On Reliability of JA3 Hashes for Fingerprinting Mobile Applications

Petr Matoušek[1(✉)], Ivana Burgetová[1], Ondřej Ryšavý[1], and Malombe Victor[2]

[1] Brno University of Technology, Brno, Czech Republic
{matousp,burgetova,rysavy}@fit.vutbr.cz
[2] Strathmore University, Nairobi, Kenya
vmalombe@strathmore.edu

Abstract. In recent years, mobile communication has become more secure due to TLS encapsulation. TLS enhances user security by encrypting transmitted data, on the other hand it limits network monitoring and data capturing which is important for digital forensics. When observing mobile traffic today most transmissions are encapsulated by TLS. Encrypted packets causes traditional methods to be obsolete for device fingerprinting that require visibility of protocol headers of HTTP, IMAP, SMTP, IM, etc. As a reaction to data encryption, new methods like TLS fingerprinting have been researched. These methods observe TLS parameters which are exchanged in an open form before the establishment of a secure channel. TLS parameters can be used for identification of a sending application. Nevertheless, with the constant evolution of TLS protocol suites, it is not easy to create a unique and stable TLS fingerprint for forensic purposes. This paper presents experiments with JA3 hashes on mobile apps. We focus especially on the stability, reliability and uniqueness of JA3 fingerprints for digital forensics.

Keywords: Mobile application · TLS fingerprinting · Network forensics · JA3 hash · Encrypted communication

1 Introduction

With the disclosure of millions of private documents on Wikileaks in 2015, users and companies massively started to improve the security of transmitted data, especially against the interception. A high demand for security of transmitted data led to the adoption of encrypted techniques by many network protocols. Today, the majority of network applications and services support only encrypted communication encapsulated by Transport Layer Security (TLS) [10,26].

Encryption was also adopted by many mobile app vendors. Table 1 shows the structure of network protocols involved in mobile communication. Datasets 1 to 4 created by the authors of this study in 2018 show that the ratio of encrypted communication to unencrypted varies from 51,7 to 91,7%. You may notice a

© ICST Institute for Computer Sciences, Social Informatics and Telecommunications Engineering 2021
Published by Springer Nature Switzerland AG 2021. All Rights Reserved
S. Goel et al. (Eds.): ICDF2C 2020, LNICST 351, pp. 1–22, 2021.
https://doi.org/10.1007/978-3-030-68734-2_1

Table 1. Encrypted and unencrypted mobile communication in 2018 and 2019

	2018				2019
	Dataset1	Dataset2	Dataset3	Dataset4	Dataset5
Time	70 min	12 min	37 min	21 min	96 min
Size	452 MB	35 MB	423 MB	18 MB	610 MB
Packets	542.725	44.699	424.922	25.525	597.097
Protocol	**Encrypted Traffic**				
SSL/TLS	44,26%	86,03%	30,52%	80,10%	98,06%
UDP over 443					1,12%
IMAPS	0,65%	1,67%			
FB Zero	0,65%	0,12%	0,01%	0,13%	
QUIC	6,15%	3,90%	4,74%	8,07%	
OpenVPN			54,94%		
Total	*51,71%*	*91,72%*	*90,21%*	*88,30%*	*99,18%*
Protocol	**Unencrypted Traffic**				
HTTP	41,47%	1,65%	7,55%	2,12%	0,32%
DNS, mDNS	0,93%	1,78%	0,64%	2,41%	0,31%
DHCP	0,05%	0,13%	0,04%	0,26%	0,01%
ICMP/IGMP	0,36%	0,53%	0,14%	0,60%	0,07%
ARP	2,11%	1,01%	1,62%	3,17%	0,06%
Total	*44,92%*	*5,10%*	*9,99%*	*8,56%*	*0,77%*

presence of non-encrypted HTTP traffic. Especially HTTP headers, e.g., *User-Agent*, *Accept-Language*, *Accept-Charset*, have been largely used as an important data source for various fingerprinting techniques [11,17].

A year after our first experiments, we noticed that the ratio of encrypted communication increased to 99% which prevented the further use of traditional fingerprinting methods. As seen in Fig. 1, besides the encrypted TLS traffic transmitted over port 443, only Domain Name System (DNS) data remained open. It is a question for how long because of various attempts to encrypt DNS traffic using DNS over TLS (DoT) or DNS over HTTP (DoH) [15,16].

