# Behind the Scenes: Uncovering TLS and Server Certificate Practice of IoT Device Vendors in the Wild


Hongying Dong
University of Virginia

Hao Shu
New York University

Vijay Prakash
New York University

Yizhe Zhang
University of Virginia

Muhammad Talha Paracha
Northeastern University

David Choffnes
Northeastern University

Santiago Torres-Arias
Purdue University

Danny Yuxing Huang
New York University

Yixin Sun
University of Virginia



## ABSTRACT

IoT devices are increasingly used in consumer homes. Despite recent works in characterizing IoT TLS usage for a limited number of in-lab devices, there exists a gap in quantitatively understanding TLS behaviors from devices in the wild and server-side certificate management.

To bridge this knowledge gap, we conduct a new measurement study by focusing on the practice of *device vendors*, through a crowdsourced dataset of network traffic from 2,014 real-world IoT devices across 721 global users. By quantifying the sharing of TLS fingerprints across vendors and across devices, we uncover the prevalent use of customized TLS libraries (i.e., not matched to any known TLS libraries) and potential security concerns resulting from co-located TLS stacks of different services. Furthermore, we present the first known study on server-side certificate management for servers contacted by IoT devices. Our study highlights potential concerns in the TLS/PKI practice by IoT device vendors. We aim to raise visibility for these issues and motivate vendors to improve security practice.



## CCS CONCEPTS

• **Security and privacy** → **Mobile and wireless security**; • **Networks** → **Network measurement**;

## KEYWORDS

Internet of Things, IoT, Measurements, Network security, Public Key Infrastructure, PKI, Transport Layer Security, TLS

**ACM Reference Format:**
Hongying Dong, Hao Shu, Vijay Prakash, Yizhe Zhang, Muhammad Talha Paracha, David Choffnes, Santiago Torres-Arias, Danny Yuxing Huang, and Yixin Sun. 2023. Behind the Scenes: Uncovering TLS and Server Certificate Practice of IoT Device Vendors in the Wild. In *Proceedings of the 2023 ACM Internet Measurement Conference (IMC '23), October 24–26, 2023, Montréal, QC, Canada.* ACM, New York, NY, USA, 21 pages. https://doi.org/10.1145/3618257.3624815





*IMC'23, October 24-26, 2023, Montréal, Canada*



ACM ISBN 979-8-4007-0382-9/23/10.
https://doi.org/10.1145/3618257.3624815


## 1 INTRODUCTION

Consumer Internet-of-Things (IoT) devices are increasingly used on home networks [43]. Given the rising number of security incidents involving compromised IoT devices [5, 15, 46] and the sensitive information that IoT devices collect from users [44], one critical question is how these devices use Transport Layer Security (TLS) and Public Key Infrastructure (PKI) to encrypt and protect their communications.

Several recent works investigate the TLS usage of in-lab IoT devices [4, 12, 37, 40]. We take a step further by performing *crowdsourced* measurements to capture much richer IoT TLS behaviors—from more device vendors and more instances of the same vendor—and provide a deeper understanding of vendor practices in the wild.

**Research questions.** We aim to answer two questions: (1) How do real-world IoT devices in consumers' homes use TLS? Unlike prior works in IoT TLS, our goal is to not only identify vulnerabilities (e.g., vulnerable ciphersuites), but also quantitatively measure the behavior of sharing and customizing of TLS libraries across devices and vendors at scale. (2) How do "IoT servers" (i.e., IoT-visited hosts) manage their digital certificates? In particular, we focus on device vendors who are oftentimes the domain owners and therefore responsible for maintaining the certificates. This is the *first known study* of server-side certificate management for IoT servers.

To this end, we utilize a crowdsourced dataset of IoT TLS traffic collected by IoT Inspector [23], consisting of 2,014 IoT devices of 286 models from 65 device vendors, across 721 users worldwide. To the best of our knowledge, this is the largest dataset of smart home IoT TLS traffic known to date. For the majority of IoT products in our dataset, there is more than one device of each product. For example, we have 75 *Wyze* Cameras, which could be of different models running different firmware images at different times. The dataset spans over 15 months from April 29, 2019 to August 1, 2020, and consists of 11,439 TLS ClientHellos. We also supplement this dataset with our own probing of IoT servers across multiple geolocations to construct the certificate dataset for our server-side analysis.

**Goal 1: Dissect the heterogeneity of IoT client-side TLS.** While many prior works perform TLS fingerprinting by matching against popular known TLS libraries [25, 37], we find that the majority (~ 98%) of IoT devices in our dataset do not have exact matches against these known libraries, which is likely due to the

heterogeneity of IoT device functionalities and configurations. To tackle the challenge of analyzing the large number of unmatched TLS fingerprints found in our 2,014 IoT devices, we employ metrics to quantify the level of *fingerprints customizing and sharing across vendors and across devices*. Our approach enables us to uncover (i) customization of TLS libraries from device vendors, i.e., TLS fingerprints only observed from devices of a given vendor; (ii) variance of TLS fingerprints from devices belonging to the same vendor; (iii) nonstandard TLS fingerprints (i.e., not matched against any known libraries) shared across different device vendors likely due to the shared software supply chain. We quantitatively measure the degrees of sharing and customization in all three scenarios as well as identify vulnerabilities in fingerprints.

**Goal 2: Characterize IoT server-side certificate management by vendors.** We supplemented the original IoT Inspector dataset, which only contains TLS ClientHellos, with server-side certificate data by initiating TLS connections to all the domains extracted from the Server Name Indication (SNI) field. To ensure reliability of the data, we initiated TLS connections from three different continents and cross-checked the obtained certificates for consistency. We segment our analysis by *vendor* and characterize how vendors (i) choose Certificate Authorities (CAs) for their domains, and (ii) maintain valid certificates. For example, *Canary Connect*, a home security system vendor, signs all certificates by itself without involving any third-party CA, while some vendors, such as *Belkin*, use *DigiCert* to sign all their domains. In addition, we identify concerns in the certificate validity period and highlight the insufficiency of Certificate Transparency (CT) in monitoring many IoT server certificates.

**Key contributions.** We leverage graph techniques combined with quantitative metrics to uncover TLS behaviors of IoT devices in the wild and present the first known study on certificate management by IoT servers or vendors. Note that our focus is on characterizing vendor's practices at scale from the perspective of network traffic rather than firmware analysis (whose source code is generally unavailable). Our key findings are:

(1) *Heterogeneity in TLS fingerprints across vendors:* 20% of vendors *only* have TLS fingerprints unique to themselves (i.e., not observed in any other vendors). 77.47% of all TLS fingerprints are only used by a single vendor. This observation highlights the high degree of vendor customization of TLS instances.
(2) *Sharing of TLS fingerprints across vendors:* aside from vendors, software and third-party applications installed on devices also contribute to device TLS fingerprints and likely lead to the sharing of customized TLS fingerprints across vendors. 17.42% of IoT servers (corresponding to specific applications) are found tied to their own unique fingerprints from multiple vendors (i.e., devices only exhibit a given fingerprint when connecting to a given server). This result highlights a potential risk factor of the IoT TLS ecosystem: devices with co-located TLS stacks from different types of services or functionalities are exposed to an increased attack surface and are likely affected by the least secure TLS stack.
(3) *Self management of IoT server certificates:* 9.86% of all server leaf certificates, affecting 391 devices, are signed by device vendors instead of a trusted public CA. Consequently, these certificates are not logged in Certificate Transparency (CT). Among them, 46.67% have a validity period longer than 5 years. This finding indicates the security risks of the long-lived vendor-signed certificates that fall out of public monitoring.

**Summary.** We quantitatively measure how TLS/PKI is deployed in the IoT context and the security concerns. Certain vendor practices, such as customized TLS libraries and server certificates issued by private CAs, make it challenging for security auditing. We aim to raise visibility for these issues, and encourage vendors to improve security practices and regulators to enforce compliance with corresponding rules.

**Open source.** In support of the community, we make available both an anonymized version of our IoT Inspector dataset and server certificate dataset at: https://github.com/hyingdon/acmimc23_iot.

## 2 RELATED WORK

We first discuss TLS/PKI work in the broader domains, and then focus on comparing with related works in IoT.

**TLS measurements.** There is a significant amount of works studying TLS. Zhu *et al.* [52] assessed TLS certificate revocation latency and pervasiveness using traffic data collected from a university. In 2018, Kotzias *et al.* [25] conducted a longitudinal study encompassing both passive monitoring and active network scans. Their research uncovered significant shifts within the TLS ecosystem in recent years. Brubaker *et al.* [8] instead took a different approach by generating synthetic certificates to test TLS certificate validation in different implementations, and Ma *et al.* [32] delved into the root store ecosystem, highlighting security risks linked to customized trust. Distinct from aforementioned studies, our work investigates the practices employed by IoT vendors when engaging with TLS, utilizing a crowdsourced dataset.

**Web.** Numerous works have studied TLS and PKI on the web. Akhawe *et al.* [3] studied how browsers validate TLS certificates and measured TLS errors on the web, while Liu *et al.* [31] explored TLS certificate revocation on the web. Cangialosi *et al.* [9], on the other hand, analyzed the prevalence of website-trust third-party hosting providers and the resulting impact on private key management practices regarding the web's HTTPS ecosystem. We note several key differences between IoT and the web: (1) Different from web clients like browsers that are relatively well-maintained by major developers [26], IoT clients are highly heterogeneous, e.g., we found only 2.55% IoT TLS fingerprints match known libraries including ones commonly used by the web [25]. (2) Such heterogeneity likely results in difficulties of monitoring and updating IoT clients, e.g., we found 42 devices prefer vulnerable ciphersuites most vs. 0 for major browsers [10]. (3) IoT servers may not expect web clients, and therefore they may have a heavier use of certificates issued by private CAs, resulting in a lack of CT logging.

**Mobile.** Prior works examining TLS/PKI on the mobile ecosystem [39, 47] largely focus on mobile operating systems (e.g., Android) and mobile applications. Our study, instead, focuses on a wide range of consumer IoT devices that exclude general computing devices such as phones, tablets, and computers.

**IoT and comparison with our work.** While there exist studies on TLS usage in IoT devices, the closest work to ours performed

both passive and active experiments on a maximum of 40 IoT devices across 6 categories in a controlled-lab environment [37]. They identified TLS vulnerabilities in devices, such as outdated protocols and weak ciphersuites, and revealed the sharing of fingerprints with other devices/applications. Despite some similarity in the client-side TLS analysis, our work differs in several aspects. First, our work also investigates server-side certificate management for servers contacted by IoT devices, which was not performed by any prior work. Second, though both works investigate fingerprint-sharing across devices, we take the fingerprint-sharing analysis to the next level by employing metrics to quantify the level of customization and sharing across vendors and across devices within the same vendor at a larger scale and at a finer granularity, as well as evaluating the security level of fingerprints to identify shared vulnerabilities and problematic vendors. Our analysis is enabled by our crowdsourced dataset of 2,014 devices from 65 vendors (compared to 40 devices of 28 vendors [37]), which reveals more diverse behaviors and provides a deeper understanding of vendor practices. To exemplify, our research shows that over 70% of vendors introduce at least one distinctive fingerprint that is not shared with other vendors, implying the existence of a substantial number of TLS instances specific to individual vendors. Additionally, our research highlights notable variations in security practices among vendors, as demonstrated by the diverse security levels associated with the fingerprints they employ.

**Table 1: Comparison with prior works on IoT TLS.**

| | | [4] | [40] | [23] | [37] | **This** |
|---|---|---|---|---|---|---|
| TLS | TLS parameters | ✓ | | ✓ | ✓ | ✓ |
| | TLS fingerprint | | | | ✓ | ✓ |
| Certificate | Client side | | | | ✓ | |
| | Server side | | | | | ✓ |
| Data Collection | Crowdsourced | | | ✓ | | ✓ |
| | Lab-based | ✓ | ✓ | | ✓ | |

Alrawi *et al.* [4] introduced a systematic methodology for conducting an SoK study on security characteristics, such as cryptographic protocols and TLS libraries, of home-based devices. They applied the methodology to assess 45 devices spanning diverse categories by looking into devices, as well as their cloud points, communication channels, and mobile applications. Ren *et al.* [40] undertook a study to characterize information exposure of 81 in-lab devices across networks, geographic regions, and device interactions, incorporating a combination of automated and manual experiments. Huang *et al.* [23] introduced IoT Inspector and harnessed a crowdsourced dataset to explore the utilization of TLS versions and communication endpoints. In contrast to our work, none of these studies focus on either the heterogeneity in devices' TLS instances or vendors' roles in the TLS behavior of IoT devices. As a summary, Table 1 captures how our work compares to the aforementioned works.

## 3 DATASET

To characterize how real-world IoT devices use TLS and PKI, we use a dataset of ClientHello packets collected from IoT Inspector from April 29, 2019 to August 1, 2020 [23].

IoT Inspector is an open-source tool that crowdsourced the network traffic from 54,094 devices across 5,404 users worldwide [23]. Each user has an option to label their devices with names of the devices and the manufacturers; in total, 12,993 devices had such labels across 2,465 users. For these labeled devices, we implemented rules to perform device identification by applying Natural Language Processing (NLP) techniques to user labels, based on the method described in Section 5.1 in the original IoT Inspector paper [23] (details of device label validation can be found in the paper as well).

For each of these 9,534 devices, IoT Inspector collects the TCP, UDP, or IP headers, along with the TLS versions, Server Name Indication (SNI), ciphersuites, and extension types for ClientHellos; IoT Inspector did not capture the full ClientHello packets to minimize potentially sensitive information collected [23]. We only observe ClientHellos on 2,298 devices (24.1%) across 771 users. We do not observe ClientHellos in the remaining devices likely because (i) they did not use TLS at all, or (ii) IoT Inspector was not running when the TLS connection happened. We perform analysis on ClientHellos in Section 4.

This dataset does not include any ServerHellos. To obtain server certificates, we need to establish TLS handshakes with the same servers that the IoT devices connected to (detailed in Section 5.1). To this end, we extracted SNIs from the ClientHellos and removed SNIs observed from two or fewer users to eliminate potential bias, which resulted in a set of 1,194 SNIs from 2,024 devices across 722 users. Due to the time lag between the ClientHello dataset and our probing (in 2022), 43 servers became unreachable. In the end, we were able to establish TLS handshakes with 1,151 SNIs across 2,014 devices from 721 users. We further cross-check our certificate dataset with additional sources and show that the time lag does not have a significant impact on certificate characteristics in Appendix C.4.2. We characterize IoT PKI leveraging this certificate dataset in Section 5.

We utilize two additional datasets to complete our study:

- *Lab dataset:* a dataset containing network traffic captured in the lab of 113 IoT devices belonging to 52 vendors from 2017 to 2021. This dataset is discussed in Appendix C.4 to cross-check our obtained certificates for consistency.
- *Smart TV and local device dataset:* a dataset containing network traffic captured in the lab, including smart TVs and 30 IoT devices on the local network. This dataset is analyzed in Section 6 as case studies.

