A Closer Look at JA4 Fingerprints for Application Identification in Encrypted Traffic

Blinded version.

***Abstract*—Encryption of network communications protects transmitted data from eavesdropping, but it also limits protocol identification, network monitoring, and malware detection. This is an important issue for cybersecurity because monitoring tools cannot access the encrypted content to identify applications and detect signatures of malicious communications. One technique for identifying encrypted traffic is TLS fingerprinting, which observes a TLS handshake and maps TLS initialization values to applications. A popular implementation of TLS fingerprinting is the JA3 and JA3S fingerprints, introduced in 2015 and updated by JA4+ in 2023. This report observes the similarities and differences between the JA3 and JA4 fingerprints, and analyzes how the JA4+ fingerprints improve application identification. Using a representative set of different mobile and desktop applications, we demonstrate the effectiveness of JA4+ fingerprints in combination with other TLS attributes. We also discuss the importance of automated generation of TLS fingerprints and present a technique for automatically learning TLS fingerprints for mobile applications and desktop services.**

***Index Terms*—TLS fingerprinting, JA4+, encrypted traffic, application identification**

I. Introduction

Widespread encryption of network communications today includes not only user applications (web, email, file transfer) but also system services such as DNS. Encryption has a huge impact on network monitoring, especially on traffic visibility. It limits the identification of protocols and applications transmitted, preventing successful detection of malware (viruses, botnets, ransomware) and filtering of unwanted applications (TikTok) transmitted through encrypted tunnels.

There are several methods for analyzing encrypted data to provide application visibility [1]:

- *SSL/TLS proxy* – Major network hardware vendors implement an SSL/TLS proxy inside their firewalls[1]. The SSL/TLS proxy establishes a TLS connection between the TLS client and the proxy, and the proxy and the TLS server. This allows an SSL/TLS proxy to inspect and filter transmitted traffic, even if it is encrypted. This solution requires a support for a special LAN configuration. Because the proxy breaks a TLS connection, it limits TLS renegotiation and certificate validation. This solution is often used in enterprise networks.
- *ML/DL classification* – This technique uses machine learning (ML) or deep learning (DL) to build classification models of application protocols using statistical features extracted from annotated samples of encrypted traffic [2], [3]. The quality of this classification is closely related to the quality of the training data. For example, a model trained on one network may not work properly on a network with different traffic. In addition, ML/DL classifiers are often memory and computationally intensive, which limits their use for real-world networks.
- *Context-based protocol identification* – Instead of analysing encrypted data, the context-based protocol identification focuses on a context in which encrypted data is transmitted, see [4], [5]. The context is represented by relevant flows for an encrypted session. By observing a number of connections, a size of data transmitted, DNS requests, etc., the context model of an application is built. The context-based models can be sensitive to the training data and the environment in which they are used.
- *TLS fingerprinting* – TLS fingerprinting extracts selected attributes from a TLS handshake and creates a so-called *TLS fingerprint*, which is a hash of the TLS attributes. The TLS parameters exchanged within the TLS handshake can identify a sending application. Unlike the ML/DL or context-based classification, TLS fingerprints are easy to extract and compute, making them an effective tool for identifying encrypted traffic. However, TLS fingerprints can overlap, i.e., they are not always unique to a particular application. TLS fingerprints are tied to an underlying TLS library, so changes to the library produce new fingerprints, affecting their stability. Further details of TLS fingerprinting are discussed in Sec. III.

In recent years, JA3 and JA3S fingerprints have been integrated into popular network analysers such as Wireshark (since version 3.6.0), Network Miner (since version 2.7.3) or nDPI, as well as into intrusion detection systems such as Surikata or Bro[2]. This has made TLS fingerprinting a popular method of identifying encrypted traffic. The arrival of the JA4+ fingerprints in 2023 raised the question of how effective JA4 fingerprints are at identifying application protocols compared to JA3 fingerprints. This study tries to answer this question.

A. Objectives

This experience report examines the effectiveness of JA4+ fingerprints for application identification compared to the previous versions, JA3 and JA3S. Our study addresses the following research questions:

1) How stable and accurate are JA4+ fingerprints compared to JA3 and JA3S fingerprints?

[1]E.g., Hillstone SSL Proxy or Cisco SSL/TLS Proxy.

[2]For a list of supported tools, see https://github.com/salesforce/ja3.

2) How can we automate the creation of a JA4+ fingerprint database for common network applications?
3) What combination of TLS attributes give the best results for application identification?

B. Contribution

The main contribution of the paper is to compare the efficiency of JA3 and JA4 fingerprints for identifying network applications in the encrypted TLS tunnels. Using an annotated dataset of mobile and desktop applications, we observe accuracy and uniqueness of different TLS fingerprints and their combinations. We mainly focus on comparison of JA3/JA3S fingerprints with JA4/JA4S fingerprints. The paper also address an issue of automated generation of annotated TLS fingerprint database that can be later use for application identification. Finally, we discuss the benefits and limitations of using of JA4+ fingerprints in practice.

C. Structure of the paper

The paper is structured as follows. In Section II, we present a recent research in the area of TLS fingerprinting. Section III gives an overview of TLS fingerprinting techniques, describes TLS features used for fingerprinting, and compares JA3/S and JA4+ hashes. Section IV describes our testing environment for creating TLS fingerprints of mobile and desktop applications. The core of our study is in Section V, where we show the results of our experiments with JA3 and JA4 fingerprints. The final section concludes our study and discusses future directions for the research in TLS fingerprinting.

II. Related Work

One of the first comprehensive studies of the use of TLS fingerprinting was published by Anderson et al. in [6], where the authors observed TLS features and examined how they detected malware in TLS-encrypted flows. They focused on the cipher suites offered, TLS extensions, the length of the client's public key, and the data obtained from the server certificate. In addition to TLS attributes, they also used flow statistics (the sequence of packet lengths and inter-arrival times), and byte distribution. By combining these attributes on a large dataset of malicious and enterprise TLS flows, they were able to achieve 99.6% accuracy in classifying malware. When using TLS attributes alone, the accuracy for different malware families ranged from 63.6% to 100%. They found a bias caused by the underlying operating system of the sandbox. They also found that some malware families attempt to mimic the behavior of legitimate applications such as Firefox. Our work extends the scope of TLS traffic identification beyond the malware. We combine various TLS attributes that allow more accurate identification of network applications and distinguish traffic from the underlying OS or tracking services.

