BRNO UNIVERSITY OF TECHNOLOGY FACULTY OF INFORMATION TECHNOLOGY

Excel@FIT 2021
http://excel.fit.vutbr.cz

# JA3cury - A new approch to TLS fingerprinting by merging fingerprinting methods

Lukáš Hejcman[1], Ing. Karel Hynek[2], Ing. Tomáš Čejka Ph.D.[3]

**Abstract**
TLS is the most popular encryption protocol used on the internet today. It aims to provide high levels of security and privacy for inter-device communication. However, it presents a challenge from a network monitoring and administration standpoint, as it is not possible to analyse the communication encrypted with TLS at a large scale with existing methods based on deep packet inspection. Analysing encrypted communication can help administrators to detect malicious activity on their networks, and can help them identify potential security threats. In this paper, we present a method that allows us to leverage the advantages of two TLS fingerprinting methods, JA3 and Cisco Mercury, to determine the operating system and processes of clients on multiple networks. Our method is able to achieve comparable or better results than the existing Mercury approach for our datasets whilst providing more analysis opportunities than JA3. Furthermore, by using JA3 fingerprints, we open the door to the utilisation of this approach in the wider industry, where JA3 fingerprinting is predominant.

**Keywords:** TLS – Fingerprint – Cisco – Mercury – JA3 – Identification – JA3cury

**Supplementary Material:** *N/A*

[1]xhejcm01@stud.fit.vutbr.cz, *Faculty of Information Technology, Brno University of Technology*
[2]hynekkar@fit.cvut.cz, *Faculty of Information Technology, Czech University of Technology*
[3]cejkat@cesnet.cz, *CESNET, z.s.p.o.*

## 1. Introduction

Network traffic analysis is the process of capturing and analysing network traffic to increase the network's performance and security. Whilst many methods exist to accomplish this, it is becoming more important to utilize automatic techniques to filter the content on the network into specific categories due to the constantly increasing amount of traffic passing through networks.

However, network traffic analysis is becoming more challenging due to the rise in the use of encrypted traffic. According to a report by Google LLC[1], the amount of encrypted network traffic using the TLS protocol has been steadily increasing since at least 2014 to the current 95% of all traffic on the internet. Encryption is generally perceived as beneficial towards the privacy and security of the communication between endpoints on the internet; however, it is making the traditional network analysis approach based on packet content inspection useless. Also, the recent report published by ENISA[1] recognizes encrypted traffic as a possible serious security threat due to hidden malicious activities[2] that current monitoring tools cannot easily detect. Therefore, it is essential to focus current research activities on encrypted traffic analysis and the retrieval of information about the connections and communicating systems.

The traditional approach for encrypted traffic analysis is its decryption (e.g., using a network proxy). However, this is very computationally expensive and is therefore not feasible for high throughput networks at a large scale. Furthermore, decrypting user communication can be seen as a security transgression and a

[1]European Union Agency for Cybersecurity

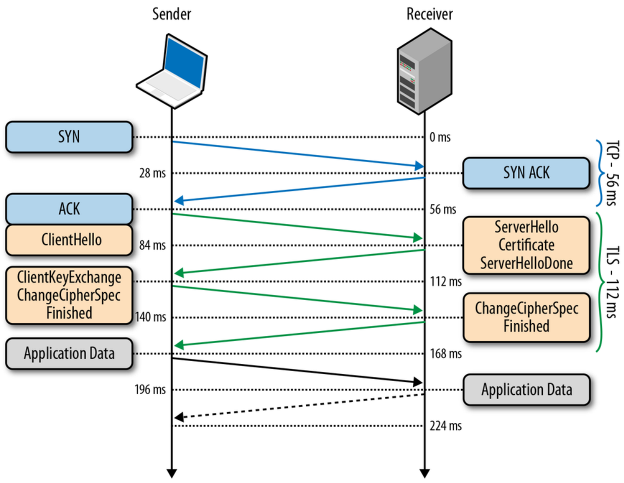


**Figure 1.** TLS Handshake overview

privacy concern. Traffic decryption is usually deployed in a highly restricted environment, such as company networks.

One of the methods that sufficiently preserves the privacy and security benefits of encryption is TLS fingerprinting. It works by gathering information about the client from the unencrypted portion of the TLS communication — the *Client Hello* packet. This packet outlines the parameters of the TLS communication supported by the client. Because different combinations of applications and operating systems support particular subsets of TLS communication parameters, we are able to create a database of applications and their TLS parameters. The database then allows us to evaluate the encrypted portion of the communication and infer the application name or the operating system type.