## 4 IOT CLIENT-SIDE TLS ANALYSIS

We begin with a traditional TLS fingerprinting analysis to identify the TLS libraries used by IoT devices (Section 4.1). However, we find that the majority (~ 98%) of devices do not have exact matches against known libraries likely due to the heterogeneity of device vendors and functionalities. To tackle such challenge, we employ an approach focusing on the *shared (or unshared) fingerprints across vendors and devices*:

- Customization across vendors (Sec. 4.2): how unique are the fingerprints used by each vendor?
- Customization across devices (Sec. 4.3): how unique are the fingerprints used by devices within the same vendor?
- Shared supply chains or applications (Sec. 4.4): why do different vendors share the same fingerprints (that are non-standard libraries)?

Furthermore, we develop a new metric to measure the *similarity* between the customized TLS fingerprints and the known libraries in Appendix B.2. We also report additional TLS parameters in Appendix B.3.

## 4.1 Matching Fingerprints to Libraries

To infer which TLS library version is *potentially* used by a given device, we do the following steps: (1) We compile different versions of known TLS libraries. (2) For each version of a TLS library, we use its default TLS client (which typically comes with the source code) to connect to our server on port 443 and capture the TLS traffic. (3) We construct a fingerprint of this particular library version by concatenating {*TLS version*, *ciphersuites*, *extensions*}. (4) Using IoT Inspector's ClientHello data, we construct the TLS fingerprints for each of the 2,014 devices in our dataset, also by concatenating *{ciphersuites, extension types, TLS version}*. In contrast to existing works [37, 39], we are only using these three fields for TLS fingerprinting, as IoT Inspector does not collect the full ClientHello payload to reduce privacy risks [23]. Note that any TLS configuration will feature the three fields we base our fingerprints on. If two fingerprints already differ using these three fields, they will also differ with more fields. (5) We compare the fingerprint of an IoT device X against the fingerprints of the known TLS libraries. If the fingerprints are the same, we say that Device X *potentially* has the said TLS library version.

In total, we compiled (i) 19 versions of the OpenSSL libraries, (ii) 38 versions of the wolfSSL libraries, (iii) 113 versions of the Mbed TLS libraries, (iv) 5,591 versions of different curl versions compiled with different OpenSSL versions, and (v) 1,130 versions of curl with wolfSSL. This gives us 6,891 fingerprints from known libraries. Note that some consecutive library versions may have the same fingerprints. For example, OpenSSL versions $i$ through $j$ (where $i < j$) all have the same fingerprint $F$. In this case, if a device $X$'s fingerprint is identical to $F$, then we will use the highest TLS version $j$. We summarize the coverage of our compiled library fingerprints in Appendix B.1.

**TLS library matching results.** We extracted 903 unique fingerprints (i.e., *{ciphersuites, extensions, TLS version}* tuples) from the 2,014 IoT devices in our dataset. However, we find that only 23 fingerprints (2.55%, compared to 45% in prior IoT TLS study [37]) have an exact match with 16 known TLS libraries (14 from curl with OpenSSL and 2 from Mbed TLS, which indicates the possibility of customization of TLS configuration by IoT vendors. Concerningly, 14 (out of 16) libraries are no longer supported as of 2020, including severely outdated versions such as OpenSSL 1.0.0t (released in 2015) with many known vulnerabilities [53].

**Fingerprinting validation (case studies).** Due to the lack of ground truth and unavailability of device firmware, we perform manual search for manufacturer documents and discuss two case studies: (i) *Enphase* Solar Monitor: our fingerprinting matches it to OpenSSL 1.0.1e, which is consistent with manufacturer's disclosure that its products use OpenSSL 1.0.1e [17][1]; (ii) *Wyze* devices: our fingerprinting matches them to OpenSSL 1.0.2u. *Wyze* disclosed the usage of OpenSSL 1.0.2o or 1.0.2f [51], where all three OpenSSL 1.0.2 versions share the same 3-tuple fingerprint with 1.0.2u being the latest version.

**Takeaway.** The majority (~ 98%) of IoT TLS fingerprints differ from known libraries, which may be due to the customization of TLS libraries by vendors.

## 4.2 Customization Across Vendors

We first analyze the *uniqueness* of the fingerprints across vendors. We also define the security level of the ciphersuite in the fingerprint as follows, based on prior work and IETF documents [14, 42]:

- *Optimal*: equivalent to a modern web browser in terms of security [21].
- *Suboptimal*: non-ideal (e.g., non-PFS ciphers), but is not vulnerable to known attacks[2].
- *Vulnerable*: with algorithms that are vulnerable to known attacks, e.g., anonymous key exchange algorithms, export-grade ciphers, NULL encryption, RC2 and RC4 encryption, DES and 3DES encryption. We do *not* include MD5 or SHA-1 as vulnerable as they may not be problematic as ciphersuites even though they are problematic as a signature algorithm [19].

**TLS fingerprint overview by vendors.** Figure 1 shows the results. Numbered nodes represent device vendors (full mapping in Appendix B.6) and colored nodes represent fingerprints. Color blue suggests optimal/suboptimal fingerprints, and colors from orange to red suggest the inclusion of vulnerable versions and/or ciphersuites, with darker color indicating more vulnerable components. The security level of each fingerprint is also illustrated by the node size, with larger suggesting more vulnerable components. Edges are only between vendors and fingerprints, indicating that at least one device of the vendor uses the fingerprint.

**Table 2: Fingerprint degree distribution.**

| *Degree* | 1 | 2 | 3 - 5 | > 5 |
|---|---|---|---|---|
| %.Fingerprints | 77.47% | 11.43% | 8.32% | 2.78% |

We have two observations on the graph: (i) clusters of fingerprints unique to certain vendors, and (ii) many vulnerable ciphersuites (red nodes) for certain vendors. Next, we delve into these two observations and quantify them.

**How many vendors share a given fingerprint?** We show the *degree* of fingerprint, which corresponds to the number of vendors that use this fingerprint in at least one device, in Table 2. We observe that the majority (77%) of fingerprints are only used by a single vendor ($Degree = 1$).

**Customization across vendors.** We then measure from the vendor's perspective and answer the question: how *unique* are the fingerprints used by a given vendor? We define *degree of customization* ***across vendors*** ($DoC_{vendor}$) as

$$DoC_{vendor} = \frac{\#.fingerprints\ solely\ used\ by\ this\ vendor}{\#.fingerprints\ used\ by\ this\ vendor}$$

Figure 2 (red dashed line) shows the cumulative distribution function (CDF) for $DoC_{vendor}$ of all vendors. We observe that more than 70% of vendors use at least one unique TLS fingerprint that is not

[1]This device is not included in our TLS analysis because it doesn't meet our threshold (>= 2 users per device type).

[2]We consider non-PFS suboptimal rather than vulnerable because attacks targeting non-PFS ciphers have the prerequisite that the long-term key is compromised.

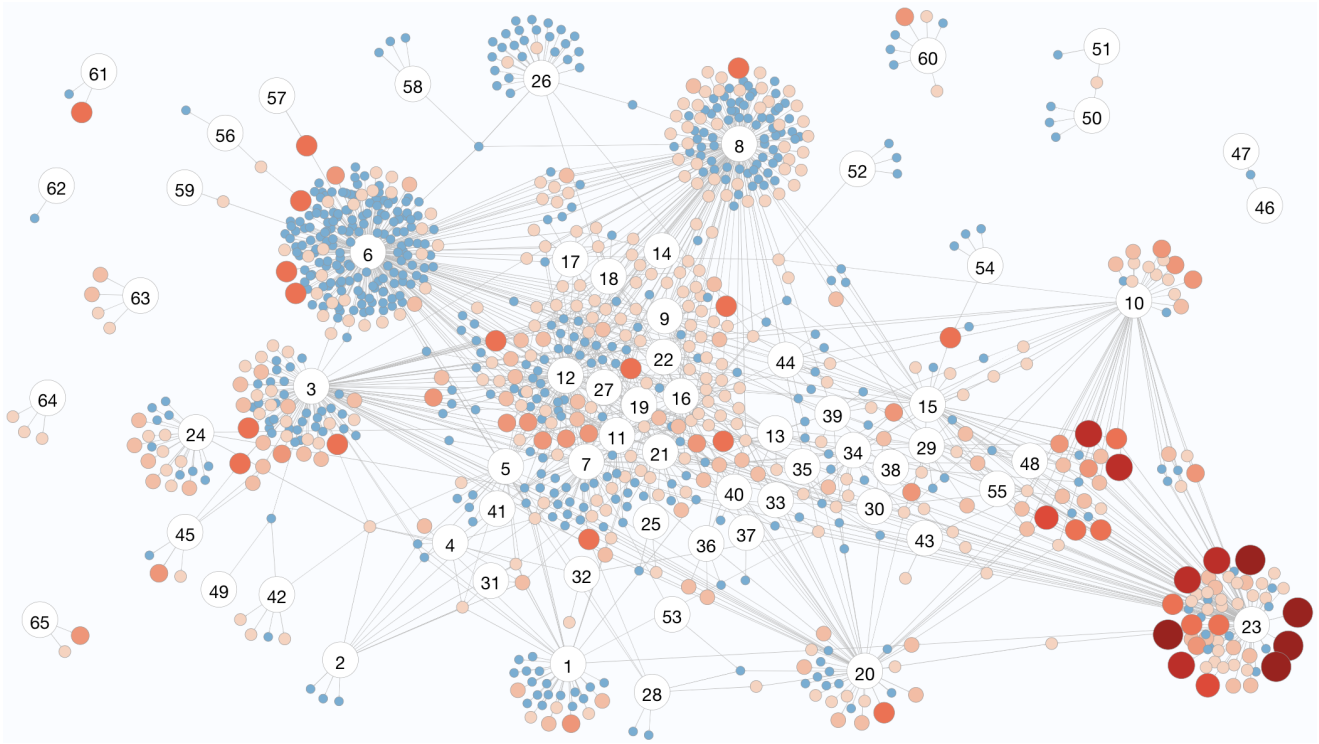


**Figure 1: TLS fingerprint overview by vendors. Nodes with colors represent unique fingerprints, with redder color indicating more vulnerable components. Numbered white nodes are vendors with full mapping in Appendix B.6.**

used by other vendors and 40% of vendors have a $DoC_{vendor}$ higher than 0.5, suggesting a considerable amount of vendor-specific fingerprints. The security level of vendor-specific TLS customization also varies. For example, *Synology* devices (vendor 23) use 22 unique fingerprints containing multiple vulnerable ciphersuite components[3]. On the contrary, most fingerprints exclusively used by *Sonos* devices (vendor 26) contain no vulnerable components. This highlights a significant difference in security practices from vendors when customizing/maintaining their TLS libraries.

**Vulnerabilities in ciphersuites.** 403 (44.63%) fingerprints contain at least one vulnerable component, 31.76% of which are used by multiple devices. In particular, 3DES is found to be the most seen vulnerable component, included in 376 (41.64%) fingerprints. Furthermore, 31 fingerprints include anonymous key exchange algorithms, export-grade ciphers, or even NULL encryption, and these fingerprints are proposed by 27 devices of 14 vendors[4]. We show details on the specific algorithms for each vendor in Appendix B.4.

## 4.3 Customization Across Devices

We now take a deeper look into devices within the same vendor: do devices share a similar set of fingerprints, or do they use largely disjoint fingerprints? Towards this end, we analyze the uniqueness of the fingerprints across devices by defining *degree of customization* for a *given device* of a specific vendor ($DoC$) as

$$DoC = \frac{\#.\textit{fingerprints solely used by this device within the vendor}}{\#.\textit{fingerprints used by this device}}$$

We then compute *degree of customization* ***across devices*** for a given vendor ($DoC_{device}$) as the *mean DoC value* across all devices for this vendor. Figure 2 (blue solid line) shows the CDF for $DoC_{device}$ of all vendors. We also show a detailed $DoC$ distribution of all devices in Appendix B.5.

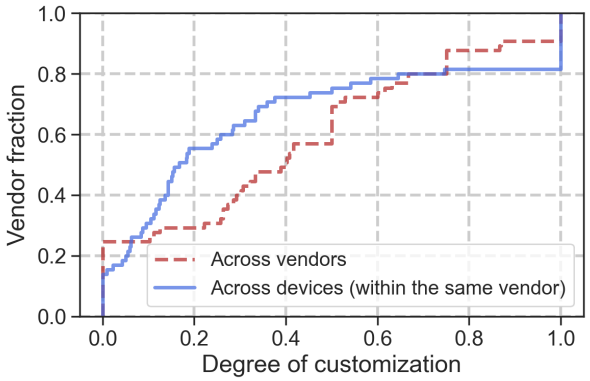


**Figure 2: Degree of TLS fingerprint customization. Higher degree indicates more unique fingerprints.**

[3]Anonymous key exchange algorithms, export-grade ciphers, NULL encryption, RC2/RC4 encryption, DES and 3DES encryption.
[4]*Synology, Western Digital, TP-Link, Sony, Amazon, HP, LG, Samsung, QNAP, Vizio, Philips, Lutron, Amcrest,* and *Google*. Sorted by # of involved devices.

Interestingly, there is a similar degree of customization across devices within the same vendors as well, where devices of close to 20% vendors have $DOC_{device} = 1$ (completely disjoint sets of fingerprints). Table 3 shows statistics of fingerprints across devices from top 10 major vendors with the most fingerprints. We can see that only a small fraction of fingerprints are shared across devices within the same vendor, which are likely vendor-specific libraries. The vast majority are only used by a single device, potentially due to application-specific libraries from installed applications on devices. We next explore this observation by taking a deeper look into *Amazon* devices (top 1 vendor in our dataset). We will further investigate the possible reasons in Section 4.4.