In their second paper, Anderson and McGrew [7] examine the evolution of TLS usage in applications over time. Their study tracks the use of different TLS versions, cipher suites, and extensions, collecting session data such as associated processes, destination IP addresses, and ports. They use unhashed data to measure fingerprint similarity via Levenshtein distance which allows the identification of similar fingerprints. While their work focuses on general TLS trends and fingerprint evolution, our study focuses on application identification.

Frolov and Wustrow discussed the use of TLS in censorship circumvention [8]. They analyzed TLS implementations of several popular censorship circumvention tools, such as Signal, Tor, TapDance, VPNs, etc., and observed how their TLS fingerprints differed from the real-world applications. Similar to Anderson and McGrew [7], they compute the closeness of similar fingerprints based on a Levenshtein distance. They observed the standard and non-standard parameters used by these tools and the effect of random values on the detection of these tools. They also developed an uTLS library that generates arbitrary Client Hello messages and evades identification. We also use TLS fingerprints to identify applications and observe the uniqueness of the fingerprints, but the goal of our research is different.

Fingerprint overlapping is addressed by Anderson and McGrew in [9]. They extend the TLS fingerprinting to include the destination address, port and Server Name Indication (SNI) so that their fingerprints are more accurate using the target context. This is similar to the server fingerprints: JA3S and JA4S hashes. We also use the server attributes along with the SNI values. As in their previous work [7], the authors measure the similarity between fingerprints using Levenshtein distance. In addition, they add weights to the attributes based on the information gain ratio. Rather than identifying families of applications, our work attempts to identify the applications themselves.

Our work extends the previous work by Matousek et al. [10], which investigates the reliability of TLS fingerprints for mobile application identification. Unlike their work, we compare JA4 and JA4S fingerprints for application identification in more detail, and discuss the contribution of additional attributes to identification. We also consider the impact of TLS version 1.3 on TLS fingerprinting.

III. TLS Fingerprint Overview

Transport Layer Security (TLS) [11], [12] is a protocol defined on top of the transport layer that provides encryption, data integrity, and authentication for application protocols. Typically, TLS is implemented over the TCP transport protocol. After a TCP connection is established, the TLS client and server perform a TLS handshake and initialize security parameters to establish a secure TLS channel. The handshake includes negotiating a TLS version, selecting ciphers for encryption, authentication, data integrity, and key exchange methods, generating session keys, and so on. Once the TLS handshake is complete, the next packets exchanged between the client and server are encrypted, as shown in Figure 1.

The essential part of TLS fingerprinting is the TLS handshake, specifically the Client Hello and Server Hello packets, which are transmitted unencrypted and contain a list of cipher suites, extensions, and other parameters supported by the client and the server. TLS defines a large set of possible parameter

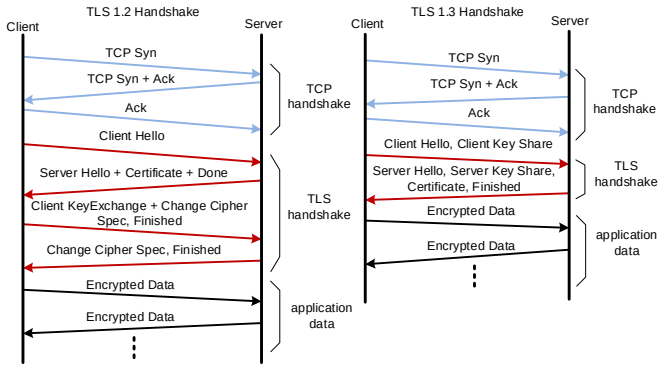


Fig. 1: TLS handshake version 1.2 and 1.3

values and their combinations, so that a selection of offered parameters represents a distinctive feature of the client or server that can be used for application fingerprinting.

There is a significant difference between TLS versions 1.2 and 1.3. TLS 1.3 makes the handshake more efficient by reducing the number of packets exchanged. It removes obsolete and insecure cipher suites, reducing the number of parameters offered. It also encrypts all server extensions that are not required to establish the cryptographic context. This includes server certificates, which is used to generate a JA4X fingerprint. As a result, TLS 1.3 reduces the list of TLS parameters that can be a part of the TLS fingerprint, see III-A.

We mentioned above that TLS is typically implemented over TCP. This is true, but TLS can also be implemented over UDP when it is a part of the QUIC protocol [13], [14]. QUIC is a connection-oriented client-server protocol originally developed by Google. It implements its own handshake to negotiate cryptographic and transport parameters before establishing a connection. QUIC integrates the TLS handshake into the CRYPTO frames exchanged by QUIC [15]. It only supports TLS version 1.3 and higher. Thus, extracted TLS parameters can also be used to create TLS fingerprints for applications encapsulated by QUIC.

A. Description of TLS Attributes

In this part we describe the TLS fields extracted from the Client and Server Hello packets to create JA3, JA3S, JA4 and JA4s fingerprints. The fields are as follows:

