RFC 2616.

[29] R. Fielding and J. Reschke. Hypertext transfer protocol (HTTP/1.1): Message syntax and routing. IETF, 2014. RFC 7230.

[30] J. Follorou and F. Johannès. Révélations sur le Big Brother français [revelations about the French Big Brother]. Le Monde, July 4, 2013. http://www.lemonde.fr/societe/article/2013/07/04/revelations-sur-le-big-brother-francais_3441973_3224.html.

[31] S. Gianvecchio and H. Wang. Detecting covert timing channels: An entropy-based approach. In *Proceedings of the 14th ACM Conference on Computer and Communications Security*, CCS '07, pages 307–316, New York, NY, USA, 2007. ACM.

[32] M. Godbe. Google deceptively tracks students' internet browsing, EFF says in FTC complaint. Electronic Frontier Foundation, December 1, 2015. https://www.eff.org/press/releases/google-deceptively-tracks-students-internet-browsing-eff-says-complaint-federal-trade.

[33] G. Greenwald. *No Place to Hide: Edward Snowden, the NSA, and the U.S. Surveillance State*. Metropolitan Books, 2014.

[34] S. Grover and N. Feamster. The internet of unpatched things. In *PrivacyCon*, Jan. 2016.

[35] J. Hoffman-Andrews. Ad network turn will suspend zombie cookie program. when will verizon? Electronic Frontier Foundation, January 16, 2015. https://www.eff.org/deeplinks/2015/01/ad-network-turn-will-suspend-zombie-cookie-program-when-will-verizon.

[36] A. Houmansadr and N. Borisov. CoCo: Coding-based covert timing channels for network flows. In *Proceedings of the 13th International Conference on Information Hiding*, IH'11, pages 314–328, Berlin, Heidelberg, 2011. Springer-Verlag.

[37] L. S. Huang, A. Rice, E. Ellingsen, and C. Jackson. Analyzing forged SSL certificates in the wild. In *Proceedings of the 2014 IEEE Symposium on Security and Privacy*, SP '14, pages 83–97, Washington, DC, USA, 2014. IEEE Computer Society.

[38] Internet of things research study, 2015 report. Hewlett Packard. http://www8.hp.com/h20195/V2/GetPDF.aspx/4AA5-4759ENW.pdf.

[39] H. Krawczyk. Perfect forward secrecy. In *Encyclopedia of Cryptography and Security*, pages 457–458. Springer, 2005.

[40] Y. Liu, D. Ghosal, F. Armknecht, A.-R. Sadeghi, S. Schulz, and S. Katzenbeisser. Hide and seek in time: Robust covert timing channels. In *Proceedings of the 14th European Conference on Research in Computer Security*, ESORICS'09, pages 120–135, Berlin, Heidelberg, 2009. Springer-Verlag.

[41] S. Loreto, J. Mattsson, R. Skog, H. Spaak, G. Gus, D. Druta, and M. Hafeez. Explicit trusted proxy in HTTP/2.0. IETF HTTPBis Working Group Internet-Draft `draft-loreto-httpbis-trusted-proxy20-01`, February 14, 2014.

[42] D. McGrew, D. Wing, Y. Nir, and P. Gladstone. TLS

proxy server extension. IETF TLS Internet-Draft `draft-mcgrew-tls-proxy-server-01`, July 16, 2012.

[43] D. A. McGrew and J. Viega. The Galois/counter mode of operation (GCM), May 31, 2005. http://csrc.nist.gov/groups/ST/toolkit/BCM/documents/proposedmodes/gcm/gcm-spec.pdf.

[44] E. Nakashima. Chinese hackers who breached Google gained access to sensitive data, U.S. officials say. The Washington Post, May 20, 2013. https://www.washingtonpost.com/51330428-be34-11e2-89c9-3be8095fe767_story.html.

[45] 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. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, SIGCOMM '15, pages 199–212, New York, NY, USA, 2015. ACM.

[46] Y. Nir. A method for sharing record protocol keys with a middlebox in TLS. IETF TLS Working Group Internet-Draft `draft-nir-tls-keyshare-02`, March 26, 2012.

[47] V. Paxson, M. Christodorescu, M. Javed, J. R. Rao, R. Sailer, D. L. Schales, M. P. Stoecklin, K. Thomas, W. Venema, and N. Weaver. Practical comprehensive bounds on surreptitious communication over DNS. In *Proceedings of the 22nd USENIX Security Symposium*, USENIX-SS'17, pages 17–32. USENIX Association, Aug. 2013.

[48] R. Peon. Explicit proxies for HTTP/2.0. IETF Network Working Group Internet-Draft `draft-rpeon-httpbis-exproxy-00`, June 8, 2012.