**Table 3: Heterogeneity in fingerprints across devices within the top 10 vendors.**

| Vendor | #.Fingerprints | %.Fingerprints shared by 10 or more devices | %.Fingerprints used by 1 device |
|---|---|---|---|
| Amazon | 244 | 12.30% | 68.85% |
| Google | 172 | 11.05% | 65.12% |
| Synology | 107 | 3.74% | 67.29% |
| Samsung | 104 | 9.62% | 60.58% |
| Sony | 97 | 6.19% | 57.73% |
| LG | 54 | 3.70% | 64.81% |
| Western Digital | 49 | 0.00% | 95.92% |
| Nvidia | 43 | 9.30% | 46.51% |
| TP-Link | 39 | 2.56% | 87.18% |
| Roku | 38 | 23.68% | 63.16% |

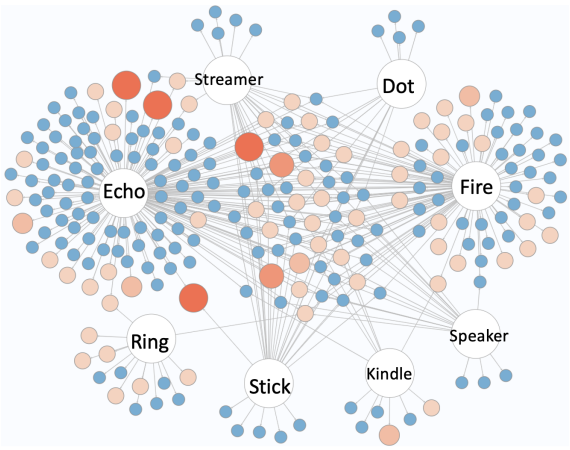


**Figure 3: Fingerprints by different types of *Amazon* devices. Colored nodes correspond to fingerprints and white nodes represent device types.**

**Deeper look into *Amazon* devices.** One potential cause for heterogeneity across devices from the same vendor is the differences in device types/functions. Figure 3 depicts the fingerprints (color indicates vulnerabilities) of each *Amazon* device type. We observe clusters of unique fingerprints surrounding each device type, where a total of 180 fingerprints are exclusively associated with only one device type. This result indicates one potential cause of heterogeneity across devices—the device type and hence its associated functions.

**Figure 4: Fingerprints by *Amazon* Echo devices. Black dots are devices. Colored nodes are fingerprints.**

**Deeper look into *Amazon* Echo devices.** We notice a particularly large cluster of fingerprints from *Amazon* Echo devices in Figure 3, which contrasts vastly with findings in prior work [37] where only 8 fingerprints were found from *Amazon* Echos. Thus, we further investigate *Amazon* Echos to answer the question whether the heterogeneity in fingerprints exists even *within the same device type*. Figure 4 shows the result, where each black dot corresponds to a single device and colored nodes correspond to fingerprints. We can see multiple device-fingerprint clusters, which could be due to differences in device updates/versions or installation of third-party applications/services.

**Takeaway.** Our analysis reveals the customization of TLS libraries across vendors and devices. Such practice makes it more challenging to continuously monitor and maintain the TLS libraries, which likely contributes to the large number of vulnerable ciphersuites we observe.

## 4.4 Shared Fingerprints Across Vendors

While we focus on the customization/uniqueness of TLS fingerprints across vendors in Section 4.2, we also observe 203 out of 903 (22.48%) fingerprints that are shared across multiple vendors. We aim to answer the question: why are these non-standard fingerprints (i.e., not matched to known libraries) used by multiple vendors?

Towards this end, we use Jaccard metric [24] to calculate pairwise similarities for all vendor nodes in Figure 1 based on their fingerprints. We employ Jaccard metric because it captures the differences in both fingerprints and the number of fingerprints,

which we believe is a contributing factor to the similarity assessment. When two sets of fingerprints exhibit a substantial difference in size, we classify them as dissimilar, even if the smaller set is a subset of the larger set. In other words, the more fingerprints that two vendors have in common, the higher the similarity value is. The similarity becomes 1 when two vendors have an identical set of fingerprints. Table 4 shows the results for vendor tuples with similarity value $\geq 0.2$.

**Table 4: Vendor tuples with Jaccard similarity $\geq 0.2$.**

| Jaccard similarity | Vendor tuple |
|---|---|
| 1 | {*HDHomeRun, Silicondust*} |
| [0.7, 1) | {*Sharp,TCL*} |
| [0.4, 0.7) | {*Arlo, NETGEAR*} |
| [0.3, 0.4) | {*Onkyo, Pioneer*}, {*Brother, Sharp*}, {*Insignia, Sharp*}, {*Bose, Texas Instruments, Skybell*}, {*Sense, Texas Instruments*}, {*Insignia, TCL*} |
| [0.2, 0.3) | {*Nvidia, Xiaomi*}, {*Brother, TCL*}, {*Arlo, iRobot*}, {*Dish Network, Skybell*}, {*Denon, Marantz*}, {*Synology, Western Digital*} |

We discuss two potential reasons of such sharing behavior:

- *Sharing due to shared supply chains:* this could happen when the different vendors are owned by the same supply chain within partnered companies.
- *Sharing due to shared applications:* devices from different vendors may install the same software or application, which has its own customized TLS library.

**Servers as a proxy for applications.** Given that we do not have any information on the applications (due to TLS-encrypted traffic), we use *servers* (through SNIs) as a proxy. The intuition is that many servers are application-specific, where the servers are only visited by devices with the corresponding applications installed. Excluding fingerprints that can be matched to standard libraries, we find 17.42% SNIs that are tied to server-specific fingerprints and are visited by multiple devices who share the same fingerprint when visiting a given server. After excluding cases where one device is involved to avoid potential outliers, 37 SNIs are found to be linked to server-specific fingerprints across devices from multiple vendors, as shown in Table 5. Note that for simplicity, we show second-level domains rather than Fully Qualified Domain Names (FQDNs).

**Key observations.** Based on the aforementioned reasons along with Table 4 and 5, our findings are:

- Devices of the same company, though branded as different vendors, share the same fingerprint when visiting the company's servers. For instance, {*Arlo, NETGEAR*} devices and {*HDHomeRun, SiliconDust*} devices are tied to specific fingerprints, respectively. This is consistent with the observation in Figure 1.
- Cooperation of companies is likely to result in the sharing of integrated TLS configuration. *Amazon*, *IKEA*, and *Sonos* devices visit *sonos.com* domains with the same client fingerprint, and these three vendors all make *Sonos*-enabled speakers; it is likely that the TLS library is a part of *Sonos*' software development kit (SDK) to provide the smart speaker functionality. Further, given that *Sonos* uses the Pandora API in the back-end for music streaming, it is presumably an integrated TLS stack.
- Co-op devices of partnered companies, such as *Roku* with *Insignia*/*Sharp*/*TCL*, are likely to share the same TLS configuration, and thus sharing the same client-proposed fingerprint when visiting servers.
- Requirements of third-party applications or services could result in the sharing of client-side TLS instances. For example, 5 *nflxvideo.net* servers all link to the same fingerprint, and this fingerprint is used by multiple devices across different vendors. On the other hand, servers associated with different functionalities/services, even within the same second-level domain, may require dissimilar client TLS configurations. Examples include differing *roku.com* servers tied to different device fingerprints.

**Takeaway.** Our analysis identifies the sharing of customized TLS libraries due to shared supply chains or third-party applications. Not only does such sharing shed light into the software supply chains of various IoT devices (possibly revealing the software bills-of-materials), it also highlights a potential security risk: if these upstream libraries or SDKs are vulnerable, the downstream devices, as shown in Table 5, may likely be affected (e.g., 118 *Roku* devices are affected by vulnerable ciphersuites containing RC and 3DES).

# 5 SERVER KEY INFRASTRUCTURE

Having measured the TLS behaviors of IoT devices, we now delve into the *IoT servers*, i.e., servers which IoT devices connect to. Vulnerabilities in server certificates have direct impact on the security of the TLS connections, and one server could affect a large number of IoT devices.

We begin by describing our IoT server certificate dataset in Section 5.1. Then, we aim to answer the following questions:

- Who are the certificate issuers/signers? (Section 5.2)
- Are the certificate chains valid? (Section 5.3)
- Are the certificates logged in Certificate Transparency (CT) logs? (Section 5.4)

Similar to the TLS analysis, we seek answers to the above questions by focusing on the *vendors*, as vendors are likely the key decision makers on the certificate management for IoT servers.

## 5.1 Certificate Dataset

So far, we have utilized the TLS ClientHello dataset from IoT Inspector, as described in Section 3 and 4. Although there is no ServerHello or certificate data in the IoT Inspector dataset, we extract over one thousand distinct Server Name Indications (SNIs) from TLS ClientHellos of the 2,014 IoT devices. These SNIs represent the real-world servers visited by consumer IoT devices *in the wild*.

With the extracted SNIs, we construct our server certificate dataset by using our own TLS client to establish multiple connections to port 443 of each SNI from 3 different global vantage points in April 2022, during which we capture the ServerHello and certificate chain. In total, we successfully captured 842 leaf certificates from 1,151 IoT servers. Table 6 summarizes the certificate dataset.

**IoT servers.** We observe a long-tail distribution on Second-Level Domains (SLDs) of the 1,151 IoT servers, with a total number of 357 distinct SLDs. Each SLD is contacted by an average of 24.42 unique devices, with a maximum of 556 and a median of 7 devices. We summarize the most popular 30 SLDs in Table 15 in Appendix C.1.

**Table 5: Servers linked with particular client fingerprint across multiple vendors. Each row represents a unique {*second-level domain, fingerprint*} tuple, and each bordered cell corresponds to one unique fingerprint.**

| Second-level domain | #.FQDNs | Vulnerability in fingerprint | #.Visiting devices | Device vendors |
|---|---|---|---|---|
| *arlo.com* | 2 | - | 13 | Arlo,NETGEAR |
| *netgear.com* | 1 | | 4 | Arlo,NETGEAR |
| *hdhomerun.com* | 2 | - | 8 | HDHomeRun,SiliconDust |
| *pandora.com* | 1 | - | 8 | Amazon,Sonos |
| *sonos.com* | 5 | | 75 | Amazon,IKEA,Sonos |
| *mgo.com* | 2 | RC,3DES | 37 | Insignia,Roku,Sharp,TCL |
| *mgo-images.com* | 2 | | 37 | Insignia,Roku,Sharp,TCL |
| *ravm.tv* | 1 | | 35 | Insignia,Roku,TCL |
| *roku.com* | 8 | | 118 | Insignia,Roku,Sharp,TCL |
| *roku.com* | 6 | 3DES | 31 | Insignia,Roku,Sharp,TCL |
| *cast4.audio* | 1 | 3DES | 7 | Onkyo,Pioneer |
| *googleapis.com* | 1 | - | 6 | Nvidia,Sony |
| *nflxvideo.net* | 5 | - | 5 | Amazon,LG |

**Table 6: IoT server certificate dataset.**

| | | | |
|---|---|---|---|
| #. Servers (FQDNs) | 1151 | #. Leaf certificates | 842 |
| #. Issuer organizations | 33 | #. Device vendors | 65 |

**Certificate sharing.** Note that there are more servers (FQDNs) than the number of distinct leaf certificates. This is due to the sharing of certificates across different server names, as exemplified by 29 distinct *Google* servers of 6 different SLDs that utilize the identical leaf certificate. We observe an average of 1.72 servers per certificate (i.e., 1.72 FQDNs present the same leaf certificate on average), with a variance of 5.53 servers and a max of 32 servers per certificate. We further investigate the IP addresses of different servers sharing the same certificate, and find that 547 (64.96%) certificates are shared across multiple IPs, with a mean of 5.43 and a max of 93 IP addresses per certificate. It is noteworthy that while servers sharing the same leaf certificate typically belong to the same organization, these servers usually have diverse service functions. For instance, *NVIDIA*'s server *services.tegrazone.com* and *ota.nvidia.com* use the same leaf certificate. The former is responsible for delivering gaming services to application users, while the latter manages user-firmware updates. This phenomenon highlights one security risk – if the shared certificate is invalid/compromised, it affects multiple servers and significantly more devices connecting to those servers. In particular, when servers sharing the problematic certificate are responsible for different functionalities, an even broader spectrum of users becomes vulnerable to security risks. Additionally, vendors may encounter greater difficulties in mitigating the risk due to the widespread impact of affected servers.

Next, we discuss two potential concerns about the certificate dataset and how we address them.

**Geographical locations of the TLS client.** Servers, especially CDN servers, may respond with different certificates based on the location of client IP address. To address this concern, we use three TLS clients located in U.S., Europe, and Asia, respectively, to initiate connections to each server. Upon analyzing the certificates obtained from the three clients, we find that the difference is negligible. We discuss in further details in Appendix C.4.1. For simplicity, we will use the certificates obtained by the client located in New York, US, for the remaining analysis in this section.

**Difference in certificates over time.** While we obtain the list of SNIs from real-world IoT devices, we have to capture the certificates using our own TLS client. One potential concern is that the certificate characteristics of the same server may change over time. To address this concern, we utilize an additional dataset consisting of ServerHello and certificate data captured at 113 IoT devices from 2017 to 2021 in a university lab, which we refer to as the *lab dataset*. We identify common IoT devices between our real-world dataset and the lab dataset, and then compare the certificates and issuers for the same IoT servers. Our results show that both the certificate issuers and other certificate characteristics are largely consistent across the two datasets for servers *in common*, suggesting that the time lag likely does not have a significant impact on the certificate characteristics. We discuss the comparison in further detail in Appendix C.4.2.

## 5.2 Certificate Issuers

The certificate issuer plays an important role in the PKI as it is responsible for establishing the chain of trust and certificate revocation. It is critical for certificate issuers to follow best practices to prevent adversaries from tampering the certificates. We extracted the certificate issuer of leaf certificates from our certificate dataset. We categorize leaf-certificate issuers into two groups:

- *Public trust CA*: as an organization or company, provides certificate signing services to domain owners, or has its root certificate in major trust stores.
- *Private CA*: only signs its own domains and its root certificate is not in major trust stores.

For example, *Roku* is a private CA as it only signs its own domains and its root certificate is not in major trust stores (despite being trusted by *Roku* devices). On the contrary, *Apple* is a public trust CA since its root certificate is in major trust stores, even though it usually signs its own domains.