- *Version*: The TLS Client Hello packet contains two version fields: TLS Record Layer version, which is the lowest version supported by the client, see [11, Appendix E], and the Handshake Protocol version, which is the requested TLS version. These versions may differ for backward compatibility. TLS client and server fingerprints use the Handshake Protocol version.
- *Cipher Suites*: A list of cipher suites defines possible combinations of a key exchange method and algorithms for authentication, encryption, and data integrity. Valid combinations are standardized by IANA[3]. The current IANA cipher suite list contains 351 different cipher suite combinations, with 37 cipher suites recommended by RFC 5246 for TLS 1.2 and 9 cipher suites recommended for TLS 1.3. Note, that the TLS 1.3 cipher suites only include symmetric ciphers and the hash function. This reduces the list of possible combinations compared to TLS 1.2 and earlier versions.
  The list of cipher suites may include random values called GREASE (Generate Random Extensions And Sustain Extensibility) [16] for client or server compatibility testing. These random values introduce instability into TLS fingerprinting, so they are excluded from the fingerprint computation. The list of GREASE values is given in [16].
- *Extensions*: A client or server may request additional functionality, which is represented by a list of requested or supported TLS extensions. Possible values are standardized by IANA[4]. Currently IANA defines 63 different extensions. Similar to the cipher suites, the list of extensions may include GREASE values, which are omitted from TLS fingerprinting.
- Supported Groups (SG): This TLS extension (extension no. 10) specifies the named groups that the client supports for key exchange, ordered from most preferred to least preferred. In versions of TLS prior to 1.3, this extension was called Elliptic Curves and contained only elliptic curve groups. GREASE values are removed from SG.
- *Elliptic Curve Point Format (EC Format)*: The EC format (extension no. 11) is required to compute the Elliptic Curve Diffie-Hellman key exchange. It specifies the encoding supported by the client when transmitting Elliptic Curve values. IANA only defines three values for this extension. The EC format extension is no longer used, and the Client Hello contains only the default value 0 (uncompressed). However, our experiments show that three different values appear in this field, see Table II.
- *Server Name Indication (SNI)*: This TLS extension (extension no. 0) allows a client to specify a domain name of the server it is contacting [17]. This is useful in cases where the destination server has multiple domain names, so that the IP address alone is not sufficient to select which domain a user is trying to reach. SNI is not a part of the TLS fingerprints, only the JA4 fingerprints indicate whether the SNI is present or not. However, the SNI plays an important role in annotating TLS sessions or as a distinguishing attribute for a fingerprint belonging to multiple applications. The SNI is only useful for application identification when the client is contacting a fixed service, such as a weather forecast server. In the case of web browsers, the SNI has a little value for application identification because the SNI value changes with each new web server requested.
  There has been an IETF draft to encrypt SNI using ESNI (Encrypted SNI) or Encrypted Client Hello (ECH) methods. These extensions have not yet been approved

[3]See IANA TLS Parameters.

[4]See IANA TLS Extensions.

by the IETF, so the SNI is still transmitted unencrypted.

- *Application Layer Protocol Negotiation (ALPN)*: When multiple application protocols are supported by a single server-side port number, such as port 443, the client and the server need to negotiate an application protocol to use for each connection [18]. A list of available ALPN values (TLS extension no. 16) is standardized by IANA[5] and its first two characters form the JA4+ fingerprint.
- *Supported Versions*: This attribute (extension no. 43) is implemented in TLS 1.3 [12] and contains a list of TLS versions supported by the client, in order of preference. The JA4 fingerprint uses the highest version offered. If this extension is empty, the handshake protocol version is used (see *Version* above).
- *Signature Algorithms*: TLS 1.3 provides a list of supported hash algorithms (extension no. 13) used for signatures. This list is a part of the JA4 fingerprint.

The above mentioned TLS fields are transmitted during the TLS handshake. Their values play an important role in identifying TLS-encrypted application protocols. These fields form a TLS fingerprint, see Table I. JA4 also adds a flag to identify the TLS or QUIC protocol.

TLS Attribute	JA3	JA3S	JA4	JA4S
TLS/QUIC protocol			x	x
Handshake Version	x	x	x	x
Cipher Suites	x	x	x	x
Extensions	x	x	x	x
Supported Groups	x			
EC Format	x			
SNI			x	
ALPN			x	x
Supported Versions			x	x
Signature Algorithms			x	

TABLE I: TLS attributes used in JA3 and JA4 fingerprints

We also observed the uniqueness of individual attribute values and their contribution to the uniqueness of TLS fingerprints. Table II shows the occurrence of unique values in our dataset of 5.140 TLS sessions from 63 different applications. We use the entropy, which represents the degree of uncertainty in the value. This means that attributes with higher entropy contribute more to the uniqueness of the fingerprint and help to better identify the application. Low entropy means that many TLS connections have the same attribute value, in which case the attribute does not help much to distinguish applications.

An empty column indicates the percentage of connections with missing values. If every TLS connection contains a non-empty value, then the number of empty values is 0.

As expected, the most important TLS attributes for fingerprinting are client and server *cipher suites* and *extensions*. It appears that *SNI* can play an important role, but *SNI* changes with each destination, so it is not a stable attribute for all applications. Comparing the JA3 and JA4 fingerprints, we can see that the unsorted JA3 cipher suites have more unique values and higher entropy. Surprisingly, the number of unique

[5]See TLS ALPN Protocol IDs.

TLS Attribute	Unique	Empty (%)	Entropy
TLS Version	2	0	0.013
JA3 Client's Cipher Suites (unsorted)	37	0	0.753
JA4 Client's Cipher Suites (sorted)	32	0	0.688
JA3 Client's Extensions (unsorted)	1360	0	0.636
JA4 Client's Extensions (sorted)	62	0	0.703
Client's Supported Groups (unsorted)	46	0	0,352
EC Format	2	14.6	0.411
SNI	1205	0.10	0.898
ALPN	12	0.06	0.575
Supported Versions	37	33.93	0.527
Signature Algorithms	14	0.06	0.578
JA3 Server's Cipher Suites (unsorted)	14	0	0.628
JA4 Server's Cipher Suites (unsorted)	14	0	0.628
JA3 Server's Extensions (unsorted)	81	0	0.655
JA4 Server's Extensions (unsorted)	34	0	0.737

TABLE II: Unique values in TLS attributes (total 5.140)

JA3 extensions is much higher than the number of unique JA4 extensions. This is because the JA4 fingerprint excludes the SNI (no. 0) and ALPN (no. 16) extensions from the list of extensions, and the values are sorted. Omitting SNI and ALPN from the server's extension list also causes the number of JA3S extensions to be greater than the number of JA4S extensions. The server ciphers suites for the JA3 and JA4 are computed in the same way, so their entropy is the same.