As a reaction to the encryption, researchers focused their activity on analysing behavior of encrypted communication in order to obtain meta data about the encrypted traffic. One research direction is focused on statistical analysis of the encrypted transmissions [9,29], the other direction deals with the features obtained from a TLS handshake that form the so called *TLS fingerprint* [3,6,18,24]. A popular implementation of TLS fingerprinting called *JA3 fingerprinting* was proposed by John B. Althouse, Jeff Atkinson and Josh Atkins in 2015[1]. This method is incorporated into multiple network monitoring and intrusion detection systems (IDS) like Flowmon, Bro, or Suricata, where it serves for the malware detection [4], identification of network applications [19], or black listing[2].

In our research, we focus on mobile devices, especially on the detection of mobile apps in network traffic. One of the features of mobile apps is that they regularly communicate over the Internet without explicit user interaction because of software updates, data synchronization, or checking on the remote status [24].

[1] See https://github.com/salesforce/ja3 [April 2020].
[2] See SSLBL project at https://sslbl.abuse.ch/ [April 2020].

This makes it possible to identify a mobile device based on a characteristic set of applications installed on the device [20]. Mobile apps can be identified from the captured TLS traffic using JA3 hashes (retrieved from the client's side of communication) or JA3S hashes (the server's side of communication).

However, there are important questions related to digital forensics: *Are these fingerprints reliable enough to identify a specific application? How stable are they? How can we create a unique fingerprint database of a mobile app?* The goal of this paper is to study the reliability of JA3 fingerprints on selected mobile apps, to demonstrate how unique fingerprints can be generated and to discuss the application of JA3 fingerprinting to the digital forensics.

1.1 Contribution

This work analyses the utilization of JA3 fingerprints for mobile apps identification. We primarily focus on the reliability and stability of JA3 fingerprints. Based on our experiments we found out that JA3 hashes alone are not sufficient for mobile app identification due to the high number of JA3 hashes common to multiple apps. By introducing additional TLS features like the JA3S server hash and Server Name Indication (SNI) extension, more accurate identification becomes possible. One of the contributions is a procedure describing the generation of unambiguous fingerprints for a given app. We also show the advantages and limits of the proposed technique. The second contribution is the generation of datasets with the captured traffic of mobile apps that contain the full TLS communication useful for further experiments with TLS fingerprinting. The third contribution includes a discussion of the stability and reliability of JA3 fingerprints which is important for digital forensics and mobile app identification.

1.2 Structure of the Text

The paper is structured as follows. Section 2 overviews recent works related to TLS fingerprinting and mobile apps identification. Section 3 gives the background of JA3 fingerprinting method and discusses its reliability for mobile apps identification. The main part of the paper is in Sect. 4 which describes how extended JA3 fingerprints of mobile apps are created and used for identification. Section 5 brings the results of our experiments and the evaluation of the proposed technique on the datasets. Section 6 discusses the application of TLS fingerprinting to digital forensics. The last section concludes our findings.

2 Related Work

TLS fingerprinting is not a new technique and its development is connected with the security research of Ivan Ristić who developed in 2008 an Apache module that passively fingerprinted connected clients based on cipher suites. Using this technique he created a signature base that identified many browsers and operating systems [1]. This technique was later applied on the identification of

HTTP clients [18] and implemented in IDS systems Bro and Surikata for passive detection.

Blake Anderson et al. in [4] studied millions of TLS encrypted flows and introduced a set of observable data features from TLS client and server hello messages like TLS version, TLS ciphers suites and TLS extensions that they used for malware detection. They also observed the server's certificate and the client's public key length, sequence of record lengths, times and types of TLS sessions. They identified cipher suites and extensions that were present in malware traffic and missing in normal traffic. The authors defined the TLS client configurations for the 18 malicious families. Similarly, they identified the TLS server configurations most visited by the 18 malicious families. They applied TLS features together with other features (flow data, inter-arrival times, byte distribution) to malware classification and achieved an accuracy from 96.7% to 98.2%. As demonstrated by their study, omitting TLS features led to a significantly worse performance.