**Certificate issuer results.** We identified 33 distinct certificate issuer organizations for the 1,151 certificates based on CCADB [35], involving 2,014 devices and 65 device vendors. Figure 5 shows the issuers on the Y axis, with respect to the device vendors on the X

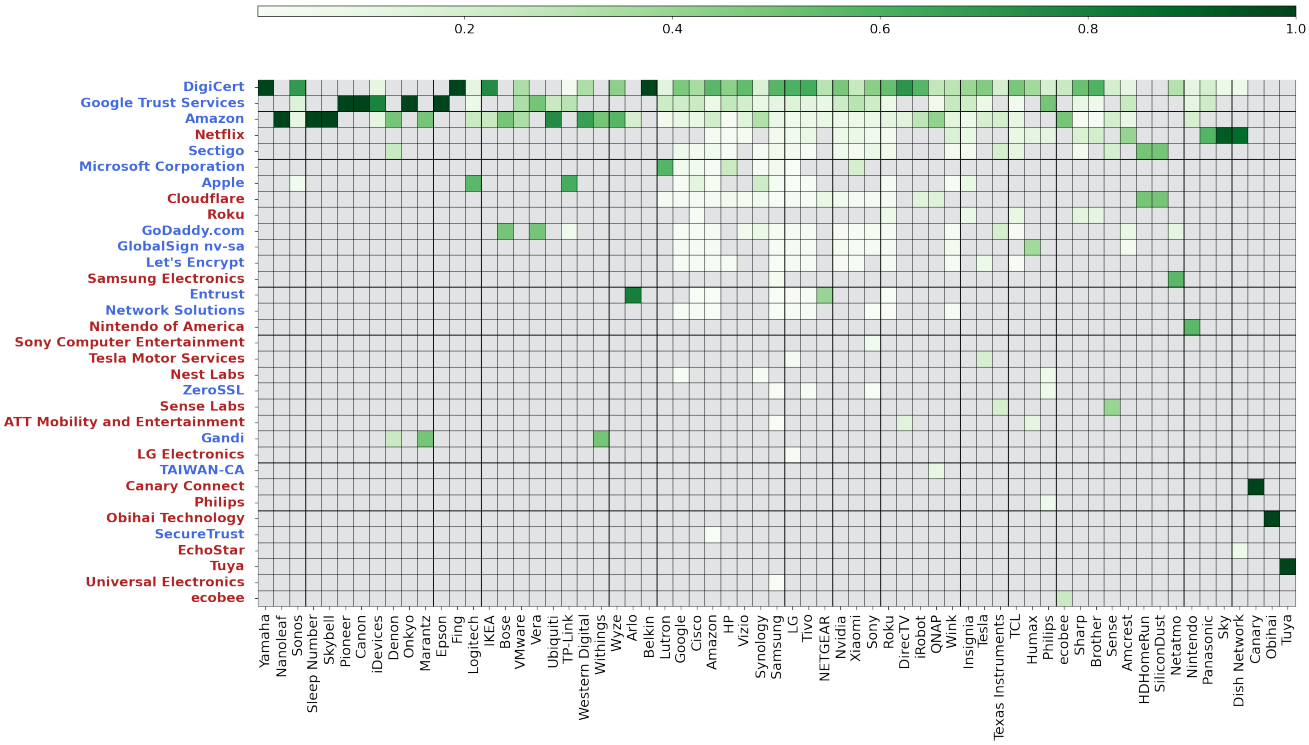


**Figure 5: Issuers of certificates sent from servers visited by IoT devices. X-axis shows vendors of devices, and is sorted by the prevalence of public trust CAs for visited IoT servers in a descending order. Y-axis shows certificate issuers for visited IoT servers, sorted by the number of issued leaf certificates in a descending order. Public trust CAs are shown in blue on the y-axis while private CAs are labeled in red.**

axis. The public trust CAs are colored in blue while the private CAs are in red. Color of squares reveals the ratio of certificates by each issuer among devices from a given vendor, and squares of value 0 are grayed out. In other words, values in each column should sum up to 1.

**Public trust CAs.** The majority of leaf certificates are issued by public trust CAs, with *DigiCert* signing 47.26% of all certificates. Devices from 31 vendors (starting from *Yamaha* on the left to *Wyze* in Figure 5) only visit servers whose leaf certificates are issued by public trust CAs. These servers behave very similar to the web PKI.

**Private CAs.** Despite the prevalent usage of public trust CAs, many private CAs are observed as well, signing 9.86% of all leaf certificates. Most private CAs are device vendors themselves, who sign leaf certificates for their own servers. In total, devices from 16 vendors visit servers signed by the vendors themselves[5], and these servers are mainly visited by their own devices. In particular, devices from 3 vendors – namely, *Canary*, *Tuya*, and *Obihai* – only visit servers with leaf certificates signed by their own vendors.

**Servers visited by devices across vendors.** It is worth noting that issuers like *Amazon*, *Google Trust Services*, *Microsoft Corporation*, and *Apple* appear more frequently in certificates used by IoT servers compared to the web [2, 13, 22]. This is expected since these issuers are, at the same time, well-known IoT device manufacturers and/or cloud providers that offer services to IoT devices. These issuers are categorized as public trust CAs due to the presence of their root certificates in major trust stores.

One interesting case is *Netflix*, who is not a device vendor but signs many certificates for *Netflix* servers that are visited by various devices, especially smart TVs. We discuss security issues in *Netflix*-signed certificates in Section 5.3 and 5.4, which impact many IoT devices.

**Takeaway.** 16 device vendors sign leaf certificates for their own servers which are visited by their own devices. As we will discuss next, such vendor-signed certificates often fall out of public monitoring and can have a profound impact on the security of the devices' TLS connections.

## 5.3 Certificate Chain Validation

We performed certificate chain validation using Zeek [38], one of the most popular tools in network security monitoring. By default, Zeek uses *Mozilla* trust store [36], and we further supplemented it with two other major trust stores from *Apple* [7] and *Microsoft* [33].

**Chains with validation failure.** Table 7 describes certificate chains with validation failure due to the lack of the presence of root certificates in both trust stores and the server-presented chains, which involves 45.78% of leaf certificates signed by private CAs. It is possible that the root is hard-coded and trusted by the device as RFC 5246 [11] specifies that the root certificate may be omitted from the

[5] *Roku*, *Samsung Electronics*, *Nintendo*, *Sony*, *Tesla*, *Nest Labs*, *Sense Lab*, *ATT Mobility and Entertainment*, *LG*, *Canary Connect*, *Philips*, *Obihai Technology*, *EchoStar*, *Tuya*, *Universal Electronics* and *ecobee*.

**Table 7: Certificate chains with validation failure.**

| Domain | #. FQDN(s) | Leaf certificate issued by | Chain length | Visited by #.devices | Visited by devices of |
|---|---|---|---|---|---|
| *netflix.com* | 6 | Netflix | 2 | 278 | Amazon, Amcrest, Brother, Dish Network, HP, Humax, Insignia, LG, Nvidia, Panasonic, Philips, Roku, Samsung, Sharp, Sky, Sony, TCL, Tivo, Vizio, Wink, Xiaomi |
| *roku.com* | 14 | Roku | 1, 2, 3 | 131 | Brother, Cisco, Insignia, Roku, Sharp, TCL |
| *nest.com* | 3 | Nest Labs | 2 | 65 | Google, Philips, Synology |
| *samsungcloudsolution.net* | 7 | Samsung Electronics | 1 | 43 | Netatmo, Samsung |
| *meethue.com* | 1 | Philips | 2 | 31 | Philips |
| *nintendo.net* | 4 | Nintendo | 1 | 24 | Nintendo |
| *samsungcloudsolution.com* | 4 | Samsung Electronics | 1 | 24 | Netatmo, Samsung |
| *playstation.net* | 1 | Sony Computer Entertainment | 1 | 13 | Sony |
| *sonyentertainmentnetwork.com* | 1 | Sony Computer Entertainment | 1 | 9 | Sony |
| *tesla.services* | 4 | Tesla Motor Services | 1, 2 | 6 | LG, Tesla |
| *amazonaws.com* | 1 | **DigiCert** | 2 | 4 | Vizio |
| *obitalk.com* | 1 | Obihai Technology | 1 | 4 | Obihai |
| *samsungrm.net* | 1 | Samsung Electronics | 1 | 4 | Netatmo, Samsung |

* Issuers in bold are public trust CAs.

chain when the remote end already possesses it. However, a recent study shows that it is also likely that servers present broken certificate chains and devices do not perform certificate chain validation properly [37], making the connection vulnerable to interception attacks. Because the lack of abilities to continuously monitor and maintain certificates can significantly increase the attack surface of underlying TLS connections, whether these public-*not*-trust certificates provide optimal security practices remains a question. Once compromised, the inability of public-*not*-trust issuers to quickly replace or rotate the certificate may open the door to attackers.

**Table 8: Details on expired certificates.**

| Domain | Not after | Issued by | Visited by #. devices | Vendor |
|---|---|---|---|---|
| *skyegloup.com* | 07/31/2018 | Gandi | 7 | Denon, Marantz |
| *wink.com* | 04/17/2019 | COMODO | 11 | Samsung, Wink |

**Expired certificate.** Table 8 shows the expired certificates, affecting 18 devices from vendors including *Samsung* and *Wink*. Note that these certificates *had already expired at the time of the IoT Inspector capture* (from 4/29/2019 to 08/01/2020). In other words, consumer devices in the wild were actively connecting to these servers even though their certificates had long expired, indicating the lack of proper validation by these devices.

**Private root CA and self-signed certificates.** We show the details of certificates with such status in Table 14 in Appendix C.2. In particular, the "self-signed certificate" status indicates that the leaf certificate has identical issuer and subject and is issued by a private CA. One interesting case is *log.samsunghrm.com*, which uses a certificate chain consists of two certificates that are exactly the same, with the identical issuer and subject **.samsunghrm.com*. Such signing practice makes it very difficult for certificate revocation, which is further exaggerated by the absence of logging in Certificate Transparency logs (Section 5.4).

**Common Name mismatch.** Such error occurs when the subject Common Name (CN) value or Subject Alternative Name (SAN) extension value of a server's leaf certificate is not associated with the SNI, making the connection vulnerable to a redirection or spoofing attack to allow host impersonation. We observe that server *a2.tuyaus.com* uses a certificate signed by the device vendor *Tuya* that does *not* include its hostname in either subject CN or SAN field. This server is regularly visited by 3 *Tuya* devices.

**Takeaway.** Many IoT device vendors, including major vendors like *Samsung* and *Wink*, do not properly manage certificates for their own servers that are regularly visited by many consumer IoT devices. Examples of such poor practice include long-expired certificates and lack of proper certificate chains, which could increase security risks.

## 5.4 CT and Validity Period

Certificate Transparency (CT) is an open-source framework of Internet security standard designed for monitoring and auditing TLS certificates that aims to allow adequate identification of mis-issued certificates in a manner of maintaining a system of public logs. CT is described in an experimental RFC 6962 [29] in 2013, and is further pushed forward by *Google* soon after. Although public trust CAs are incentivized, rather than required by any policies, to log issued certificates in CT, the fact that CT is enforced by several major browsers including Chrome[45] and Safari[6] makes CT mandatory for certificates issued by public trust CAs as of 2021.

On the other hand, IoT devices are unlikely to enforce CT logging for certificates. We aim to study two questions here: (1) Do public trust CAs still submit certificates to CT for IoT servers? (2) Unlike certificates issued by private CAs (without a trusted root) that

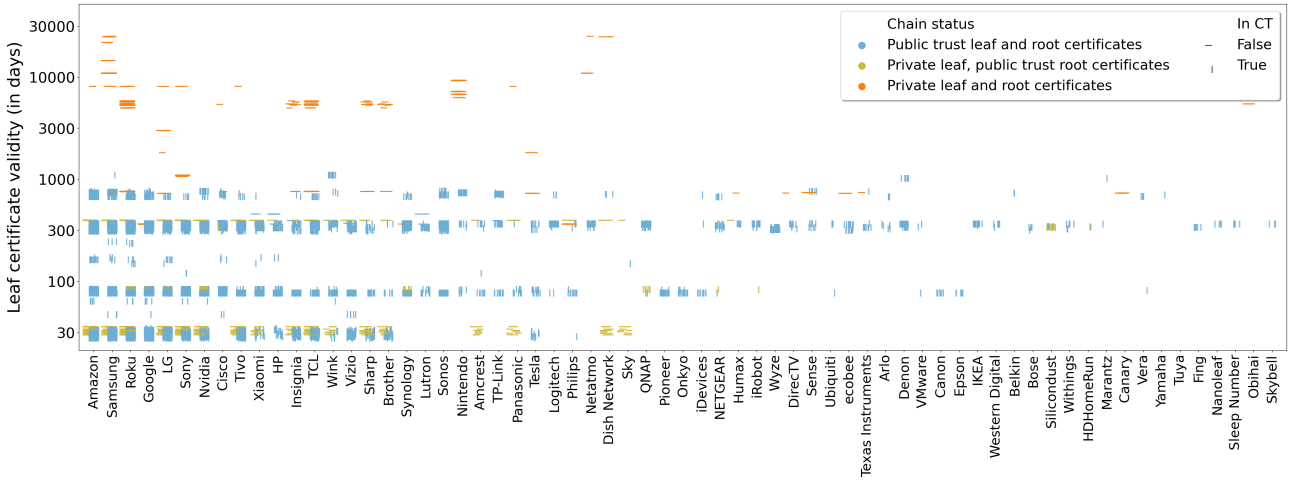


**Figure 6: Certificates of servers visited by IoT devices. X-axis shows device vendors, and is sorted by the number of appearance of manufactured devices in the dataset. Y-axis shows certificate validity period in days. Colors suggest different chain status: blue represents "public trust leaf and root certificates", yellow speaks "private leaf, public trust root certificates", and orange indicates "private leaf and root certificates". The shape of point shows the corresponding status in CT, with the horizontal line denoting absence and the vertical line signifying its presence.**

cannot be submitted to CT, leaf certificates issued by a private CA but chained to a trusted root *can* be submitted to CT. Are any of such certificates logged in CT?

**CT dataset.** To investigate the logging of leaf certificates in CT, we extracted 4,949 distinct {*server*, *leaf certificate*, *device vendor*} tuples from our certificate dataset. We queried each certificate in CT logs [30] to check its existence.

Figure 6 shows whether a certificate is in CT together with the chain status (i.e., public trust leaf and root certificates, private leaf and root certificates, *etc.*) and the certificate validity period for each device vendor.

**Validity periods.** We can see that nearly all leaf certificates issued by public trust CAs have validity periods less than 1,000 days, while certificates signed by private CAs have validity periods exceeding way beyond 1,000 days. In addition, we performed multiple scans throughout 2021 and found no reissuance of these long-lasting private issuer-signed certificates.

The primary motivation of issuing certificates with shorter validity is that renewing more frequently allows for rolling in security updates faster while ensuring the regeneration of keys frequently as well. However, several private CAs (i.e., device vendors) have issued certificates for extremely long validity periods, ranging from 19.8 to 100 years[6].

The pattern of vendor-signed certificates with long validity periods highlights one concerning practice: vendors sign and install certificates on their own IoT servers, and likely do not plan to ever update the certificates. Long validity period leads to heightened security risks, while shorter validity period facilitates algorithm upgrades and faster certificate/key replacements, especially during attacks [20, 49]. In fact, *Let's Encrypt*, a public trust CA that issues certificates for free, only provides a 90-day validity period mainly to limit damage from key compromise and mis-issuance [1]. One possible reason that leads to the long validity periods by vendors is the lack of expertise in certificate maintenance, i.e., it is way easier to "set it and forget it" instead of performing certificate updates and risking bricking devices. While there is always a tension between security and availability, this has already played out in the Web PKI with the adoption of ACME (Automated Certificate Management Environment) protocol [16] that is initiated by *Let's Encrypt* and streamlines certificate management. We hope the same can be done for the IoT PKI if device vendors choose to adopt a similar model and automation framework for certificate management.

**CT logging.** Most certificates issued by public trust CAs are logged in CT as expected. However, we found 8 certificates issued by public trust CAs, including *Microsoft* (4), *Apple* (2), *Sectigo* (1) and *DigiCert* (1), do *not* appear in CT. Meanwhile *none* of the certificates signed by private CAs with valid chains to public trust roots are logged in CT (which *can* be submitted to CT if they choose to). They involve popular device vendors such as *Roku*, *Amazon*, *Samsung*, *Tivo*, *Xiaomi*, and many others. This result indicates that these IoT servers are indeed *for IoT only*, which are not used by web clients, otherwise the certificates should be logged in CT due to browser CT enforcement. This finding also reinforces the concern of lack of transparency and monitoring for IoT certificates without any CT logging or enforcement. We further show the relation between invalid chains and CT appearance in Appendix C.3.