B. Computing JA3 and JA3s Hashes

The JA3 fingerprint consists of the following fields extracted from the Client Hello packet: Version, Cipher Suites, Extensions, Elliptic Curves, and EC Format. These values are concatenated using a comma to delimit each field and a dash to delimit each value in the field. The order of multiple values in the field is preserved. GREASE values in cipher suites and extensions are omitted from the calculation. The final string hashed using the MD5 algorithm, is called *a JA3 fingerprint*[2], see Figure 2. The JA3 fingerprint identifies a client application opening a TLS connection.

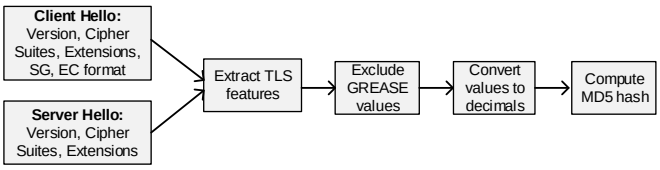


Fig. 2: Computing JA3 and JA3S fingerprints

The JA3S fingerprint describes the behavior of a TLS server. JA3S is a concatenation of three fields extracted from the TLS Server Hello: Version, Cipher Suites, and Extensions. Unlike the Client Hello, the Server Hello only transmits TLS values that have been selected by the server side for communication, so the list of possible values is shorter. More details on JA3 and JA3 implementations can be found at https://github.com/salesforce/ja3.

C. JA4 and JA4s Fingerprints

JA4 and JA4S fingerprints[6] have a different structure. Instead of a single hash value like JA3 or JA3S, JA4+ fingerprints

[6]See JA4+ Network Fingerprinting Blog.

consist of three parts linked by the underscore: a_b_c. The a part concatenates the following values: the underlying protocol ("t" for TCP, "q" for QUIC), the TLS supported version ("10" for SSL 3.0, "12" for TLS 1.2, "13" for TLS 1.3, etc.), the number of cipher suites without GREASE values, the number of extensions without GREASE values, and the ALPN string. If an ALPN field is empty, the value "00" is selected, otherwise the first two characters of ALPN are used for the fingerprint.

The b part of the JA4 is a SHA256 hash calculated from the cipher suites. After excluding GREASE values the cipher suites are sorted, converted to hexadecimal format without the leading "0x", and concatenated by the comma. The calculated hash is truncated to twelve characters.

The c part of the JA4 is also a SHA256 hash calculated from a list of extensions and signature algorithms concatenated by the underscore. The list of extensions is converted to hexadecimal format, sorted, excluding GREASE values, SNI, and ALPN extensions. The values are separated by a dash. The list of signature algorithms is unsorted and converted to hexadecimal format, see Figure 3.

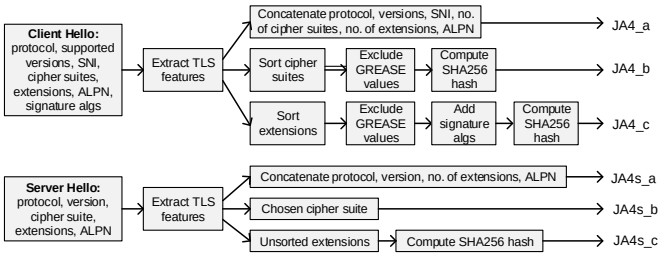


Fig. 3: Computing JA4 and JA4S fingerprints

The JA4S fingerprint is also made up of three parts a_b_c, where part a is a concatenation of the protocol, the supported version[7], the number of extensions, and the ALPN. The part b is the selected cipher suite, and the c part is a SHA256 hash computed from a list of supported TLS extensions separated by commas and truncated to twelve characters.

D. Additional TLS Fingerprints

Besides the widespread TLS fingerprints like JA3 and JA4, there are other TLS fingerprints defined:

- *JA4H*: HTTP Client Fingerprint based on HTTP requests. The fingerprint consists of $a\_b\_c\_d$ parts and includes the following attributes: HTTP method, HTTP version, Cookie (yes/no), Referer (yes/no), number of HTTP headers, Accepted Language, hash of headers, hash of cookie fields, hash of cookie fields and values.
- *JA4X*: X.509 TLS Certificate Fingerprint identifies the application that created the certificate. It does not contain values from the certificate but the name and order of the Relative Distinguished Names (RDNs). The JA4X contains three hashes concatenated by a dash: hash of issuer RDNs, hash of subject RDNs, and hash of Extensions. The JA4X fingerprint is useful for malware detection, but requires a X.509 certificate from the Server Hello, which is encrypted in TLS v1.3 and higher.
- *JA4TS/C*: TCP Client Fingerprint (JATC) and TCP Server Fingerprint (JATS) provide TCP fingerprinting similar to `nmap`. They include the following attributes obtained from the TCP header: TCP Window Size, TCP Options, TCP Maximum Segment Size, TCP Window Scale, and TCP Retransmission Timings. These fingerprints are more useful for identifying operating systems.
- *HASSH*: HASSH is the SSH fingerprint invented to identify specific client and server SSH implementations [8]. [9]. It analyses the SSH initialization and extracts a list of supported key exchange methods, encryption and authentication algorithms, and compression methods. These strings are concatenated by the colon and hashed using the MD5 algorithm.
- *JARM*: JARM is an active TLS server fingerpriting tool[10] that observes the TLS configuration of the server by sending ten specially crafted TLS Client Hello packets and analyzing the response. By running the JARM test repeatedly, we can see if the server configuration changes. The JARM is able to identify the default configuration of application servers.

IV. Testing Environment

This section outlines our environment for automated generation and annotation of TLS-based application communications, which includes i) an Android emulator for running application packages, and ii) a virtual Windows host simulating a sandbox for desktop applications. Both tools capture network communications in PCAP files and label connections based on application processes. We focus on Android and Windows applications due to their market dominance, although applications from other environments can be similarly analyzed.

A. Generating TLS Fingerprints of Mobile Apps

To generate the TLS fingerprints of mobile applications, we have created a tool[11] that emulates the behavior of mobile apps using the Android Virtual Device (AVD). Our tool downloads an APK file containing the app and installs it on the virtual device. We then emulate its behavior using the ADB shell and the `monkey` command. Typically, the apps opens a number of connections to the application server, which are captured using `tshark`. The extracted TLS connections are then parsed for TLS Client and Server Hellos and fingerprints are created. The tool generates JA3/S, JA4/S, and JA4X fingerprints, which are later stored in the TLS database. The database currently contains 417 fingerprints from 34 mobile apps (e.g., Discord, Messenger, Facebook, Signal, TikTok).