This information about the user application and operating systems is beneficial for network administrators and security experts. It allows the detection of network policy violations, malware presence, or just insecure and outdated versions of operating systems or applications.

One of the most common fingerprinting approaches is the JA3 fingerprint introduced by Salesforce[3]. JA3 is the de-facto industry standard, supported in high-performance monitoring tools (such as Flowmon ADS[4]) and IDSs (such as Suricata[5]). Even though JA3 is very popular, it lacks a well-maintained and curated public fingerprint database. The existing open-source databases usually lack information, are unmaintained, and sometimes even do not follow the standard JA3 fingerprint format.

Anderson et al.[6] from Cisco research presented a more accurate fingerprinting approach called Mercury [7]. The mercury fingerprint approach allows a much more accurate client taxonomy due to the vastly larger amounts of information stored in the database. Even though the Mercury database includes a well maintained public database and it is in active development, its adoption across the industry is poor. One of the reasons might be the long plain-text fingerprint, which put even more strain on network analysis tools.

Therefore, we propose a new approach to TLS fingerprinting called JA3Cury, which combines the JA3 fingerprint with other context information from the Mercury database. Surprisingly, even though it uses a smaller fingerprint, the JA3cury matching algorithm achieved larger accuracy than the original database based on Mercury fingerprints in our testing environment. Furthermore, we are able to introduce this larger precision to systems that currently use the JA3 approach without redesigning or reengineering these systems.

Our paper is organized as follows: section 2 briefly summarizes the TLS protocol and the concept of TLS fingerprinting. Section 3 describes the JA3 and Mercury fingerprinting methods, section 4 introduces the JA3cury approach, section 5 provide comparisons of results obtained with JA3cury and Mercury, and finally section 6 concludes the paper.

## 2. TLS Fingerprinting

Transport Layer Security (TLS) is a cryptographic protocol designed to facilitate secure and encrypted communication between two parties. It is based on the now deprecated Secure Socket Layer (SSL) protocol. TLS is the de-facto standard for secure communication on the internet, as it is the protocol used by HTTP Secure (HTTPS).

Before two clients can communicate through a TLS secured connection, they must first agree on the parameters of the connection, such as the ciphers supported by both sides. This negotiation happens during the “handshake” phase of the communication. An overview of the handshake can been seen in figure 1.

The *Client Hello* and *Server Hello* messages are always sent without encryption, because the parameters of the communication haven’t yet been agreed upon. This gives us the opportunity the intercept these messages and analyse their contents.

Both fingerprinting methods work by selecting a subset of data from the Client Hello packet of the TLS communication, compiling them into some format, and comparing them with a database of collected and annotated fingerprints.

## 3. Existing solutions

### JA3

The JA3 fingerprinting method works by only extracting information from the following five fields of the Client Hello packet:

- TLS version
- Supported cipher suites
- TLS extension headers
- Elliptic curves
- Elliptic curves point formats

The fingerprint is then generated by concatenating these fields in their decimal representation into a string separated by comas, to generate the following:

```
Version,Ciphers,Extensions,EC,ECPF
```

The TLS standard does not require all these field to be present in the Client Hello packet[8]. If any field is missing, it is replaced with an empty string in the fingerprint representation. The fingerprint is then hashed with MD5.

For example, the following string is a valid JA3 fingerprint in decimal format:

```
769,4-5-47-51-50-10-22-19-9-
21-18-3-8-20-17-255,,,
```

This string would then be hashed with MD5 to generate the string

```
b677934e592ece9e09805bf36cd68d8a
```

which would then be used as the primary key in the fingerprint database.