***Netflix.*** We observe that *Netflix* is the only private CA that signs leaf certificates with validity periods as short as 30 days and as long as 8,150 days, as shown in Table 9. The certificate lasting 8,150 days is for *appboot.netflix.com* and *cloud.netflix.net*. The leaf certificates with shorter validity periods have a valid chain to a trusted public CA, while the chain with long-valid certificates are completely self-signed by *Netflix*. In addition, *none* of the *Netflix*-signed leaf certificates are logged in CT.

[6] *Tuya* (36,500 days), *Samsung Electronics* (25,202 and 10,950 days), *EchoStar* (24,855 days), *Universal Electronics* (21,946 days), and *Nintendo* (9,300 and 7,233 days).

**Table 9: Variance in certificate validity periods by *Netflix*.**

| Leaf issuer | Leaf validity days | Topmost issuer | #.Cert | In CT |
|---|---|---|---|---|
| Netflix Primary Certificate Authority | 8150 | Netflix Primary Certificate Authority | 3 | False |
| Netflix Public SHA2 RSA CA 3 | 30,31,32,33, 34,36,396 | VeriSign Class 3 Public Primary Certification | 13 | False |

**Takeaway.** Leaf certificates signed by device vendors tend to have very long validity periods, the risk of which is further heightened by the lack of logging and monitoring through CT. We urge the private CAs (e.g., device vendors) to adopt an automation framework such as ACME [16] to facilitate certificate management, and motivate the community to develop an auditing mechanism that can accommodate certificates issued by private CAs.

## 6 CASE STUDIES

We take a deeper look into the key infrastructure in smart TVs and communications between IoT devices on the local network. For the following case studies, we use additional traffic datasets directly captured from the devices in the lab (rather than by IoT Inspector).

### 6.1 Smart TVs

Smart TVs are of particular interest among IoT devices given the rich information it collects from users (e.g., viewing history, locations) [34]. We perform a case study on smart TVs by using two additional datasets with network traffic directly captured from *Amazon* and *Roku* smart TVs in 2019 [34]. We find that while servers managed by third-party channels/applications use certificates from public trust CAs, the vendor-owned servers all use certificates signed by the vendor, where *Roku* signs certificates with validity period over 13 years and one *Amazon* server provides an expired certificate.

**Servers managed by third-party channels or applications.** The majority of such servers (except for *Netflix*) use certificates signed by public trust CAs. However, the majority send an incomplete chain and 5 servers provide expired certificates. This highlights the fact that some channels may not correctly perform certificate validation [34]. *Netflix* uses certificates signed by both public trust CAs and itself for different servers, which is consistent with our findings in Section 5.4.

**Servers managed by device vendors.** *Roku* exclusively signs all its certificates, consistent with our findings in Section 5.2. *Amazon* also signs its certificate. However, we find that one *Amazon* server[7] uses an expired certificate. We show details of certificate issuers and status in Table 17 in Appendix C.5.

**Vendor key infrastructure.** To further investigate the key infrastructure for servers managed by smart TV vendor *Amazon* and *Roku*, we divide traffic into two groups:

- *Amazon*: traffic between *Amazon* devices and *Amazon* servers.
- *Roku*: traffic between *Roku* devices and *Roku* servers.

We exclude connections to domain *amazonaws.com* and *amazonvideo.com* from the *Amazon* group because these two domains are visited by *Roku* devices as well.

Figure 7 shows the validity periods and CT log status for different issuers. *Amazon* servers have a simple key infrastructure where leaf certificates are issued by either *Amazon* or *DigiCert* with consistent certificate validity around 400 days. Additionally, all certificates used by *Amazon* servers are found logged in CT. On the contrary, *Roku* servers have a more complicated key infrastructure, involving certificates signed by a mixture of public trust CAs and private CAs, including *Amazon*, *DigiCert*, *Let's Encrypt*, and *Roku*, leading to a large variance in certificate validity period. In particular, most certificates signed by *Roku* itself have a much longer validity period, reaching around 5,000 days (over 13 years), and none are logged in CT. These findings align closely with our earlier observations on smart TV devices and vendors.

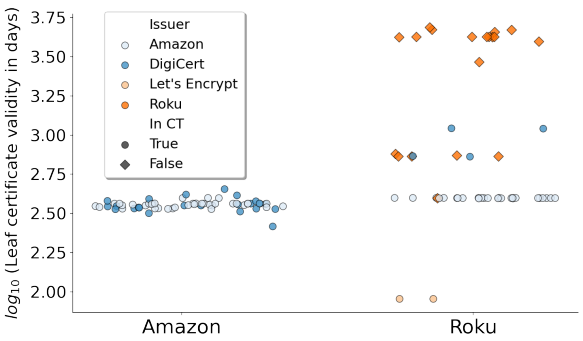


**Figure 7: Leaf certificates in *Amazon* and *Roku* groups.**

### 6.2 PKI on the Local Network

Many IoT devices communicate with one another on the local network over TLS. To study the PKI involved in such local communications, we capture the local network traffic of *Amazon* and *Google* devices, along with a phone and computer on the same network in the lab. Using Wireshark, we identify TLS packets and extract server certificates in TLS handshakes. We find that *Amazon* and *Google* devices communicate with each other and with other local devices over TLS with certificates issued by private CAs (*Amazon* devices), with root certificates not in either Android's or macOS' trust store (*Google* devices), and with certificates that would expire in 20+ years (*Google* devices).

**Method.** We set up a wireless network in the lab, where the following devices are connected: *Amazon* Echo and *Amazon* Fire TV; *Google* Chromecast and *Google* Home Voice Assistant; one *Google* Pixel 5 Android phone whose screen is locked; a MacBook with the Chrome browser open; and a Raspberry Pi that is capturing all the local network traffic with a modified version of IoT Inspector [23]. We capture the traffic for 24 hours, during which we do not actively interact with these devices.

**Observations for *Amazon* devices.** The *Amazon* Echo and Fire TV communicate with each other over TLS 1.2 on the local network. Echo listens on a non-IANA assigned port 55443, and Fire TV connects to Echo from random ports. During each TLS handshake, Echo presents what appears to be a single self-signed

[7] *arcus-uswest.amazon.com.*

certificate with no intermediate certificates. The Common Name on the certificate is the IP address of Echo. The certificate has an expiry date of one year in future from the issuance date.

**Observations for *Google* devices.** Chromecast and Home communicate with each other and also with the Pixel phone and MacBook over TLS. In particular, *Google* Home connects with Chromecast on port 10101 over TLS 1.2; Pixel connects with Chromecast on port 8443 over TLS 1.2; and MacBook connects with Chromecast on port 32245 over TLS 1.3. Additionally, Pixel also connects with *Google* Home on port 8443 over TLS 1.2.

We are able to extract server certificates in TLS 1.2 connections, as the certificates are encrypted in TLS 1.3 [41]. In each TLS handshake with *Google* Home and Pixel, Chromecast would present two certificates in the chain: a leaf certificate with what looks like a serial number in the Common Name, followed by a root certificate with a Common Name "Chromecast ICA 12", signed by "Cast Root CA", with 22 years of validity. Similarly, *Google* Home also presents two certificates in the chain: a leaf certificate with a potential serial number in the Common Name (different from Chromecast), followed by a root certificate with the Common Name "Chromecast ICA 16 (Audio Assist 4)", signed by, again, "Cast Root CA". The validity period is 20 years.

*Google* appears to be using certificate pinning or its own PKI for Chromecast and *Google* Home. However, we do not find any of its certificates in CT and we do not find any "Cast Root CA" in the trust stores of either the Pixel phone or macOS. These results complement our prior analysis IoT server certificates, showing concerning suboptimal server-side certificate practices in local network as well.

## 7 DISCUSSION

**Recommendations.** Our objective is to reveal the current TLS and certificate management practices, and motivate IoT device vendors towards adopting a proactive approach to security by consistently maintaining and improving the TLS instances integrated into their devices. This commitment to ongoing enhancement will serve as a fundamental pillar in strengthening overall security practices within the IoT industry.

In parallel, we strongly advocate for private CAs like device vendors to embrace automation frameworks to streamline the process of certificate management. By doing so, they can efficiently and effectively handle the issuance and renewal of certificates, ultimately bolstering security measures.

Furthermore, we envision a collaborative effort within our community to develop an auditing mechanism specifically tailored to accommodate certificates issued by private CAs. Such a mechanism would play a crucial role in enforcing compliance with relevant regulations and standards. It is our hope that this collaborative initiative will lead to greater transparency and accountability in the IoT ecosystem, ensuring the protection and trustworthiness of IoT devices and services.

**Limitations.** Our research is mainly limited by the lack of ground truth in IoT-deployed TLS instances and the unavailability of device firmware. Consequently, our fingerprinting results may exhibit biases associated with the extent of our collected TLS library fingerprints, and our examination of fingerprint sharing is restricted to what we have observed, inferred, and validated. Moreover, our analysis could further benefit from continuous data collection of a larger number of devices from IoT Inspector. While IoT Inspector provides us with the largest smart home IoT dataset known to date, the data collected from many devices is intermittent due to its crowdsourced nature, leading to potential bias in flavor of certain device vendors and device types (also discussed in [23]). If we can obtain continuous data collection from a given device, we would be able to perform additional measurements that delve deeper into the change of TLS behaviors potentially resulting from maintenance and updates during the device's life cycle, especially when considering the availability of release dates for IoT TLS instances on individual devices.

## 8 CONCLUSION

We have conducted a measurement study that provides a quantitative analysis on the extensive heterogeneity present in TLS instances used by IoT devices in real-world scenarios. Leveraging a crowdsourced dataset comprising network traffic data from 2,014 distinct IoT devices, our research unveils instances of vulnerable TLS configurations shared across various devices and device vendors, probably due to shared supply chains or third-party applications. Furthermore, we have identified multiple contributors to the configuration of IoT TLS and exposed shortcomings in certificate management practices employed by IoT servers, including the use of long-lasting or even long-expired certificates, the delivery of improperly configured certificate chains, and a notable absence of public auditing mechanism to ensure adherence to regulations and standards. These findings underscore significant security concerns associated with TLS and PKI practices in the IoT landscape, particularly when considering the sheer scale and diversity of IoT deployments. Our study serves as a call to action, aiming to raise awareness within the broader community about these critical security issues in the IoT ecosystem.

## ACKNOWLEDGMENTS

We thank our shepherd Mattijs Jonker and the anonymous IMC reviewers for their insightful and constructive suggestions and feedback. This work is supported by National Science Foundation grants CNS-2154962, CNS-2319421, CNS-2219867, CNS-1955227, and a Consumer Reports Digital Lab Fellowship.

## REFERENCES

[1] Josh Aas. 2015. Why ninety-day lifetimes for certificates? (2015). https://letsencrypt.org/2015/11/09/why-90-days.html.
[2] Josh Aas, Richard Barnes, Benton Case, Zakir Durumeric, Peter Eckersley, Alan Flores-López, J Alex Halderman, Jacob Hoffman-Andrews, James Kasten, Eric Rescorla, et al. 2019. Let's Encrypt: an automated certificate authority to encrypt the entire web. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. 2473–2487.
[3] Devdatta Akhawe, Johanna Amann, Matthias Vallentin, and Robin Sommer. 2013. Here's my cert, so trust me, maybe? Understanding TLS errors on the web. In *Proceedings of the 22nd international conference on World Wide Web*.
[4] Omar Alrawi, Chaz Lever, Manos Antonakakis, and Fabian Monrose. 2019. Sok: Security evaluation of home-based iot deployments. In *2019 IEEE symposium on security and privacy (sp)*. IEEE.
[5] Manos Antonakakis, Tim April, Michael Bailey, Matt Bernhard, Elie Bursztein, Jaime Cochran, Zakir Durumeric, J Alex Halderman, Luca Invernizzi, Michalis Kallitsis, et al. 2017. Understanding the mirai botnet. In *26th {USENIX} security symposium ({USENIX} Security 17)*. 1093–1110.
[6] Apple. 2023. Apple's Certificate Transparency policy. (2023). https://support.apple.com/en-us/HT205280.

[7] Apple. 2023. Available trusted root certificates for Apple operating systems. (2023). https://support.apple.com/en-us/HT209143.

[8] Chad Brubaker, Suman Jana, Baishakhi Ray, Sarfraz Khurshid, and Vitaly Shmatikov. 2014. Using frankencerts for automated adversarial testing of certificate validation in SSL/TLS implementations. In *2014 IEEE Symposium on Security and Privacy*. IEEE.

[9] Frank Cangialosi, Taejoong Chung, David Choffnes, Dave Levin, Bruce M Maggs, Alan Mislove, and Christo Wilson. 2016. Measurement and analysis of private key sharing in the https ecosystem. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. 628–640.

[10] Songqing Chen, Kim-Kwang Raymond Choo, Xinwen Fu, Wenjing Lou, and Aziz Mohaisen. 2019. *Security and Privacy in Communication Networks: 15th EAI International Conference, SecureComm 2019, Orlando, FL, USA, October 23–25, 2019, Proceedings, Part II*. Vol. 305. Springer Nature.

[11] T. Dierks and E. Rescorla. 2008. The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246 (Proposed Standard). (Aug. 2008). http://www.ietf.org/rfc/rfc5246.txt Updated by RFCs 5746, 5878, 6176.

[12] Daniel J Dubois, Roman Kolcun, Anna Maria Mandalari, Muhammad Talha Paracha, David Choffnes, and Hamed Haddadi. 2020. When speakers are all ears: Characterizing misactivations of iot smart speakers. *Proceedings on Privacy Enhancing Technologies* (2020).

[13] Zakir Durumeric, James Kasten, Michael Bailey, and J Alex Halderman. 2013. Analysis of the HTTPS certificate ecosystem. In *Proceedings of the 2013 conference on Internet measurement conference*.

[14] Zakir Durumeric, Zane Ma, Drew Springall, Richard Barnes, Nick Sullivan, Elie Bursztein, Michael Bailey, J Alex Halderman, and Vern Paxson. 2017. The Security Impact of HTTPS Interception.. In *NDSS*.

[15] Sam Edwards and Ioannis Profetis. 2016. Hajime: Analysis of a decentralized internet worm for IoT devices. *Rapidity Networks* 16 (2016), 1–18.

[16] Let's Encrypt. 2022. ACME Client Implementations. (2022). https://letsencrypt.org/docs/client-options/.

[17] Enphase Energy. 2023. Envoy 3.8.X. (2023). https://www4.enphase.com/en-us/legal/open-source-license-compliance-envoy-3.8.x.