[7]If a supported version is 0xfb1a (Facebook draft), the 00 is used.

[8]See https://engineering.salesforce.com/open-sourcing-hassh.

[9]See https://engineering.salesforce.com/open-sourcing-hassh-abed3ae5044c/.

[10]See https://github.com/salesforce/jarm.

[11]Blinded version.

B. Generating TLS Fingerprints of Windows Applications

We generate TLS fingerprints for desktop applications by running a script on a Windows machine to capture network communications and record processes with open network sockets. This allowed us to tag communications with specific application names. Using this method, we identified 377 fingerprints from 33 applications, including Spotify, WhatsApp, Chrome, Firefox, Edge, and Tor. The high number of fingerprints from web browsers, such as Chrome with 242 unique JA3 fingerprints and 5 JA4 fingerprints, and MS Edge with 77 JA3 and 10 JA4 fingerprints, is due to JA4 rearranging TLS extensions while JA3 retains the original order.

We also discovered that many popular desktop applications in the Microsoft Store, like Instagram, TikTok, Pinterest, and Facebook, are deployed as Progressive Web Apps (PWAs). PWAs are web applications that provide an app-like experience and can be installed on a device to run in a dedicated window without a traditional browser interface. Consequently, it is impossible to identify such applications by their process names, as they run within the web browser process.

C. Filtering Noise Sessions

As mentioned in [10], network applications often open TLS connections not only to the application servers, but they also communicate with analytics, tracking, or advertising servers, creating *TLS noise*. These noise TLS connections add new TLS fingerprints that do not identify the specific application and are common to different services. For example, the fingerprint of the TLS connection to `www.google-analytics.com` is shared by 11 different applications in our dataset.

The important task in building an accurate TLS fingerprint database of network applications is to eliminate the noise. One way to achieve this is to monitor an SNI value and exclude TLS sessions where the SNI matches a domain name of tracking and ad servers. There are several lists of tracking and analytics servers available on the web. We combined some of them and created a list of ad servers with 181,205 entries.

Using this approach, we eliminated about 57,6% of the TLS connections from our dataset, which corresponded to the noise. The remaining TLS connections are used for our TLS analysis.

D. Testing Dataset

For our experiments, we captured TLS communications from mobile (Section IV-A) and desktop applications (Section IV-B). We automatically annotated the entries based on the source ports and associated processes, and manually verified them using SNI. Table III gives an overview of our dataset[12]. The dataset includes bi-directional TLS sessions where the Client Hello message received a corresponding Server Hello response. Unanswered connections are excluded from the analysis. It includes 89 unique JA3 fingerprints from mobile applications and 377 unique fingerprints from desktop applications. Due to some overlapping fingerprints, the total number of unique JA3 fingerprints in the dataset is 459.

[12]A link to the dataset has been removed in the blind version.

Captured PCAP files	96
Total number of TLS connections	5.140
Total number of application fingerprints	2.959
Total number of noise fingerprints	2.180
Number of unique client JA3 fingerprints (without noise)	459
Number of unique client JA4 fingerprints (without noise)	80
Number of unique client JA3S fingerprints (without noise)	89
Number of unique client JA4S fingerprints (without noise)	109

TABLE III: Overview of the testing dataset

Since the goal of this study is not to develop a comprehensive database of TLS fingerprints, but rather to compare the effectiveness of JA3 and JA4 fingerprints, we use only a representative sample of applications for our experiments. However, using a proposed technique for automatic fingerprint generation, an arbitrarily large database of accurate fingerprints can be created for network applications.

V. Experiments with TLS Fingerprinting

A. Efficiency of Different TLS Fingerprints

This section presents a comparative analysis of the differences between JA3/S and JA4/S fingerprints in terms of their ability to identify applications, as observed in our dataset. To accurately identify an application based on TLS fingerprinting, two conditions should be met: the fingerprint must be unique, and there should be a set of fingerprints that characterise the application. Table IV shows the identified metrics for different types of fingerprints (client, server) and their combinations, along with the percentage of unique fingerprints (associated with only one application) and the percentage of applications covered by these unique fingerprints.

Fingerprint type	Total	Unique	Covered apps	Efficiency
JA3	459	95.4%	44.4%	1.17
JA4	80	62.5%	33.3%	2.20
JA3S	89	46.1%	34.9%	3.18
JA4S	109	48.6%	39.7%	2.78
JA3+JA3S	637	95.9%	76.2%	1.11
JA4+JA4S	300	86.7%	71.4%	1.28
JA3+JA3S+SNI	1185	98.6%	96.8%	1.016
JA4+JA4S+SNI	1020	97.9%	96.8%	1.022

TABLE IV: Efficiency of different TLS fingerprints

Table IV compares the efficiency of different TLS fingerprints for application identification. The experiments were performed using our annotated dataset which includes 67 mobile and desktop applications with 2,959 TLS fingerprints. The table compares the efficiency of different TLS fingerprints and their combinations for application identification. The goal is to find the best combination of TLS attributes that covers the maximum of applications and provides fingerprints that are unique to each application.

To measure the efficiency of the fingerprints, we use the metric E, which computes the average number of applications per fingerprint. Let $i = 1 \dots n$ be the number of unique fingerprints and the function $f(i)$ maps each fingerprint to an application. For example, if a fingerprint is assigned to two

different applications, $f(i) = 2$. Then the efficiency E of the fingerprint type is calculated as follows:

$$E = \frac{\sum_{i=1}^{n} f(i)}{n} \quad (1)$$

For example, for the JA3 fingerprint we find 459 unique fingerprints (n=459). These fingerprints are unique for 95.4% of the applications, which means that some fingerprints are shared between several applications. However, the JA3 fingerprints only cover 44.4% of the applications, the rest of the applications do not have unique fingerprints. The efficiency of 1.17 means, that one fingerprint is used for 1.17 applications. So the optimal fingerprint type has a coverage of 100% and an efficiency of 1.