The JA3 databases usually contain 3 fields: the JA3 string, the JA3 hash, and the description. However, the description of the fingerprint which is then used for identification is not standardized; this results in many fingerprints containing information about the application in a format which is completely unsuitable for further classification and analysis. For example, the string

```
BurpSuite Free
(Tested: 1.7.03 on Windows 10),
eclipse,JavaApplicationStub,idea
```

contains information about the application, the operating system, and some further processes without conforming to a specified format, and thus makes it impossible to parse in large quantities. This results in databases where the fingerprint classification must be taken at face value and no further analysis is possible.

```
Fingerprint
├── Total count
├── Timestamps
└── Processes
    └── Process 1
        ├── Name
        ├── Total count
        ├── Application Category
        ├── SHA256
        ├── SHA256s
        │   └── ...
        ├── Autonomous Systems
        │   ├── AS 1
        │   └── ...
        ├── Hostname Domains
        │   ├── Domain 1
        │   └── ...
        ├── Application Port
        │   ├── https
        │   └── ...
        ├── OS
        │   ├── (F)(N)(V)
        │   │   └── Count
        │   └── ...
        └── ...
    └── ...
```

**Figure 2.** The structure of an entry in the Mercury database.

### Mercury

The Mercury fingerprint format was developed by Cisco by David McGrew and Blake Anderson. The fingerprint itself contains much more information, as it is basically a string representation of important fields in the Client Hello packet in hexadecimal format.

The overall format of the fingerprint is the following:

```
(version)(cipher suites)
   ((extensions)...)
```

Where `(version)` is the hex representation of the advertised TLS version, `(cipher suites)` is a list of hex values of cipher suited offered by the client, and `((extensions)...)` contains the hex representation of the extensions and their values (where applicable).

Furthermore, compared to JA3, the Cisco Mercury database contains much more information, and is formatted so that it encourages further analysis of the results. This format is visualized in figure 2. The database was created by a novel approach of fusing network endpoint data and captured fingerprints[9], so it contains detailed process and contextual information.

The approach of Mercury prefers generating a knowledge base about a network based on a few clients, which is then used for detection[10]. However, the Mercury GitHub repository also includes a well maintained open source database which can be used for identification without the need to generate custom knowledge bases.

The database fields in the Mercury database are much more complex than in a JA3 databases. The fields don't contain a simple 1 : 1 mapping of fingerprint to process, but instead contains many possible processes for each fingerprint, including the number of times they were encountered when building the knowledge base. This number can then be used in further classification. In the latest version of the Mercury database which we used for our experiments, the largest number of distinct processes mapped to a single fingerprint was 9.

## 4. JA3cury

To leverage both the widespread usage of JA3 throughout the industry and the better classification and analysis opportunities presented by the Mercury fingerprinting approach and database, we have devised an approach we call JA3cury. With this approach, we are able to search for JA3 fingerprints in the Mercury database by converting existing Mercury fingerprints to their corresponding JA3 representation.

This conversion works by extracting the relevant information from the Mercury fingerprint, formatting it as a JA3 fingerprint and hashing it using the MD5 hash function. The result is a fingerprint database based on the Cisco Mercury database that can be indexed using JA3 fingerprints.

An example of this conversion can be seen in figure 3. As is shown in the figure, the conversion from Mercury to JA3 is destructive; some information is lost during the conversion. This means that the Mercury database that previously contained only unique fingerprint entries now contains duplicate fingerprint entries for some fingerprints. Out of the 9,060 entries in the database we used, 1,914 unique fingerprints were lost; this is a 21.1% decrease in fingerprint count.

Initially, we were worried this would lead to a decrease in accuracy of client and process identification compared to the original database; however, as we will show in Section 5, the accuracy in our experiments remained the same or even increased for some certain scenarios.

This decrease in the number of unique fingerprints also influenced the approach of our classification algorithms to fingerprint collisions. Kotzias et al. found that around 7.3% of JA3 fingerprints cause collisions with each other[11]. However, since our approach of converting fingerprints introduces collisions into the database, our classification algorithms were designed to deal with them and they didn't cause a perceptible decrease in detection accuracy.

Furthermore, it is important to note that the JA3cury approach congregates the Client Hello classifications over some time period, rather than identifying and classifying a single Client Hello. This has lead to better classification results overall, as gathering the information over many handshakes increases the operating system detection accuracy, as well as overall process detection accuracy due to the systems ability to overcome statistical anomalies.

## 5. Detection

Thanks to the complexity of the Mercury/JA3cury database, we were able to try many different fingerprinting approaches with varying degrees of accuracy. Overall, we created 7 algorithms for traffic classification. Each algorithm took into account different combinations of information from the database, as well as some contextual information about the Client Hello packet, such as the destination port, server domain name, etc.

During detection, we compared three sets of results:

- As a baseline measurement, we used an unmodified Mercury classification created with the official `pmercury` utility.
- Our classification algorithms performed using the unmodified version of the database using Mercury fingerprints.