Kotzias et al. [19] passively monitored the TLS and SSL connections from 2012 to 2015 and observed changes in TLS cipher suites and extensions offered by clients and accepted by servers. They also used client TLS fingerprinting with features similar to JA3 fingerprinting. From handshakes they omitted GREASE values. Using the captured data, they observed 7.3% fingerprint collisions in the TLS fingerprints. They also mapped fingerprints to a program or library and the version. One of their main results was the observation of the TLS fingerprints stability. They noticed that the maximum duration of a fingerprint seen in their databases was 1.235 days (3 years, 4 months). However, the median of duration 1 day and the mean was 158.8 days. They noticed some fingerprints that were seen very briefly and did not reappear later. They found out that 1,203 fingerprints of the 69,874 fingerprints were responsible for 21.75% of connections. Further, they analysed the vulnerability of TLS against various attacks which is a different direction comparing to our research. Their results related to the stability and collisions of TLS fingerprints was also observed in our experiments.

Another interesting approach published by Anderson and McGrew [3] combines the end host data with the network data in order to understand application behavior. This approach, however, requires an access to both the end hosts and the network. Their fingerprint database represented the real traffic generated by 24,000 hosts and having 471 million benign and millions of malware connections[3]. Using the end point data, the authors were able to associate the destination information with the end point data like the timestamp, endpoint ID, operating system and process name. They also observed that while GREASE values are generated randomly, their position is deterministic. Thus, instead of removing GREASE values, they set them to a fixed string `0a0a`. They also studied the similarity of TLS fingerprints using Levenshtein distance. Two TLS fingerprints were similar if their distance was less than or equal to 10% of the number of cipher suites, the extension types, and the extension values. The authors stated that the Levenshtein distance was an intuitive method for identifying close fingerprints. Especially TLS libraries often make minor adjustments

[3] Data capturing tools are available at https://github.com/cisco/mercury [April 2020].

to the default cipher suites or extensions between the minor version releases and more drastic changes between the major version releases. They also noticed that some TLS libraries change their default parameters to better suit the platform on which they are running. Another interesting point is the prevalence of application categories in the dataset where 37.1% connections belong to browsers, 19.3% to email applications, 17,2% to communication tools, 9% to the system, etc. Longevity of fingerprints like system libraries, tools osquery and DropBox, and browsers was 6 months or greater.

The above mentioned approaches worked mostly with common network traffic and network application. Another work closer to ours deals with TLS usage in Android Apps [24]. The authors analyzed the behavior of TLS in mobile platforms. They developed an Android app Lumen that was installed on a mobile device where it intercepted the TLS connections and gathered statistics about the traffic. Using Lumen, the authors observed how 7.258 apps use TLS. They analyzed handshakes with respect to the TLS API and the library that the app used. Their work was focused on apps security and TLS vulnerabilities. They showed that TLS libraries and OS API modified supported cipher suites across versions which caused changes in the TLS fingerprints. They also showed that each TLS library and OS version had a unique cipher suite lists. They built a database of fingerprints paired with corresponding OSes and libraries where they observed the influence of major and minor revisions of OS or TLS libraries on the fingerprint. Unfortunately, Lumen was not able to captured TLS handshakes which would have been useful to our research. Thus, we used an additional approach of how to obtain reliable TLS fingerprints of mobile apps.

The mobile application fingerprinting using characteristic traffic was considered by Stöber et al. [28]. They created a classifier that identified communicating applications based on the analysis of side-channel information such as timing and data volume. Mobile application fingerprinting has been tackled by machine learning techniques using timing and size of packets [31], which improved previous work presented in [30] that observed the traffic that was common among more than one apps. The method is applicable to encrypted traffic, which is used by most smartphone applications and relies only on information available from the side channel. The fingerprinting system was trained and tested on 110 most popular Android applications. The training was done automatically using the implemented application AppScanner. The significant feature of the method was that it analyzed the traffic represented as bursts. A burst was defined as a group of packets within TCP flow representing an interaction for a typical smartphone application that communicated using HTTPS protocol. Statistical features were then extracted for bursts and used for training random forests classifier. The method did not rely on any other source of information, e.g., DNS, TLS, IP addresses, etc. The achieved accuracy as presented by the authors was between 73 to 96% for the selected set of applications. Recently, the work was extended by [12] that used a semi-supervised method for both app recognition and detection of previously unseen apps.

Another line of research that considered mobile device identification is represented by Govindaraj, Verma and Gupta [14]. They proposed a methodology

for extracting and analyzing ads on mobile devices to retrieve user-specific information, reconstruct a user profile, and predict user identity. As the published results showed it was possible to identify a user in various settings even if he/she used multiple devices or different networks. Their work stemmed from the study by Castelluccia, Kafar and Tran [7] who demonstrated the possibility to infer user interests from targeted ads.