Table IV shows that the uniqueness of JA3 fingerprints is high but the application coverage is below 50%, i.e., we cannot identify more than 50% of applications using the JA3 fingerprint alone. By adding the JA3+JA3S combination, we maintain the uniqueness and increase the coverage to 76.2% of applications. By adding SNI, we achieve 98.6% uniqueness and 96.8% coverage.

We can see that the best results in application identification are achieved by the combination of JA3+JA3S+SNI, which proves the previous results by [10]. Using JA4 and JA4S fingerprints with SNI does not increase the efficiency of identification.

B. Comparison of Efficiency of JA3 and JA4 fingerprints

When comparing the JA3 and JA4 fingerprints, the following observation was made:

1) The number of unique JA3 fingerprints is almost six times higher than JA4 fingerprints. This is expected because cipher suites and extensions are sorted in the JA4 fingerprints.
2) For certain applications JA4 reduces the number of unique fingerprints per application. For example, Chrome has only eight unique JA4 fingerprints, but 366 unique JA3 fingerprints.
3) When sorting extensions by JA4 fingerprints, we are unable to distinguish eight mobile applications that are distinguishable by the JA3 fingerprint.
4) The ALPN value in the JA4 fingerprint helps to distinguish some applications that cannot be distinguished by the JA3 fingerprint alone.
5) Comparison of the JA3S and JA4S fingerprints shows that both fingerprints have similar uniqueness and coverage. The higher efficiency of the JA4 over the JA3 is due to additional attributes such as protocol type and ALPN. This allows us to identify five applications in our datasets that would not be distinguished by the JA3S fingerprint.

As we can see, the combination of client and server fingerprints improves the ability to identify applications using JA3 and JA4 fingerprints. By preserving the order of TLS extensions in JA3 fingerprints, JA3 and JA3S fingerprints demonstrate superior performance in identifying applications compared to JA4 and JA4S fingerprints.

C. Combining TLS Attributes for Greater Efficiency

Analyzing the JA3 and JA4 fingerprints, we can see that neither is successful enough to uniquely identify an application. Even the highly unique JA3 hashes with an efficiency of 1.17 only cover 44.4% of the applications tested. To get better results, we need combine client and server fingerprints together with SNI.

The question is whether we can achieve higher efficiency by combining JA3 and JA4. Table V provides the answer. The upper part of the table shows the combination of JA3+JA4 and JA3S+JA4S. The first combination is similar to the JA3 fingerprint, the second is slightly better than the JA4S. The combination of both pairs gives the same efficiency as the JA3+JA3S pair.

Fingerprint type	Total	Unique	Covered apps	Efficiency
JA3+JA4	469	95.1%	46.0%	1.17
JA3S+JA4S	119	53.8%	42.9%	2.59
JA3+JA4+JA3S+JA4S	658	95.9%	77.8%	1.10
JA34	80	62.6%	33.3%	2.21
JA34S	92	43.5%	34.8%	3,15
JA34+JA34S	284	85.9%	71.4 %	1.32
JA34+JA34S+SNI	1020	97.8%	96.8%	1.02

TABLE V: Combination of TLS fingerprints and attributes

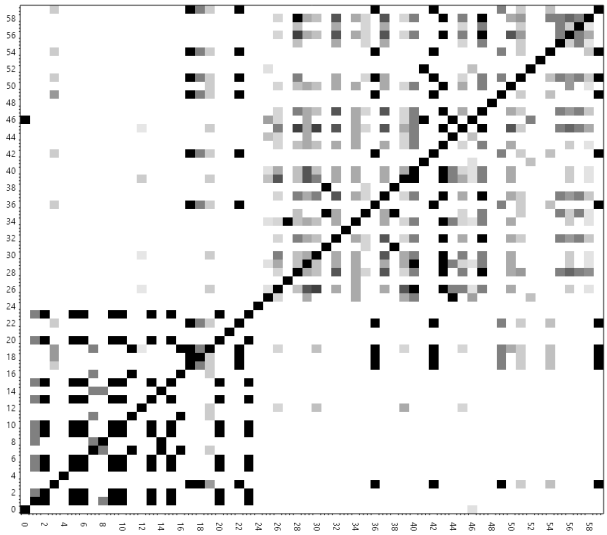
We then experimented with other TLS attributes based on the entropy as listed in Table II. First, we combined the JA3 cipher suites with the JA4 extensions, ALPN and protocol. We call this combination the JA34 fingerprint. We can see that the uniqueness is the same as for the JA4 fingerprint with similar coverage and efficiency.

We created a similar combination for server side TLS attributes: JA3 cipher suites, JA4 extensions, ALPN and protocol. We call this combination the JA34S fingerprint. This combination is even worse than JA3S and JA4S. By adding SNI to JA34 and JA34S we achieve the coverage and efficiency similar to JA3+JA3S+SNI.

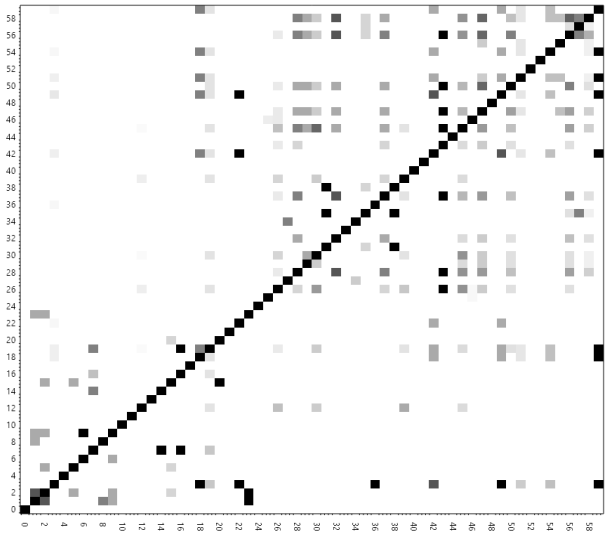
D. Shared Fingerprints over Applications

The distinctive characteristics of JA4 fingerprints across a range of desktop and mobile applications are graphically illustrated in Figure 4. The representation for JA3 is analogous and is not presented due to space limitations. Applications are assigned to the X and Y axes in the same order as listed in Figure 4d. Desktop applications are shown in the lower left corner, while mobile applications are depicted in the upper right corner. The individual figures illustrate the extent to which different applications share the fingerprints, with darker regions indicating a higher ratio of shared values between applications. Each column represents the ratio of shared fingerprints with the applications in the corresponding rows. In the optimal scenario, where all fingerprints are unique to their applications, the black cells should only be present on the diagonal, with the rest being white. The remainder of this section is devoted to a comprehensive examination of individual fingerprint schemes.