[18] Trusted Firmware. 2023. Mbed TLS ChangeLog. (2023). Retrieved September 2023 from https://review.trustedfirmware.org/plugins/gitiles/mirror/mbed-tls/+/7c94d8bcab1ed7e7a0079c67aa41731243de6f54/ChangeLog

[19] Sergey Frolov and Eric Wustrow. 2019. The use of TLS in Censorship Circumvention.. In *NDSS*.

[20] Google. 2023. Certificate Lifetimes. (2023). https://chromium.googlesource.com/chromium/src/+/HEAD/net/docs/certificate_lifetimes.md.

[21] Google. 2023. Chromium.IsSecureTLSCipherSuite function. (2023). https://chromium.googlesource.com/chromium/src/net/+/master/ssl/.

[22] Ralph Holz, Lothar Braun, Nils Kammenhuber, and Georg Carle. 2011. The SSL landscape: a thorough analysis of the x. 509 PKI using active and passive measurements. In *Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference*.

[23] Danny Yuxing Huang, Noah Apthorpe, Frank Li, Gunes Acar, and Nick Feamster. 2020. Iot inspector: Crowdsourcing labeled network traffic from smart home devices at scale. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* (2020).

[24] Paul Jaccard. 1912. The distribution of the flora in the alpine zone. 1. *New phytologist* 11, 2 (1912), 37–50.

[25] Platon Kotzias, Abbas Razaghpanah, Johanna Amann, Kenneth G Paterson, Narseo Vallina-Rodriguez, and Juan Caballero. 2018. Coming of age: A longitudinal study of tls deployment. In *Proceedings of the Internet Measurement Conference 2018*.

[26] Lydia Kraus, Martin Ukrop, Vashek Matyas, and Tobias Fiebig. 2020. Evolution of SSL/TLS Indicators and Warnings in Web Browsers. In *Security Protocols XXVII: 27th International Workshop, Cambridge, UK, April 10–12, 2019, Revised Selected Papers 27*. Springer, 267–280.

[27] ABI Laboratory. 2023. API/ABI changes review for mbed TLS. (2023). Retrieved September 2023 from https://abi-laboratory.pro/index.php?view=timeline&l=mbedtls

[28] ABI Laboratory. 2023. API/ABI changes review for wolfSSL. (2023). Retrieved September 2023 from https://abi-laboratory.pro/?view=timeline&l=wolfssl

[29] Ben Laurie, Adam Langley, and Emilia Kasper. 2013. Certificate Transparency. RFC 6962. (June 2013). https://doi.org/10.17487/RFC6962

[30] Sectigo Limited. 2023. Crt.sh. (2023). Retrieved September 2023 from https://crt.sh/

[31] Yabing Liu, Will Tome, Liang Zhang, David Choffnes, Dave Levin, Bruce Maggs, Alan Mislove, Aaron Schulman, and Christo Wilson. 2015. An end-to-end measurement of certificate revocation in the web's PKI. In *Proceedings of the 2015 Internet Measurement Conference*.

[32] Zane Ma, James Austgen, Joshua Mason, Zakir Durumeric, and Michael Bailey. 2021. Tracing your roots: exploring the TLS trust anchor ecosystem. In *Proceedings of the 21st ACM Internet Measurement Conference*.

[33] Microsoft. 2023. Certificate Stores. (2023). https://docs.microsoft.com/en-us/windows-hardware/drivers/install/certificate-stores.

[34] Hooman Mohajeri Moghaddam, Gunes Acar, Ben Burgess, Arunesh Mathur, Danny Yuxing Huang, Nick Feamster, Edward W Felten, Prateek Mittal, and Arvind Narayanan. 2019. Watching you watch: The tracking ecosystem of over-the-top tv streaming devices. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*.

[35] Mozilla. 2023. Common CA Database. (2023). https://www.ccadb.org/.

[36] Mozilla. 2023. Mozilla's CA Certificate Program. (2023). https://wiki.mozilla.org/CA.

[37] Muhammad Talha Paracha, Daniel J Dubois, Narseo Vallina-Rodriguez, and David Choffnes. 2021. IoTLS: understanding TLS usage in consumer IoT devices. In *Proceedings of the 21st ACM Internet Measurement Conference*.

[38] The Zeek Project. 2020. Zeek. (2020). https://zeek.org/.

[39] Abbas Razaghpanah, Arian Akhavan Niaki, Narseo Vallina-Rodriguez, Srikanth Sundaresan, Johanna Amann, and Phillipa Gill. 2017. Studying TLS usage in Android apps. In *Proceedings of the 13th International Conference on emerging Networking EXperiments and Technologies*.

[40] Jingjing Ren, Daniel J Dubois, David Choffnes, Anna Maria Mandalari, Roman Kolcun, and Hamed Haddadi. 2019. Information exposure from consumer iot devices: A multidimensional, network-informed measurement approach. In *Proceedings of the Internet Measurement Conference*.

[41] Eric Rescorla. 2018. RFC 8446: The Transport Layer Security (TLS) Protocol Version 1.3. (2018). Retrieved September 2023 from https://datatracker.ietf.org/doc/html/rfc8446/

[42] Hans Christian Rudolph and Nils Grundmann. 2022. Ciphersuite Info. (2022). https://ciphersuite.info/.

[43] Paul Shorey. 1907. Emendation of Plato Charmides 168b. *Classical Philology* 2, 3 (1907), 340–340.

[44] Sabrina Sicari, Alessandra Rizzardi, Luigi Alfredo Grieco, and Alberto Coen-Porisini. 2015. Security, privacy and trust in Internet of Things: The road ahead. *Computer networks* 76 (2015), 146–164.

[45] Certificate Transparency. 2023. Google's Certificate Transparency project. (2023). https://certificate.transparency.dev/.

[46] Bhagyashri Tushir, Hetesh Sehgal, Rohan Nair, Behnam Dezfouli, and Yuhong Liu. 2021. The impact of dos attacks onresource-constrained iot devices: A study on the mirai attack. *arXiv preprint arXiv:2104.09041* (2021).

[47] Yingjie Wang, Guangquan Xu, Xing Liu, Weixuan Mao, Chengxiang Si, Witold Pedrycz, and Wei Wang. 2020. Identifying vulnerabilities of SSL/TLS certificate verification in Android apps with static and dynamic analysis. *Journal of Systems and Software* 167 (2020), 110609.

[48] WIKIPEDIA. 2023. OpenSSL. (2023). Retrieved September 2023 from https://en.wikipedia.org/wiki/OpenSSL

[49] Ben Wilson. 2020. Reducing TLS Certificate Lifespans to 398 Days. (2020). https://blog.mozilla.org/security/2020/07/09/reducing-tls-certificate-lifespans-to-398-days/.

[50] wolfSSL. 2023. wolfSSL change log. (2023). Retrieved September 2023 from https://github.com/wolfSSL/wolfssl/blob/master/ChangeLog.md

[51] Wyze. 2021. Open Source Software. (2021). https://support.wyze.com/hc/en-us/articles/360012546832-Open-Source-Software.

[52] Liang Zhu, Johanna Amann, and John Heidemann. 2016. Measuring the latency and pervasiveness of TLS certificate revocation. In *International Conference on Passive and Active Network Measurement*. Springer.

[53] Serkan Özkan. 2023. OpenSSL verion 1.0.0: Security vulnerabilities. (2023). Retrieved September 2023 from https://www.cvedetails.com/vulnerability-list/vendor_id-217/product_id-383/version_id-453965/Openssl-Openssl-1.0.0.html/


# A ETHICS

The authors' use of the IoT Inspector dataset has gained the appropriate permission from the original authors' IRB.

# B IOT CLIENT-SIDE TLS ANALYSIS

## B.1 Fingerprints Coverage

While our compiled TLS libraries may not encompass all possible scenarios, it is worth noting that our collection of standard TLS libraries is similar in scope to that of [37], as it includes the major libraries commonly employed by IoT devices. We show the coverage of our compiled library fingerprints as follows:

- 19 OpenSSL versions:
  - 1.0.0m, 1.0.0q, 1.0.0t
  - 1.0.1h, 1.0.1l, 1.0.1r, 1.0.1u
  - 1.0.2, 1.0.2f, 1.0.2-beta1, 1.0.2-beta2, 1.0.2m, 1.0.2u

  - 1.1.0l, 1.1.0-pre1, 1.1.0-pre2, 1.1.0-pre3
  - 1.1.1i, 1.1.1-pre2
- 38 wolfSSL versions:
  - 1.8.0
  - 2.1.1, 2.2.1, 2.2.2, 2.3.0, 2.4.6, 2.4.7, 2.5.0, 2.5.2, 2.5.2b, 2.6.0, 2.8.0, 2.9.0
  - 3.0.0, 3.0.2, 3.1.0, 3.4.0, 3.4.2, 3.4.8, 3.6.0, 3.7.0, 3.8.0, 3.9.0, 3.9.10-stable, 3.10.2-stable, 3.10.3, 3.11.0-stable, 3.12.0-stable, 3.13.0-stable, 3.14.2, 3.14.5, 3.15.0-stable, 3.15.3-stable, 3.15.6, 3.15.7-stable,
  - 4.0.0-stable
  - WCv4.0-RC4, WCv4.0-RC5
- 113 Mbed TLS versions:
  - PolarSSL 0.13.1, 0.14.0, 0.14.2, 0.14,3
  - PolarSSL 1.0.0, 1.1.0, 1.1.1, 1.1.2, 1.1.3, 1.1.4, 1.1.5, 1.1.6, 1.1.7, 1.1.8
  - PolarSSL 1.2.0, 1.2.1, 1.2.2, 1.2.3, 1.2.4, 1.2.5, 1.2.6, 1.2.7, 1.2.8, 1.2.9, 1.2.10, 1.2.11, 1.2.12, 1.2.13, 1.2.14, 1.2.15, 1.2.16, 1.2.17, 1.2.18, 1.2.19
  - 1.3.0, 1.3.1, 1.3.2, 1.3.3, 1.3.4, 1.3.5, 1.3.6, 1.3.7, 1.3.8, 1.3.9
  - Mbed TLS 1.3.10, 1.3.11, 1.3.12, 1.3.13, 1.3.14, 1.3.15, 1.3.16, 1.3.17, 1.3.18, 1.3.19, 1.3.20, 1.3.21, 1.3.22
  - Mbed TLS 1.4-dtls-preview
  - 2.1.0, 2.1.1, 2.1.2, 2.1.3, 2.1.4, 2.1.5, 2.1.6, 2.1.7, 2.1.8, 2.1.9, 2.1.10, 2.1.11, 2.1.12, 2.1.13, 2.1.14, 2.1.15, 2.1.16, 2.1.17, 2.1.18
  - 2.2.0, 2.2.1
  - 2.3.0
  - 2.4.0, 2.4.2
  - 2.5.1
  - 2.6.0
  - 2.7.0, 2.7.2, 2.7.3, 2.7.4, 2.7.5, 2.7.6, 2.7.7, 2.7.8, 2.7.9, 2.7.10, 2.7.11, 2.7.12, 2.7.13, 2.7.14, 2.7.15
  - 2.8.0
  - 2.9.0
  - 2.11.0
  - 2.12.0
  - 2.13.0
  - 2.14.0, 2.14.1
  - 2.16.0, 2.16.2, 2.16.2, 2.16.3, 2.16.4, 2.16.5, 2.16.6
- 5,591 versions of different curl versions (from curl 7.19.010 to curl 7.71.09) compiled with different OpenSSL versions
- 1,130 versions of different curl versions (from curl 7.25.0 to curl 7.68.0) compiled with different wolfSSL versions

Specifically, we document the release dates of significant versions of the aforementioned libraries in Table 10 [18, 27, 28, 48, 50]. We note that our ClientHello dataset is collected from April 29, 2019 to August 1, 2020. Our matching findings indicate that 23 fingerprints of IoT Inspector ClientHellos align precisely with 16 established TLS libraries. Among these, 14 out of 16 are no longer supported as of 2020, including severely outdated versions like OpenSSL 1.0.0q, 1.0.0t, 1.0.1h, 1.0.1l, 1.0.1r, 1.0.1u, 1.0.2f, 1.0.2u, wolfSSL 2.9.0.

## B.2 Semantics-aware TLS Fingerprinting

Client-side TLS fingerprinting help reveal vulnerabilities in device configuration. Even without exact match, proposing ciphersuites with the same or similar components as known libraries enables the sharing of similar risks. In order to draw comparison across all device fingerprints, we need to expand the TLS library matching in Section 4.1 to significantly more devices. Thus, we develop a new *semantics-aware* TLS fingerprinting approach that identifies the *likely* known library based on the specific algorithms in the proposed ciphersuites. We use the following criteria and assign categories corresponding to the degree of similarity:

- *Exact match:* the proposed ciphersuite list is exactly the same as a known TLS library. This is slightly different from the exact matching in Section 4.1, where here it may include device fingerprints that have the same ciphersuite list but not the same extensions or TLS version.
- *Same set different ordering:* the proposed ciphersuite list contains the same ciphersuites as a known TLS library, but they appear in different orderings in the list.
- *Same component:* the proposed ciphersuites contain the same sets of components as a TLS library, i.e., key exchange and authentication algorithm set, encryption algorithm set, and MAC set, but have different combinations of these algorithms, leading to different ciphersuites.
- *Similar component:* the proposed ciphersuites contain *similar* (albeit not the same) sets of components as a TLS library. We define two algorithms to be similar to each other when they only differ in key length and provide the same level of security[8]. E.g., cipher *AES_128_CBC* and *AES_256_CBC*, and MAC *SHA256* and *SHA384*, are two pairs of similar algorithms. But MAC *SHA-1* and *SHA256* are *not* similar.

[8]We exclude key exchange and authentication algorithm as there is no key length involved.