Our work uses previously published results and focuses on passive identification of mobile apps using JA3 fingerprints. It also observes traffic that is common to multiple apps and that should be excluded from fingerprinting. Based on the app, we employ JA3, JA3S, and Server Name Indication (SNI) features to accurately identify the unknown traffic that was sent by a mobile app. We are able to detect only apps that were previously learnt and stored in the fingerprinting database. Unlike some of the above mentioned approaches, our technique for mobile app detection is simple, fast and reliable. Its accuracy depends on the quality of learnt fingerprints.

3 How JA3 Fingerprinting Works

In this section we give a brief overview of principles of TLS communication and JA3 hashing that is necessary for understanding the proposed method.

Transport Layer Security (TLS) [10,26] is a transmission protocol that works on top of TCP where it provides privacy and data integrity for communicating applications. The protocol is composed of two parts: TLS Handshake Protocol and TLS Record Protocol. TLS Handshake Protocol negotiates the security parameters, e.g., version, methods for key exchange, encryption, authentication, and data integrity, secure channel options, etc. TLS handshake communication is not encrypted. The TLS Record Protocol encapsulates high-level protocol data and transmits encrypted packets. An example of TLS handshake is in Fig. 1.

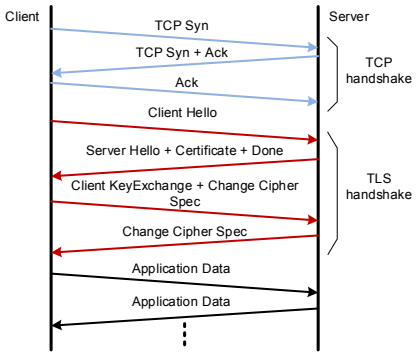

Fig. 1. Establishing TLS connection.

After opening a TCP connection by a three-way handshake, the TLS negotiates security parameters using TLS Client and Server Hello packets. The client

application offers a set of supported encryption and authentication methods using the TLS Client Hello. The TLS server processes these options and sends back options that are supported on the server side. The server can also include a server certificate to authenticate itself. After all security parameters are agreed on, the application data encapsulated by TLS Record Protocol are exchanged.

Most of TLS fingerprinting methods use the first packet sent by the client: the *Client Hello*. The Client Hello contains an imprint of TLS configuration of the client application that depends on the used TLS library and operating system. In this paper we study *JA3 fingerprint* that is computed as an MD5 hash from five TLS handshake fields: TLS Handshake version, Cipher suites, Extensions, Supported Groups (former Elliptic Curve), and Elliptic Curve point format, see Fig. 2. Some TLS fingerprinting implementations use different TLS fields, e.g., Kotzias et al. [19] omit the TLS version.

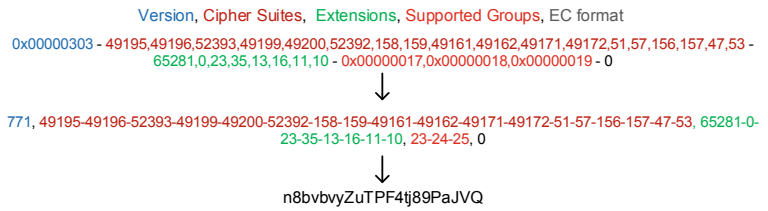

Fig. 2. Computing JA3 hash

The computation of JA3 fingerprint includes (i) the extraction of selected fields from TLS Hello packet, (ii) concatenation of extracted data in decimal format into one string, and (iii) application of MD5 hash algorithm on the string. The result is a 32-bit string in hexadecimal format. There are open implementations of JA3 fingerprinting available[4]. Unlike `nmap` or web browser fingerprinting methods which actively request the source device or application, JA3 fingerprinting uses a passive approach. The process of creating TLS fingerprints is fast because it only works with a TLS header. Common network monitoring and IDS tools implement the extraction of TLS parameters for analysis of the encrypted network traffic.

The application of TLS fingerprints to the identification of network apps requires TLS fingerprint values to be unique, accurate and stable. The following subsections describe aspects that limits the reliability of TLS fingerprints.

3.1 Impact of TLS Library on JA3 Hashes

A TLS fingerprint of a mobile app depends on the TLS library that was used during its implementation. There are plenty of TLS libraries available to developers, e.g., GnuTLS, Oracle JSSE, BSD LibreSSL, OpenSSL, or Mozilla NSS.