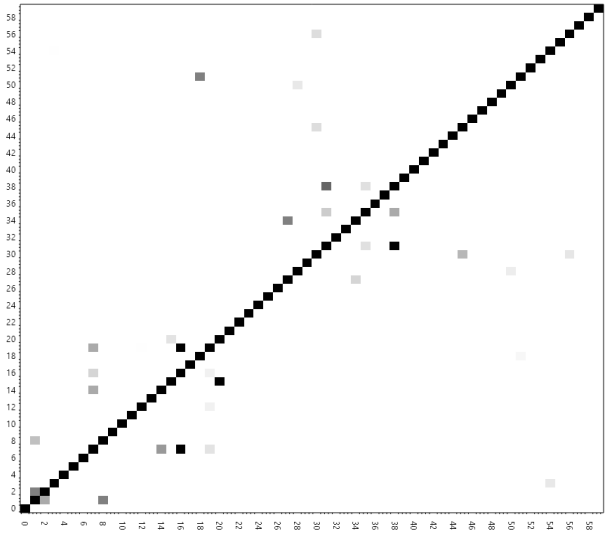
Figure 4a illustrates a fingerprint distribution in JA4. It highlights the previous results that the combination of JA4,



(a) JA4 fingerprints



(b) JA4 + JA4S fingerprints



(c) JA4 + JA4S + SNI fingerprints

Idx	Application	Idx	Application
0	DESKTOP-amazon-music	30	MOBILE-duolingo
1	DESKTOP-amplibraryagent	31	MOBILE-facebook
2	DESKTOP-appletv	32	MOBILE-foodora
3	DESKTOP-chrome	33	MOBILE-gmail
4	DESKTOP-firefox	34	MOBILE-idos
5	DESKTOP-fournet	35	MOBILE-instagram
6	DESKTOP-gamebar	36	MOBILE-linkedin
7	DESKTOP-hp	37	MOBILE-mapycz
8	DESKTOP-itunes	38	MOBILE-messenger
9	DESKTOP-maps	39	MOBILE-moje-vut
10	DESKTOP-messenger	40	MOBILE-mujvlak
11	DESKTOP-ms-teams	41	MOBILE-navlak-data
12	DESKTOP-msedge	42	MOBILE-netflix
13	DESKTOP-msmpeng	43	MOBILE-packeta
14	DESKTOP-omencmdcenter	44	MOBILE-reddit
15	DESKTOP-primevideo	45	MOBILE-regiojet
16	DESKTOP-runtimebroker	46	MOBILE-seznam
17	DESKTOP-sky-go	47	MOBILE-shazam
18	DESKTOP-skype	48	MOBILE-signal
19	DESKTOP-spotify	49	MOBILE-snapchat
20	DESKTOP-telegram	50	MOBILE-spotify
21	DESKTOP-tor	51	MOBILE-tiktok
22	DESKTOP-trello	52	MOBILE-tmobile
23	DESKTOP-whatsapp	53	MOBILE-tor
24	DESKTOP-zoom	54	MOBILE-twitter
25	MOBILE-accuweather	55	MOBILE-uc-browser
26	MOBILE-alza	56	MOBILE-viber
27	MOBILE-cestovne-poriadky	57	MOBILE-whatsup
28	MOBILE-discord	58	MOBILE-wolt
29	MOBILE-disneyplus	59	MOBILE-youtube

(d) List of applications in the dataset

Fig. 4: Shared TLS fingerprints by applications

JA4S and SNI reduces the number of shared fingerprints and increases the efficiency of application identification.

The data also demonstrates the ability to distinguish between desktop and mobile applications. In particular, there is significant sharing of fingerprints between desktop applications, while less sharing is observed between mobile applications. It is also interesting to note, that there are not many shared TLS fingerprints between desktop and mobile applications, see two separate rectangles.

The inclusion of SNI has significantly increased the number of applications. SNI provides such robust information that it almost eliminates the differences between the JA3 and JA4 versions of the fingerprints. Using the combinations with SNI, we identified unique fingerprints for 61 out of 63 applications. Furthermore, the number of overlapping fingerprints is further reduced, as shown in Figure 4c.

VI. Conclusion

A. Discussion

We compared the application identification capabilities of JA3 and JA4 using the mobile and desktop application datasets created. Both methods are designed to help identify the encrypted communicating applications by creating a database of known application fingerprints. In our experiment, we showed the different accuracy of these methods for application identification. In the case of JA3, about 95% of the hashes uniquely identify the application, compared to only 62% of the JA4 hashes. On the other hand, some applications are associated with many JA3 hashes, notably the Chrome web browser, because the JA3 hash depends on TLS values that vary with each initiated connection.

We also tested server-side JA3S and JA4S hashes. In this case, the unique identification of the application was less

than 50% for both hash types. The authors of the original JA3 method suggested a combination of JA3 and JA3S for application identification, assuming that Internet applications consist of unique client-server pairs. In our experiments, we found that this combination slightly improved the detection capabilities compared to using the JA3 hash value alone.

Finally, we also tested application identification based on a combination of client and server JA3/JA4 hashes supported by SNI analysis. Adding the SNI value provides additional information that improves application detection to 98%. This method would require not only a database of JA3/JA4 fingerprints, but also SNI values. However, both can be obtained using the automated techniques described in Section IV. Various combinations of JA3 and JA4 attributes do not give better results than JA3+JA3S+SNI, see Table V. The result can be improved by adding new attributes outside the TLS headers, e.g., packet size, flow length, etc.