**Table 10: Release date of major library versions.**

| Library | Version | Release date | Last minor version |
|---|---|---|---|
| *OpenSSL* | 1.0.0 | 29 March 2010 | 1.0.0t (3 December 2015) |
| | 1.0.1 | 14 March 2012 | 1.0.1u (22 September 2016) |
| | 1.0.2 | 22 January 2015 | 1.0.2u (20 December 2019) |
| | 1.1.0 | 25 August 2016 | 1.1.0l (10 September 2019) |
| | 1.1.1 LTS | 11 September 2018 | 1.1.1w (11 September 2023) |
| *wolfSSL* | 1.8.0 | 23 December 2010 | - |
| | 2.1.1 | 25 May 2012 | - |
| | 3.0.0 | 29 April 2014 | - |
| | 3.4.0 | 23 February 2015 | - |
| | 3.10.0 | 21 December 2016 | - |
| | 3.12.0 | 4 August 2017 | - |
| | 3.14.0 | 2 March 2018 | - |
| | 3.15.0 | 5 June 2018 | - |
| | 4.0.0 | 20 March 2019 | - |
| *Mbed TLS* | 1.0.0 | 27 July 2011 | - |
| | 1.2.0 | 31 October 2012 | - |
| | 1.3.0 | 1 October 2013 | - |
| | 2.1.0 | 4 September 2015 | - |
| | 2.2.0 | 4 November 2015 | - |
| | 2.3.0 | 27 June 2016 | - |
| | 2.6.0 | 10 August 2017 | - |
| | 2.7.0 | 5 February 2018 | - |
| | 2.8.0 | 16 March 2018 | - |
| | 2.16.0 | 21 December 2018 | - |
| | 2.16.4 | 15 January 2020 | - |

**Table 11: Semantics-aware fingerprinting results.**

| Category | %Total | #.Vendors | %Outdated* |
|---|---|---|---|
| *Exact same* | 10.69% | 34 | 99.20% |
| *Same set diff order* | 0.46% | 10 | 81.48% |
| *Same component* | 6.42% | 17 | 97.59% |
| *Similar component* | 35.80% | 56 | 99.66% |
| *Customization* | 46.63% | 51 | 71.99% |

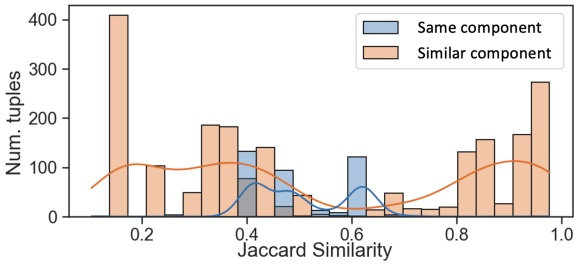


**Figure 8: Jaccard Similarity of client-proposed ciphersuite lists and the most likely libraries for category *Same component* and *Similar component*. Y-axis shows the number of each unique *{device, ciphersuite list}* tuple.**

- *Customization:* the proposed ciphersuites are further customized by device vendors.

**Semantics-aware fingerprinting results.** We show fingerprinting results of 5,827 unique {device, ciphersuite list} tuples in Table 11, where each value represents the number of such tuples. We are able to identify the closest library for over half of the device fingerprints. This result suggests that most devices use customized TLS configurations that cannot be matched against standard libraries.

Most of the devices are matched to libraries that are no longer supported as of 2020, with more than half using legacy library versions whose last update was prior to 2017, resulting in many known vulnerabilities. It is worth mentioning that we observed combinations of up-to-date curl versions and outdated TLS library versions such as *curl 7.71.018 OpenSSL 1.0.0t* and *curl 7.65.1 wolfSSL 3.7.1*.

**Table 12: TLS version proposed by IoT devices.**

| TLS version | TLS 1.2 | TLS 1.1 | TLS 1.0 | SSL 3.0 |
|---|---|---|---|---|
| **#. Proposals** | 5214 | 18 | 236 | 31 |

We next dig further into ciphersuite overlaps of category *same component* and *similar component* by measuring Jaccard Similarity of device-proposed ciphersuites and the most likely library. We show the distribution in Figure 8. Comparing to *same component*, *similar component* distributes more towards both end. The higher-end distribution suggests strong possibilities of many devices making changes based on specific libraries. This includes *Roku* and *Amazon* devices. On the contrary, though devices at the lower-end show a much higher degree of customization, they still share similar risks as known libraries because of the usage of similar algorithm components.

**Limitation.** Our method of matching fingerprints helps us infer the *likely* TLS libraries and versions used. Absent the ground truth at scale (e.g., vendors' disclosures), our results are not definitive, although they provide directional values to researchers. Given the large variety of devices and their black-box nature, our analysis can potentially point researchers to relevant directions and suggest possible IoT device candidates (e.g., those with outdated OpenSSL versions) for further in-lab analysis (which is often time-consuming).

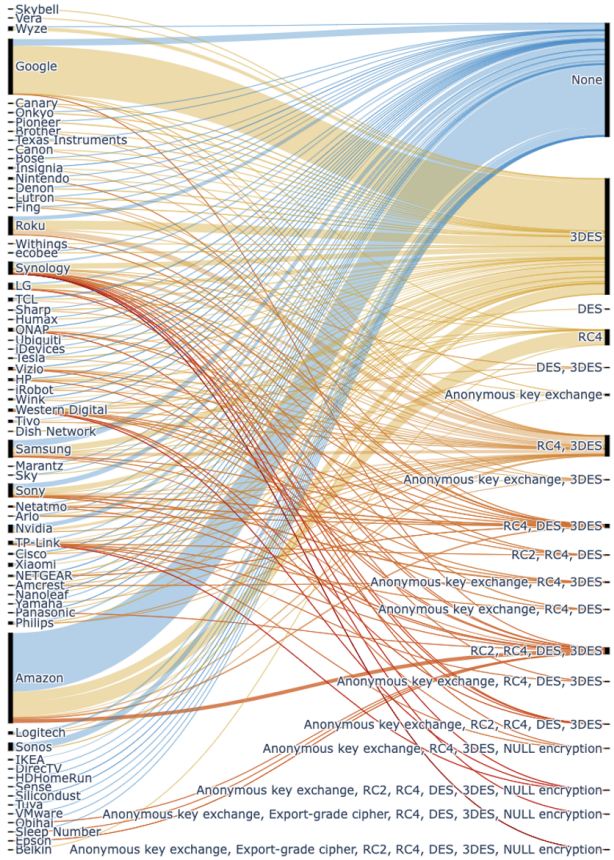


**Figure 9: Inclusion of vulnerable ciphersuite components. Each flow unit shows a unique {*device, ciphersuite list*} tuple.**

## B.3 TLS Parameters

*B.3.1 Ciphersuites.* Besides unfolding vulnerable ciphersuite algorithms used by IoT devices, we perform additional analysis on client-proposed ciphersuites in terms of specific ciphersuite inclusion and algorithm ordering.

**TLS_FALLBACK_SCSV ciphersuite.** The presence of such ciphersuite, though does not reveal any weakness at the device side, indicates the possibility of a downgrade attack. We observed that 20 devices from 6 vendors, including *Amazon*, support such ciphersuite.

**Other metrics.** We developed other metrics to evaluate client-proposed ciphersuites such as lowest index of vulnerable ciphersuites, and most preferred algorithms. We show details in Appendix B.7 - B.8.

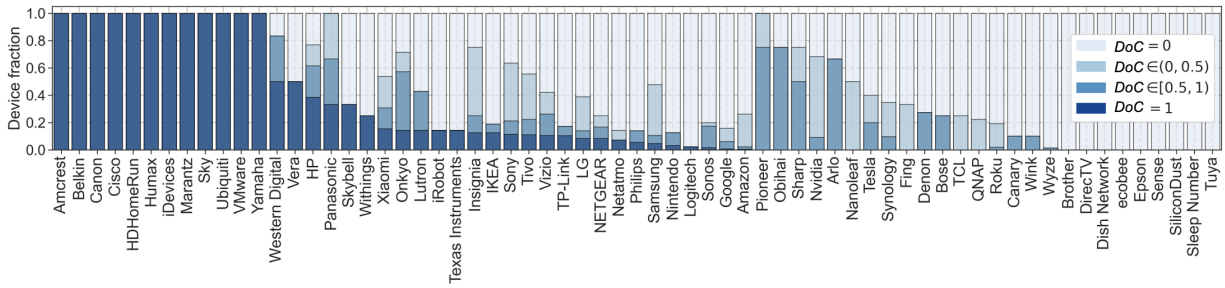


**Figure 10: Degree of customization across devices for vendors.**

**Table 13: Index and vendor mapping in Figure 1.**

| Index | Vendor | Index | Vendor | Index | Vendor | Index | Vendor | Index | Vendor |
|---|---|---|---|---|---|---|---|---|---|
| 1 | Roku | 2 | TCL | 3 | Samsung | 4 | Sharp | 5 | Insignia |
| 6 | Amazon | 7 | Nvidia | 8 | Google | 9 | HP | 10 | Western Digital |
| 11 | Xiaomi | 12 | Sony | 13 | Lutron | 14 | iDevices | 15 | TP-Link |
| 16 | Vizio | 17 | Pioneer | 18 | Onkyo | 19 | wink | 20 | LG |
| 21 | Cisco | 22 | Philips | 23 | Synology | 24 | TiVo | 25 | Wyze |
| 26 | Sonos | 27 | Amcrest | 28 | Panasonic | 29 | QNAP | 30 | Fing |
| 31 | Brother | 32 | Dish Network | 33 | Skybell | 34 | NETGEAR | 35 | Arlo |
| 36 | iRobot | 37 | Yamaha | 38 | Texas Instruments | 39 | Tesla | 40 | Bose |
| 41 | Sky | 42 | Humax | 43 | Ubiquity | 44 | Logitech | 45 | Netatmo |
| 46 | SiliconDust | 47 | HDHomeRun | 48 | Sense | 49 | DirecTV | 50 | Denon |
| 51 | Marantz | 52 | Nanoleaf | 53 | VMware | 54 | Obihai | 55 | Canary |
| 56 | ecobee | 57 | Epson | 58 | IKEA | 59 | Belkin | 60 | Nintendo |
| 61 | Sleep number | 62 | Tuya | 63 | Canon | 64 | Vera | 65 | Withings |

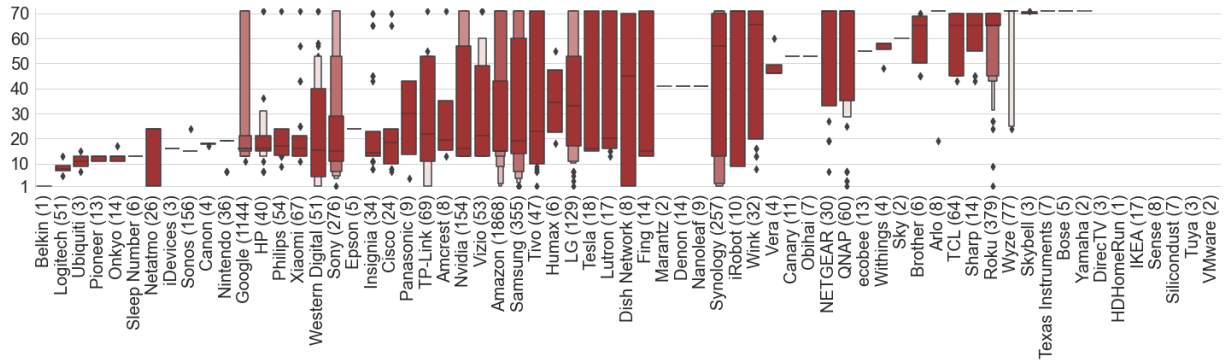


**Figure 11: The distribution of the lowest index of vulnerable ciphersuites. Without or with higher y-values generally indicate more secure practices. The x-axis is sorted by the mean index of vulnerable ciphersuites in an ascending order, from left to right. The y-axis shows the distribution of indices of vulnerable ciphersuites. The values in parentheses on the x-axis show numbers of each unique *{device, ciphersuite list}* tuple per vendor.**

*B.3.2 TLS versions.* Table 12 shows the number of connections of each TLS version, excluding those that cannot be parsed. We do not observe any TLS 1.3 connection, even though TLS 1.3 was released 8 months prior to our data capturing. 194 (out of 2,014) devices propose more than one TLS version during our 15-month capturing period. However, we do not observe any trend in proposed TLS versions over the capturing period. Alarmingly, 26 devices[9] still propose SSL 3.0 for 31 times even though it was deprecated in 2015. This may be due to two reasons: (i) the vendor may not have installed the latest TLS library on the device when manufacturing it; (2) the device never updates TLS library after being manufactured, which is the more likely reason.

*B.3.3 TLS extensions.* We observe some devices have the same ciphersuite lists as known TLS libraries, but do not have the same fingerprints due to different extensions. After taking a further look, we find that the differences mainly attribute to the inclusion of application-specific extensions[10] and the extension *padding*. Extension *session_ticket* and *renegotiation_info* are much more frequently included by IoT devices compared to known libraries. We show further details of this analysis, including other extensions such as

[9] *Amazon* (13), *Synology* (5), *Samsung* (4), *LG* (2), *TP-Link* (1), and *Western Digital* (1).

[10] *application_layer_protocol_negotiation* and *next_protocol_negotiation*.

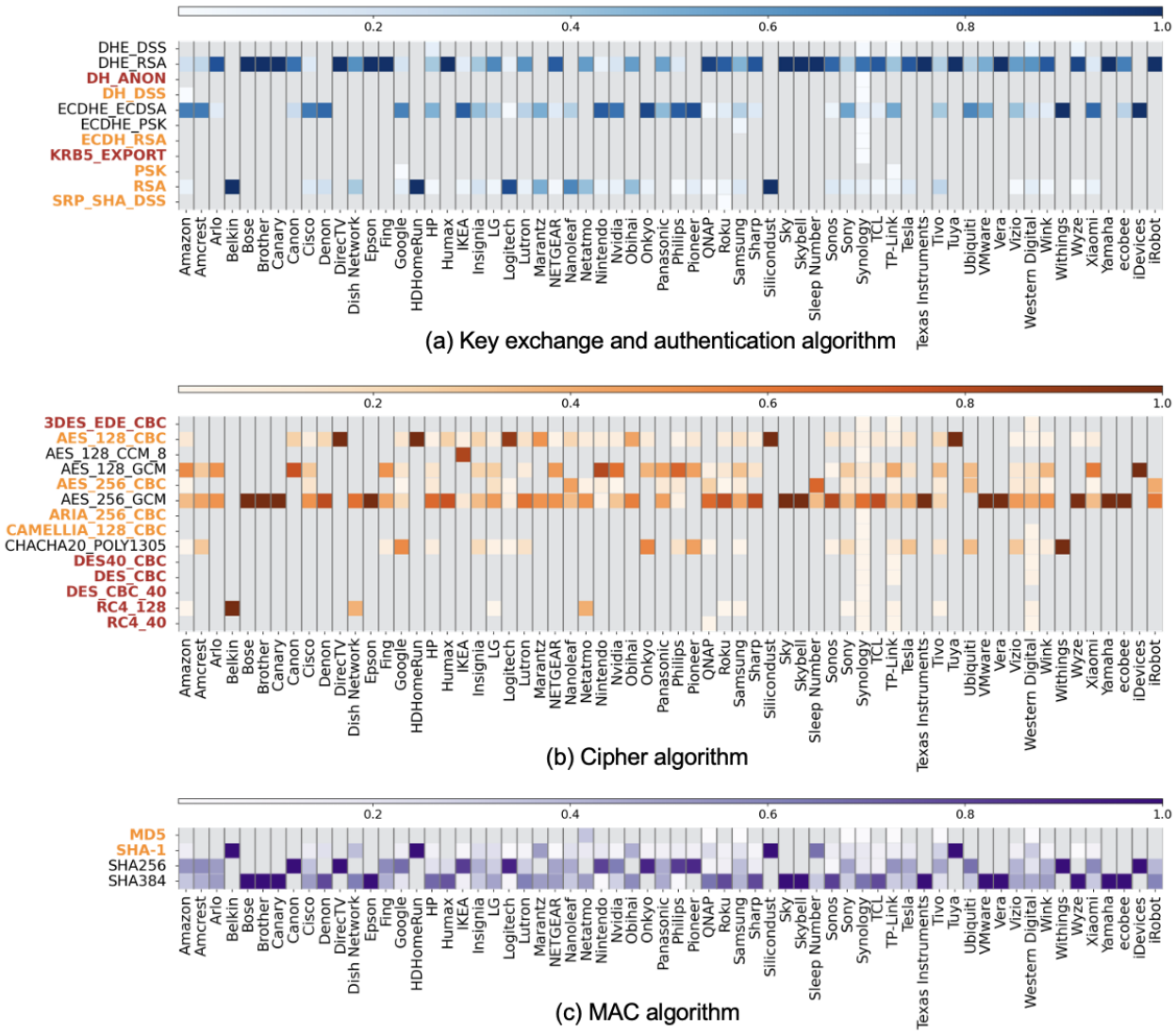


**Figure 12: Component algorithms of ciphersuite proposed by client devices in the first place of each ciphersuite list. Squares of value 0 (*i.e.*, no usage) are grayed out. Algorithms in color orange are considered suboptimal, and those in color red are vulnerable.**

OCSP request and GREASE, in Appendix B.9 and B.10.