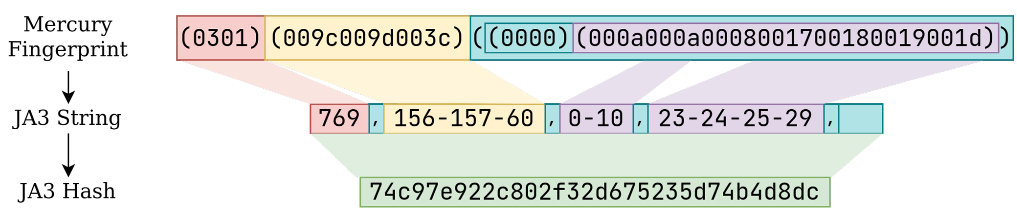


**Figure 3.** Converting a Mercury fingerprint to JA3

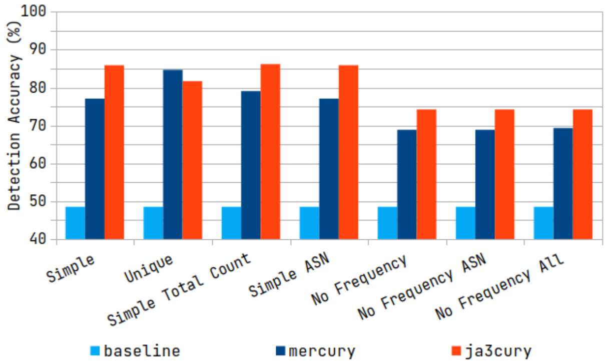


**Figure 4.** Process classification results.

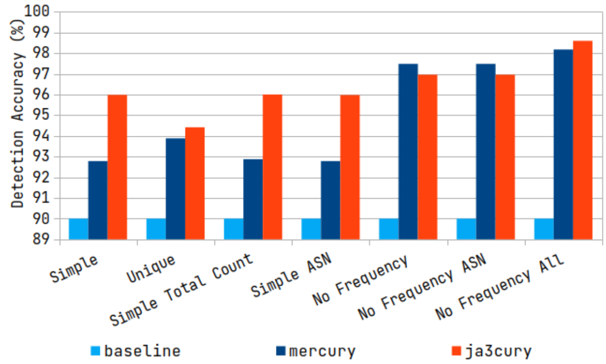


**Figure 5.** Category classification results.

- Our classification algorithms performed using the JA3database.

Our datasets contain over 48,000 Client Hello packets collected over 6 home networks made up of personal computers, laptops, home servers, and phones. All the major operating systems (Windows, Mac, Linux, Android, iOS) were represented in our experiments.

Some of our classification algorithms even exposed problems with the current version of the Mercury database; it was created on corporate Cisco networks, and thus skews heavily towards enterprise applications, such as Cisco Webex, and towards very specific operating systems, mainly Mac OS. However, we discovered that JA3cury is largely able to overcome this due to the meshing together of different fingerprints from the original Mercury database, which tends to average out the discrepancies.

## Process and Category Detection

Each process in the database contains a classification into many categories, such as productivity, security, or gaming. This means that process and category detection are closely connected together. However, we discovered it is possible to obtain a high accuracy of category detection even with relatively low process detection accuracy. This is due to the fact that the erroneous classifications tend to get averaged out due to the vastly lower number of categories than processes, which leads to larger detection scores.

The average results for process classification of the top 5 processes for each client can be seen in figure 4.

Our JA3cury method was able to outperform the baseline results generated by the official `pmercury` for all our classification algorithms. Furthermore, our modified JA3cury database outperformed the original Mercury database in all but one experiment.

Furthermore, the category detection using our algorithms was also successful, viz figure 5. Again, our classification using JA3cury was more successful overall than either the original `pmercury` detection, or even the detection using our algorithms and the original Mercury database.

The difference in scoring processes and categories between the default Mercury and our JA3cury approaches is well illustrated in figure 6. This graph shows the scores of different processes on the Y axis, with each process being represented by a bar on the X axis. Each process is also scored with Mercury and JA3cury. You can see that JA3cury identified more processes and categories, and it attributed a higher score to correct processes compared to Mercury.