[4] See https://github.com/salesforce/ja3 or https://ja3er.com/ [April 2020].

When two applications are implemented using the same TLS library, their TLS fingerprints are usually the same. TLS fingerprints can also change with a new version of the app, TLS library, or operating system. This change can be caused by adding strong ciphers or removing weak ciphers, changing default parameters that better suit the running platform, or by adopting a new standard.

Table 2 shows JA3 hashes for popular web browsers: Mozilla Firefox v.73, Chrome v.80, and Opera v.66 under four operating systems: Linux Ubuntu, Windows 10, Kali Linux and Mac OS. We can see that Firefox has four unique JA3 fingerprints. Two of them are present in all tested operating systems. In case of Chrome and Opera, one JA3 fingerprint value corresponds to both browsers under all operating systems. These browsers were possibly compiled with the same TLS library. This experiment proves that TLS fingerprints change with the version and operating system. A similar experiment over a larger dataset was carried out by Razaghpanak et al. [24].

Table 2. JA3 hashes of common Web browsers

	Firefox				Chrome				Opera			
JA3 hash	Ubuntu	Win	Kali	MacOS	Ubuntu	Win	Kali	MacOS	Ubuntu	Win	Kali	MacOS
0e6f3c8f2b18f3011f1d6cbbdcfcbd65						x				x		
1344ed2e9d7d8e3e84e6ab655047ba32	x	x	x	x								
1f3c530fc35e41300422550c3c980e85							x	x	x	x	x	x
4863015f73b8332cf91cfa3a14a4893d		x										
5a291b49748c50adf1da70f8142d4cc4					x				x			
756094f51da8214018fbfba93211d59f	x	x	x	x								
a839cfeed30d55439b09de5f1b47fa3a					x	x	x	x	x	x	x	x
d889531a0389787425d5638caf6d84b3					x	x	x	x	x	x	x	
d90d517f72e9b8af9a8c1e2fe1fb2da8	x			x								

3.2 Randomized Values in TLS Extensions

In 2016, Google started to Generate Random Extensions And Sustain Extensibility (GREASE) values to TLS. This technique was adopted by IETF in January 2020 as RFC 8701 [5]. GREASE values are randomly generated numbers of cipher suites, extensions and supported groups present in TLS Hello packets. They prevent extensibility failures in TLS ecosystem. During TLS handshake, the responding side must ignore unknown values. Peers that do not ignore unknown values fail to inter-operate which means a bug in the implementation. Therefore, RFC 8701 adds GREASE values as a part of the list of cipher suites, extensions and supported groups to detect the invalid implementations.

When experimenting with Opera browser under Win 10 we noticed that the browser generates 155 unique JA3 fingerprints out of 207 TLS handshakes. By excluding GREASE values, the number of unique JA3 fingerprints decreased to four. The high number of JA3 fingerprints was caused by random GREASE values in TLS handshakes. Table 3 shows six JA3 fingerprints of Opera browser under Ubuntu with all extracted TLS values (the upper six lines). The last six lines presents TLS values without GREASE values. The brown values in the

upper table represent GREASE values as defined in [5]. When ignoring these values, the last four lines in the upper table would have the same JA3 hashes.