## B.4 Proposal of Vulnerabilities

Figure 9 shows details on the proposal of vulnerabilities in ciphersuites on a vendor basis where each flow unit represents a unique {*device*, *ciphersuite list*} tuple.

## B.5 Degree of Customization Across Devices

Figure 10 demonstrates the *DoC* distribution for all 65 vendors where a darker color indicates a higher degree of customization.

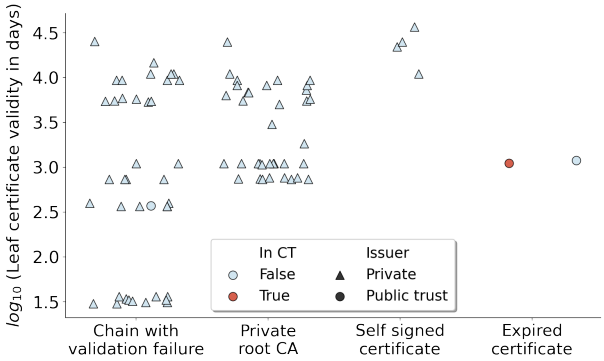


**Figure 13: Leaf certificates in invalid chains.**

**Table 14: Certificate chains with private issuers.**

| Certificate chain validation | Domain | #. FQDN(s) | Leaf certificate issued by | Chain length | Visited by #.devices | Visited by devices of |
|---|---|---|---|---|---|---|
| Private root CA | *roku.com* | 15 | Roku | 2, 3 | 102 | Brother, Cisco, Insignia, Roku, Sharp, TCL |
| | *rokutime.com* | 1 | Roku | 2 | 38 | Brother, Insignia, Roku, TCL |
| | *nintendo.net* | 14 | Nintendo | 2 | 28 | Nintendo |
| | *playstation.net* | 11 | Sony Computer Entertainment | 2 | 17 | Sony |
| | *netflix.com* | 4 | Netflix | 2 | 15 | Amazon, LG, Panasonic, Roku, Samsung, Sony, Tivo |
| | *lgtvsdp.com* | 2 | LG Electronics | 2 | 12 | LG |
| | *canaryis.com* | 2 | Canary Connect | 4 | 10 | Canary |
| | *netflix.net* | 1 | Netflix | 2 | 9 | Roku, Samsung |
| | *samsungelectronics.com* | 1 | Samsung Electronics | 4 | 8 | Samsung |
| | *pavv.co.kr* | 1 | Samsung Electronics | 2 | 7 | Netatmo, Samsung |
| | *sense.com* | 2 | Sense Labs | 3 | 6 | Sense, Texas Instruments |
| | *sonyentertainmentnetwork.com* | 1 | Sony Computer Entertainment | 2 | 5 | Sony |
| | *ecobee.com* | 1 | ecobee | 3 | 4 | ecobee |
| | *dtvce.com* | 1 | ATT Mobility and Entertainment | 4 | 3 | DirecTV, Humax, Samsung |
| | *tesla.services* | 1 | Tesla Motor Services | 3 | 3 | LG, Tesla |
| Self-signed certificate | *ueiwsp.com* | 1 | Samsung Electronics | 1 | 8 | Samsung |
| | *dishaccess.tv* | 2 | EchoStar | 1 | 3 | Dish Network |
| | *samsunghrm.com* | 1 | Samsung Electronics | 2 | 3 | Samsung |
| | *tuyaus.com* | 1 | Tuya | 1 | 3 | Tuya |

## B.6 Vendor and Fingerprint Mapping

We show the mapping between device vendor and vendor index in Table 13.

## B.7 Lowest Index of Vulnerable Ciphersuites

The order of the ciphersuites proposed by the client has significant security implications, as many servers default to choose the first supported ciphersuite from the list. For example, if the client proposes *(vulnerable_ciphersuite, optimal_ciphersuite)*, and the server also supports the vulnerable ciphersuite, then the vulnerable ciphersuite will likely be chosen despite the presence of a secure ciphersuite.

Thus, we performed analysis on the lowest index of ciphersuites proposed by clients containing vulnerable algorithms (defined in 4.2). The results are shown in Figure 11. We find that at least one device from 13 vendors proposes a vulnerable ciphersuite as the first (i.e., most preferred) ciphersuite. Devices of 7 vendors never contain any vulnerable ciphersuite.

## B.8 Ciphersuites: Most Preferred Algorithms

We pick the first ciphersuite in each client-proposed list, and divide it into three components: the key exchange and authentication algorithm, the cipher algorithm, and the MAC (Message Authentication Code) algorithm. We show the ratio of usage of each component algorithm that is ever proposed in the first place on a device vendor basis in Figure 12. Note that the number of device vendors may be less than the total number of vendors in *IoT Inspector Dataset* as devices that put renegotiation information[11] in the first place of its preferred list are excluded.

Considerable amount of devices prefer a ciphersuite consists of suboptimal or even vulnerable algorithms most. Some devices of vendor *Synology* put a ciphersuite containing vulnerable algorithms in at least one algorithm component, and *Synology* is the only vendor with devices proposing a ciphersuite including vulnerable DH_ANON and KRB5_EXPORT key exchange and authentication algorithms as the most preferred ciphersuite. **All** *Belkin* devices propose a ciphersuite containing the vulnerable RC4_128 cipher algorithm in the first place. Other devices including an insecure cipher algorithm are with 12 vendor[12]. Devices from several vendors still prefer MD5 as the MAC algorithm[13].

## B.9 Extension: OCSP Request

OCSP (Online Certificate Status Protocol) stapling is a standard that allows the presenter of a certificate to staple an OCSP response signed by the CA during TLS handshake. In TLS connections, a client can include the extension *status_request* in the ClientHello to request for an OCSP response from the server. In *IoT Inspector Dataset*, we find that 648 out of 2,014 devices include this extension in at least one connection. These devices are from 33 (out of 65) vendors.

[11] *TLS_EMPTY_RENEGOTIATION_INFO_SCSV.*
[12] *Synology, TP-Link, Western Digital, Amazon, Dish Network, LG, Netatmo, Roku, Samsung, Sony, Tivo,* and *QNAP.*
[13] *Netatmo, QNAP, Samsung, Sony, Synology, TP-Link, Tivo,* and *Western Digital.*

### B.10 GREASE

GREASE (Generate Random Extensions And Sustain Extensibility), according to RFC 8701, is a mechanism to prevent extensibility failures in the TLS ecosystem by reserving a set of TLS protocol values that may be advertised to ensure peers correctly handle unknown values.

**Ciphersuite GREASE.** 501 out of 2,014 distinct devices use GREASE in the supported ciphersuite lists. These 501 devices belong to 23 device vendors.

**Extension GREASE.** 503 out of 2,014 devices use GREASE in extensions. These 503 devices belong to 15 device vendors. In particular, we note that 2 devices are observed to include GREASE only in extensions rather than ciphersuites, of vendor *Google* and *Amazon*, respectively.

## C SERVER PKI INFRASTRUCTURE

### C.1 SLDs of IoT Servers

Table 15 summarizes the most popular 30 SLDs of 1,151 IoT servers, sorted by their popularity among IoT devices.

**Table 15: Popular SLDs of 1,151 IoT servers.**

| SLD | #. Servers (FQDNs) | Contacted by #. unique devices |
|---|---|---|
| *amazon.com* | 57 | 556 |
| *google.com* | 24 | 499 |
| *googleapis.com* | 35 | 420 |
| *amazonalexa.com* | 2 | 337 |
| *gstatic.com* | 10 | 328 |
| *netflix.com* | 30 | 327 |
| *amazonaws.com* | 33 | 250 |
| *doubleclick.net* | 9 | 232 |
| *youtube.com* | 2 | 217 |
| *cloudfront.net* | 21 | 150 |
| *googleusercontent.com* | 6 | 146 |
| *roku.com* | 42 | 135 |
| *nflxext.com* | 2 | 125 |
| *sonos.com* | 10 | 124 |
| *scdn.co* | 11 | 124 |
| *spotify.com* | 8 | 117 |
| *facebook.com* | 9 | 112 |
| *googlesyndication.com* | 3 | 105 |
| *amazonvideo.com* | 23 | 101 |
| *ggpht.com* | 5 | 99 |
| *ytimg.com* | 4 | 94 |
| *media-amazon.com* | 1 | 93 |
| *amazon-dss.com* | 1 | 90 |
| *meethue.com* | 3 | 84 |
| *amcs-tachyon.com* | 1 | 82 |
| *sentry-cdn.com* | 1 | 75 |
| *ssl-images-amazon.com* | 1 | 70 |
| *plex.tv* | 11 | 69 |
| *nest.com* | 4 | 68 |
| *google-analytics.com* | 2 | 63 |

**Table 16: Certificates usage across geographical locations.**

| | New York | Frankfurt | Singapore |
|---|---|---|---|
| #.SNIs with certificate successfully extracted | 1151 | 1149 | 1150 |
| #.SNIs with certificate ***shared across all places*** | | **1087** | |
| #.SNIs with certificate ***exclusive in this location*** | 106 | 99 | 82 |

### C.2 Certificate Chains with Private Issuers

Table 14 shows details on domains that present a certificate chain with status "untrusted root CA" or "self-signed certificate". Certificate chains with status "self-signed certificate" usually just have one certificate in chains, leading to difficulties in chain validation.

### C.3 CT and Certificate Chains with Private Issuers

We show the correlation between leaf certificates in chains signed by private issuers and their corresponding CT existence in Figure 13, with the majority of leaf certificates in such chains *not* logged in CT. We observe usage of 2 expired leaf certificates that are issued by public trust issuers, including 1 by Sectigo (*not* logged in CT) and 1 by Gandi, suggesting that besides authenticated CAs, server owners should also prioritize monitoring their TLS certificates even if issued by public trust issuers.

**Table 17: Servers presenting an invalid or misconfigured certificate chain.**

| Chain validation | Domain(s) visited by *Amazon* device | Domain(s) visited by *Roku* device |
|---|---|---|
| **Incomplete chain** | *netflix.com* (5)<br>*playstation.net* (2)<br>*tremorvideo.com* (1)<br>*hsn.com* (1) | *netflix.com* (12)<br>*roku.com*[1] (6)<br>*vvond.net* (2)<br>*tremorvideo.com* (1)<br>*cymtv.com* (1)<br>*rhythmxchange.com* (1)<br>*rubiconproject.com* (1)<br>*contextweb.com* (1)<br>*sonyentertainmentnetwork.com* (1)<br>*otherworlds.tv* (1)<br>*spotxchange.com* (1) |
| **Untrusted root CA** | *roku.com*[1] (2) | *roku.com*[1] (12)<br>*netflix.com* (1)<br>*rokutime.com*[1] (1) |
| **Expired certificate** | *amazon.com*[2] (1)<br>*clikia.com* (1) | *altitude-arena.com* (1)<br>*saddleback.com* (1)<br>*smartott.com* (1)<br>*yumenetworks.com* (1) |

[1] Domains run by *Roku*.
[2] Domains run by *Amazon*.

### C.4 Dataset Validation

As mentioned in Section 5.1, we address several concerns on the accuracy of our certificate dataset by comparing across multiple geographical locations and cross checking with a lab dataset.

*C.4.1 Comparison across geographical locations.* We first investigate the impact of geographical locations of TLS client on IoT servers' choices of leaf certificates. We compared certificates

obtained using TLS clients located in New York (U.S.), Frankfurt (Europe), and Singapore (Asia). Table 16 shows the number of identical certificates across the three locations.

Though a small amount of servers use exclusive certificates for specific places, including 106 servers in New York, 99 and 82 in Frankfurt and Singapore respectively, we observe 1,087 SNIs with certificates shared by all geographical locations, indicating an overall consistency in certificate usage across places. Given that the difference is negligible, we use certificates captured at New York for our analysis for simplicity.

*C.4.2 Comparison with Lab Dataset.* The *Lab Dataset*, as described in Section 5.1, contains network traffic captured in the lab of 113 IoT devices belonging to 52 vendors from 2017 to 2021. Although it has far fewer devices than the IoT Inspector dataset, its certificate data is captured directly from the devices, serving as a supplemental data source for cross checking.

17 device vendors are found in both datasets, covering 88.61% of all connections in the lab dataset and 81.52% of connections in our certificate dataset . Among them, 362 SNIs are visited by devices in common in both datasets. In particular, 356 (out of 362 SNIs that are in common in both datasets) present certificates issued by the same issuer organization in both datasets, covering 13 device vendors. These certificates show overall consistency in certificate validity period and CT appearance. The remaining 7 SNIs provide certificates issued by different issuers, but are largely consistent in CT appearance. In summary, certificates across both datasets exhibit similar properties.

We also notice growing trend of CT logging from the same public trust CAs in our 2022 certificate dataset compared to the lab dataset. This is consistent with the increasing deployment of CT in recent years. Thus, we argue that our certificate dataset provides an up-to-date view on IoT server certificates as of early 2022.

## C.5 Case Study: Smart TVs

Table 17 shows domains with invalid or misconfigured certificate chains visited by *Amazon* and *Roku* smart TVs. Note that values in brackets suggest number of FQDN(s) for each domain. Servers sending incomplete chains are with certificates issued by public trust issuers, except 5 from *netflix.com* (signed by *Netflix*) and 6 from *roku.com* (with issuer *Roku*). Servers with the issue "untrusted root CA" use certificates signed by private issuer *Roku* and *Netflix*, while all expired certificates are found issued by public trust issuers, which is consistent with our previous observation in Section 5.3.