B. Future Challenges

TLS fingerprinting is still considered as a promising method for identifying encrypted traffic. The main advantage of TLS fingerprinting is that it is a fast technique with low computational and memory requirements. The application can be easily identified after processing two packet headers: TLS Client Hello and Server Hello. In fact, we do not need to capture full packets but only extract selected fields from the TLS headers which is supported by extended IPFIX flow monitoring[13].

As mentioned in the previous text, there are publicly available tools that implement JA3, JA3S, and JA4+ hash computation. What is missing is a comprehensive and regularly updated database of TLS fingerprints for common network applications. In our work, we presented a tool to help build fingerprints for mobile and desktop applications. Progressive Web Apps (PWA) pose a challenge because these applications run within the web browser, so their fingerprint is not unique, but rather corresponds to the web browser being used. Identifying the specific PWA would require using more information from the communication, such as SNI, domain, and IP addresses of the contacted services for proper identification, which is a direction for further research.

Similarly, we identified many TLS connections to ad servers, tracking and web analytics services. As this *noise* overlaps with application sessions, it is necessary to filter out the fingerprints created by these noise connections. Currently, filtering is done by blacklisting tracking services, which does not cover all cases. Additional filtering is done by expert analysis of the SNI names in the client hello and destination IP addresses. Automated detection of noise sessions is therefore another area for future research. Accurate noise detection is particularly important for the training phase, i.e. building a fingerprint database.

Another area for future research is the study of TLS fingerprint spoofing in the context of malware detection. Frolov and Wustrow presented TLS attribute spoofing to evade filtering [8]. A similar technique can be abused by malware. Malware communicating over TLS will also attempt to disguise itself as normal traffic. Therefore, it would be useful to study this topic and show which TLS headers are susceptible to spoofing and how to prevent this attack. In the case of malware, it is possible to obtain a suitable dataset for TLS connection analysis from various sandbox analysis frameworks, some of which run as internet services, e.g. Triage, Anyrun. Malware fingerprint generation can then be automated in a similar way as presented in this paper for mobile and desktop applications.

[13]Implemented, for example, in Flowmon Probe.

ACKNOWLEDGMENT

Blinded version.

REFERENCES

[1] ENISA. Encrypted Traffic Analysis. Use Cases & Security Challenges. Technical report, European Union Agency for Network and Information Security (ENISA), November 2019.

[2] Petr Velan, Milan Čermák, Pavel Čeleda, and Martin Drašar. A Survey of Methods for Encrypted Traffic Classification and Analysis. *Netw.*, 25(5):355–374, September 2015.

[3] Gerard Draper-Gil., Arash Habibi Lashkari., Mohammad Saiful Islam Mamun., and Ali A. Ghorbani. Characterization of Encrypted and VPN Traffic using Time-related Features. In *Proceedings of the 2nd International Conference on Information Systems Security and Privacy - ICISSP*, pages 407–414. INSTICC, SciTePress, 2016.

[4] Mohammad A. Hoque, Benjamin Finley, Ashwin Rao, Abhishek Kumar, Pan Hui, Mostafa Ammar, and Sasu Tarkoma. Context-driven Encrypted Multimedia Traffic Classification on Mobile Devices. In *2022 IEEE International Conference on Pervasive Computing and Communications (PerCom)*, pages 54–64, 2022.

[5] R. Ocampo, A. Galis, C. Todd, and H. De Meer. Towards Context-Based Flow Classification. In *International Conference on Autonomic and Autonomous Systems (ICAS'06)*, pages 44–44, 2006.

[6] Blake Anderson, Subharthi Paul, and David McGrew. Deciphering malware's use of TLS (without decryption). *Journal of Computer Virology and Hacking Techniques*, 2018.

[7] Blake Anderson and David McGrew. TLS Beyond the Browser: Combining End Host and Network Data to Understand Application Behavior. In *Proceedings of the Internet Measurement Conference*, pages 379–392, 2019.

[8] Sergey Frolov and Eric Wustrow. The use of tls in censorship circumvention. In *The Proceedings of Network and Distributed Systems Security (NDSS) Symposium 2019*, pages 1–15, February 2019.

[9] Blake Anderson and David A. McGrew. Accurate TLS fingerprinting using destination context and knowledge bases. *CoRR*, abs/2009.01939, 2020.

[10] Petr Matoušek, Ivana Burgetová, Ondřej Ryšavý, and Malombe Victor. On Reliability of JA3 Hashes for Fingerprinting Mobile Applications. In *Digital Forensics and Cyber Crime, ICDF2C 2020*, volume 351 of *LNICST*, pages 1–22. Springer, 2021.

[11] T. Dierks and E. Rescorla. *The Transport Layer Security (TLS) Protocol Version 1.2*. IETF RFC 5246, August 2008.

[12] E. Rescorla. *The Transport Layer Security (TLS) Protocol Version 1.3*. IETF RFC 8446, August 2018.

[13] J. Iyengar and M. Thomson. *QUIC: A UDP-Based Multiplexed and Secure Transport*. IETF RFC 9000, May 2021.

[14] M. Thomson and S. Tuner. *Using TLS to Secure QUIC Transport*. IETF RFC 9001, May 2021.

[15] Shan Chen, Samuel Jero, Matthew Jagielski, Alexandra Boldyreva, and Cristina Nita-Rotaru. Secure Communication Channel Establishment: TLS 1.3 (over TCP Fast Open) versus QUIC. *Journal of Cryptology*, 34, 07 2021.

[16] D. Benjamin. *Applying Generate Random Extensions And Sustain Extensibility (GREASE) to TLS Extensibility*. IETF RFC 8701, Jan 2020.

[17] D. Eastlake. *Transport Layer Security (TLS) Extensions: Extension Definitions* . IETF RFC 6066, January 2011.

[18] S. Friedl, A. Popov, A. Langley, and E. Stephan. *Transport Layer Security (TLS) Application-Layer Protocol Negotiation Extension*. IETF RFC 7301, July 2014.