## Operating System Detection

The information about operating system classification in the Mercury database is dependent on the process classification, as the operating system information is nested inside each process (see figure 2). Furthermore, the database unfortunately does not contain information about mobile operating systems; instead, they tend to be classified as desktop operating systems with the most similar kernel architecture; MacOS for iOS devices, and Linux for Android devices.

The database contains operating system information split into three parts: the family (Linux, MacOS, Windows), the name (Windows 10 Professional, Linux 4.19, ...), and the build version (10.5.6.7, ...). For our experiment, we decided to classify the operating system using a tree structure with depth of 4, where the operating system frequency trickles down into the leaf nodes. An example of this tree can be seen in figure 8.

Furthermore, the tree is sorted such that each parent has its children ordered from the most frequent to the least frequent. This allows us to find the most probable operating system by taking the leftmost nodes. In this case, the classification would result in `WinNT - Windows 10 Enterprise - 10.0.18363`.

The operating system classification was performed on all clients in each network. The comparison of result for detection of the operating system family using our classifiers can be seen in figure 7. The figure

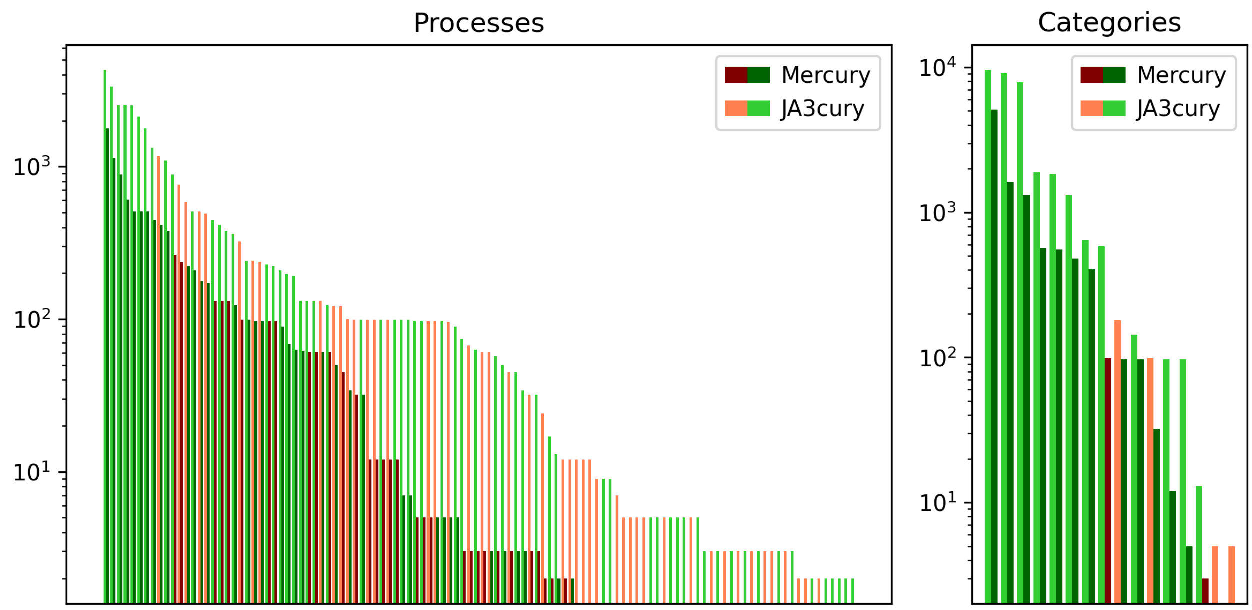


**Figure 6.** Detailed look at JA3cury and Mercury classification scores for one client.

contains only results created with our classification algorithms, because `pmercury` doesn't return information about the operating system.

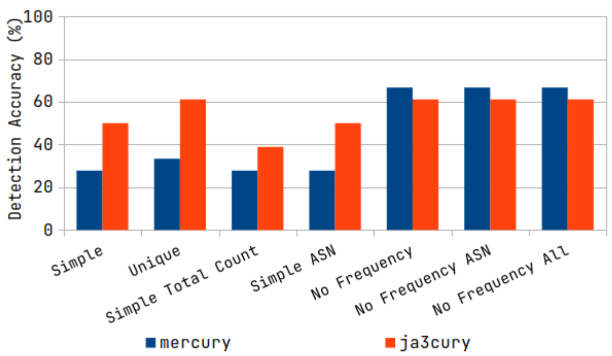


**Figure 7.** Operating system classification results.

JA3cury was able to detect the operating system of a client more accurately overall. However, the operating system detection was less reliable on our datasets compared to process detection due to the prevalence of Linux machines in our datasets. The database, however, contains many more entries for Windows and Mac OS, than it does for Linux. Furthermore, the fact that the database doesn't contain mobile operating systems leads to a lower accuracy as well.