[49] A. Peterson. How the NSA may be using games to encourage digital snooping. The Washington Post, June 18, 2014. https://www.washingtonpost.com/news/the-switch/wp/2014/06/18/how-the-nsa-may-have-used-games-to-encourage-digital-snooping/.

[50] R. A. Popa, C. M. S. Redfield, N. Zeldovich, and H. Balakrishnan. CryptDB: Protecting confidentiality with encrypted query processing. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, SOSP '11, pages 85–100, New York, NY, USA, 2011. ACM.

[51] R. A. Popa, E. Stark, J. Helfer, S. Valdez, N. Zeldovich, M. F. Kaashoek, and H. Balakrishnan. Building web applications on top of encrypted data using Mylar. In *Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation*, NSDI'14, pages 157–172, Berkeley, CA, USA, 2014. USENIX Association.

[52] M. W. R. Seggelmann, M. Tuexen. Transport layer security (TLS) and datagram transport layer security (DTLS) heartbeat extension. IETF, 2012. RFC 6520.

[53] E. Rescorla. The transport layer security (TLS) protocol version 1.3. IETF, 2017. draft-ietf-tls-tls13-19.

[54] J. Risen and E. Lichtblau. Bush lets U.S. spy on callers without courts. The New York Times, December 16, 2005. https://www.nytimes.com/2005/12/16/politics/bush-lets-us-spy-on-callers-without-courts.html.

[55] P. Rogaway. The moral character of cryptographic work. Cryptology ePrint Archive, Report 2015/1162, 2015. http://eprint.iacr.org/2015/1162.

[56] E. Ronen, C. O'Flynn, A. Shamir, and A. Weingarten. IoT Goes Nuclear: Creating a ZigBee Chain Reaction, Preliminary Draft Version 0.93, Nov. 2016. http://iotworm.eyalro.net/iotworm.pdf.

[57] J. Salowey, H. Zhou, P. Eronen, and H. Tschofenig. Transport layer security (TLS) session resumption without server-side state. IETF, 2008. RFC 5077.

[58] D. E. Sanger and J. H. Davis. Hacking linked to China exposes millions of U.S. workers. The New York Times, June 4, 2015. https://www.nytimes.com/2015/06/05/us/breach-in-a-federal-computer-system-exposes-personnel-data.html.

[59] S. Schultze. How the Nokia browser decrypts SSL traffic: A "man in the client". Freedom To Tinker Blog, January 11, 2013. https://freedom-to-tinker.com/blog/sjs/how-the-nokia-browser-decrypts-ssl-traffic-a-man-in-the-client/.

[60] G. Shah, A. Molina, and M. Blaze. Keyboards and covert channels. In *Proceedings of the 15th Conference on USENIX Security Symposium - Volume 15*, USENIX-SS'06, Berkeley, CA, USA, 2006. USENIX Association.

[61] J. Sherry, C. Lan, R. A. Popa, and S. Ratnasamy. BlindBox: Deep packet inspection over encrypted traffic. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, SIGCOMM '15, pages 213–226, New York, NY, USA, 2015. ACM.

[62] G. R. Simpson. Treasury tracks financial data in secret program. The Wall Street Journal, June 23, 2006. http://www.wsj.com/articles/SB115101988281688182.

[63] R. Singel. Whistle-blower outs NSA spy room. Wired, April 7, 2006. https://archive.wired.com/science/discoveries/news/2006/04/70619.

[64] D. X. Song, D. Wagner, and A. Perrig. Practical techniques for searches on encrypted data. In *Proceedings of the 2000 IEEE Symposium on Security and Privacy*, SP '00, pages 44–55, Washington, DC, USA, 2000. IEEE Computer Society.

[65] Vulnerability note VU#792004. CERT Vulnerability Notes Database. https://www.kb.cert.org/vuls/id/792004.

[66] F. Wang, J. Mickens, N. Zeldovich, and V. Vaikuntanathan. Sieve: Cryptographically enforced access control for user data in untrusted clouds. In *Proceedings of the 13th Usenix Conference on Networked Systems Design and Implementation*, NSDI'16, pages 611–626, Berkeley, CA, USA, 2016. USENIX Association.

[67] C. Wisniewski. Smart meter hacking can disclose which TV shows and movies you watch. naked security by SOPHOS, Jan. 8, 2012. https://nakedsecurity.sophos.com/2012/01/08/28c3-smart-meter-hacking-can-disclose-which-tv-shows-and-movies-you-watch/.

[68] Z. Zhou and T. Benson. Towards a safe playground for HTTPS and middle boxes with QoS2. In *Proceedings of the 2015 ACM SIGCOMM Workshop on Hot Topics in Middleboxes and Network Function Virtualization*, HotMiddlebox '15, pages 7–12, New York, NY, USA, 2015. ACM.