## 6. Conclusion

In conclusion, we have developed an alternative TLS fingerprinting approach based on the strengths of JA3 and Mercury fingerprinting. This approach allows us to utilize the more content rich Mercury database in a setting where we would have to rely on the results of JA3 without any further analysis. Furthermore, our approach is compatible with most existing fingerprinting modules thanks to the wide adoption of JA3, and can be used with existing modules and infrastructure without the need of re-engineering or redesigning these systems. Because our approach takes into account Client Hello packets over a time period, and doesn't classify each packet separately, our approach is able to overcome the disadvantage of missing information in the JA3 fingerprint compared to the full Mercury fingerprint.

Our method has proven to be at least as accurate as default Mercury fingerprinting. Furthermore, when used outside of corporate networks, it tends to be more accurate. Furthermore, the process category was detected correctly for all major categories.

The major area of further development could include increasing the accuracy of operating system detection and the addition of mobile operating systems into the Mercury database.

## Acknowledgments

We would like to thank Blake Anderson and David McGrew for their comments and feedback.

## References

[1] Google Transparency Report. HTTPS encryption on the web – Google Transparency Report, 2021.

[2] ENISA. Encrypted Traffic Analysis, April 2020.

[3] John Althouse, Jeff Atkinson, and Josh Atkins. salesforce/ja3, June 2017. original-date: 2017-06-13T22:54:10Z.

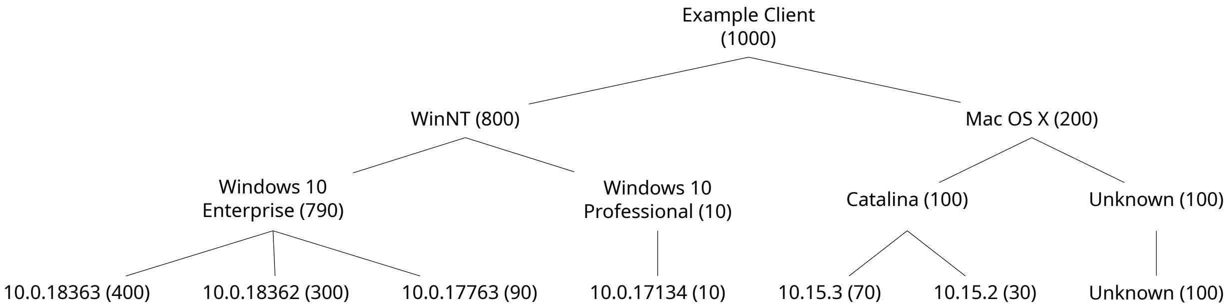


**Figure 8.** OS Classification Tree


[4] Flowmon. Encrypted Traffic Analysis.

[5] Suricata. 6.17. JA3 Keywords — Suricata 6.0.1 documentation, 2019.

[6] Blake Anderson and David McGrew. Accurate TLS Fingerprinting using Destination Context and Knowledge Bases. *arXiv:2009.01939 [cs]*, September 2020. arXiv: 2009.01939.

[7] Blake Anderson, David McGrew, Brandon Enright, Lucas Messenger, Adam Weller, Andrew Chi, and Shekhar Acharya. cisco/mercury, August 2021. original-date: 2019-08-30T21:58:25Z.

[8] Eric Rescorla. RFC 8446 - The Transport Layer Security (TLS) Protocol Version 1.3. Technical report, August 2018.

[9] Blake Anderson, David McGrew, and Keith Schomburg. The generation and use of tls fingerprints, Jan 2019.

[10] Blake Anderson and David McGrew. Video correspondece in regards to cisco cognitive intelligence and cesnet collaboration., Mar 2021.

[11] Platon Kotzias, Abbas Razaghpanah, Johanna Amann, Kenneth G. Paterson, Narseo Vallina-Rodriguez, and Juan Caballero. Coming of age: A longitudinal study of tls deployment. In *Proceedings of the Internet Measurement Conference 2018*, IMC '18, page 415–428. Association for Computing Machinery, Oct 2018.