Table 3. JA3 hashes with and without GREASE values

List of Cipher Suites	List of Extensions	Supported Group	JA3 hash
49199-49200-49195-49196-52392-52393-49171-49161-49172-49162-156-157-47-53-49170-10	13172-0-5-10-11-13-65281-16-18	29-23-24-25	839868ad711dc55bde0d37a87f14740d
49199-49200-49195-49196-52392-52393-49171-49161-49172-49162-156-157-47-53-49170-10	13172-0-5-10-11-13-65281-16-18	29-23-24-25	839868ad711dc55bde0d37a87f14740d
56026-4865-4866-4867-49195-49199-49196-49200-52393-52392-49171-49172-156-157-47-53-10	60138-0-23-65281-10-11-35-16-5-13-18-51-45-43-27-19018-21	35466-29-23-24	ee972d7d47ec01a9cb9b04efb7346e32
60138-4865-4866-4867-49195-49199-49196-49200-52393-52392-49171-49172-156-157-47-53-10	39578-0-23-65281-10-11-35-16-5-13-18-51-45-43-27-56026-21	23130-29-23-24	cb4415a180704432d2e3f70f8dca5783
31354-4865-4866-4867-49195-49199-49196-49200-52393-52392-49171-49172-156-157-47-53-10	47802-0-23-65281-10-11-35-16-5-13-18-51-45-43-27-51914	43690-29-23-24	74a57a5f55ce2c9fa637b1f4567308b4
14906-4865-4866-4867-49195-49199-49196-49200-52393-52392-49171-49172-156-157-47-53-10	31354-0-23-65281-10-11-35-16-5-13-18-51-45-43-27-43690	56026-29-23-24	a10f93ffdc89d383db0f4437a0530569
49199-49200-49195-49196-52392-52393-49171-49161-49172-49162-156-157-47-53-49170-10	13172-0-5-10-11-13-16-18	29-23-24-25	5a291b49748c50adf1da70f8142d4cc4
49199-49200-49195-49196-52392-52393-49171-49161-49172-49162-156-157-47-53-49170-10	13172-0-5-10-11-13-16-18	29-23-24-25	5a291b49748c50adf1da70f8142d4cc4
4865-4866-4867-49195-49199-49196-49200-52393-52392-49171-49172-156-157-47-53-10	0-23-10-11-35-16-5-13-18-51-45-43-27	29-23-24	a839cfeed30d55439b09de5f1b47fa3a
4865-4866-4867-49195-49199-49196-49200-52393-52392-49171-49172-156-157-47-53-10	0-23-10-11-35-16-5-13-18-51-45-43-27	29-23-24	a839cfeed30d55439b09de5f1b47fa3a
4865-4866-4867-49195-49199-49196-49200-52393-52392-49171-49172-156-157-47-53-10	0-23-10-11-35-16-5-13-18-51-45-43-27	29-23-24	a839cfeed30d55439b09de5f1b47fa3a
4865-4866-4867-49195-49199-49196-49200-52393-52392-49171-49172-156-157-47-53-10	0-23-10-11-35-16-5-13-18-51-45-43-27	29-23-24	a839cfeed30d55439b09de5f1b47fa3a

In addition to GREASE values, it is also good to omit extension value 65281 from TLS fingerprinting. This value represents renegotiation option in TLS handshake [27], see red numbers in the list of extensions. Additionally, TLS Client Hello Padding Extension defined by RFC 7685 [23] can be omitted. The padding extension (value 21, depicted by green value in the table) is added by a client to make sure that the packet is of a desired size.

The above mentioned values can be added by a TLS client or server based on the local setting of a network connection which means they produce volatility of JA3 fingerprints. By removing these values we can increase the stability of JA3 fingerprints. Most of the JA3 implementations already ignore GREASE values.

3.3 Ads and Tracking Services in TLS Traffic

By observing TLS handshakes of mobile apps, we noticed that an app does not open the connection to its application server only, but it communicates with various sides without explicit user activity. These connections include ad servers, tracking services, or web analytic servers. This is typical especially for free apps that receive funding from ads providers. The destinations of these services are usually dynamic which means that each time the application is launched, it connects to a different site with a different TLS fingerprint. This causes problem for finding ground-truth communication for learning TLS fingerprints.

Dynamic behavior of ad connections is caused by mobile advertising auctions that redirect the app from the ad server to the content provider based on the results of an auction [21]. Since different mobile apps include the same ad, tracking or analytic plugins, the captured traffic of these apps contain the same TLS fingerprints. This extra traffic is called *a communication noise* or *ambiguous traffic* [31]. Table 4 shows TLS fingerprints obtained from Gmail app communication. There are five different JA3 fingerprints computed from captured TLS handshakes that were invoked by the Gmail app. Using the Server Name Indication (SNI) extension we can recognize the noise traffic directed to Google API (www.googleapis.com) and Google user content (googleusercontent.com). This traffic is not directly related to the app and can be found in communication

of other apps. The remaining fingerprint with SNI mail.google.com uniquely characterizes the Gmail app. Thus, it is important to exclude ad, tracking and analytic traffic from TLS fingerprinting. One solution how to remove the noise traffic is using available black lists of ad and tracking servers[5]. By comparing server names in SNI field of TLS handshake with names of ad servers in the black lists, we can partially clean up the captured TLS communication from the noise during the learning phase. Table 5 shows a percentage of the noise for selected mobile apps. Especially free apps include ad plugins producing such noise.

Table 4. JA3 hashes of Gmail App

SrcIP	DstIP	Server Name Indication	JA3 Fingerprint
10.0.2.15	172.217.23.193	ci5.googleusercontent.com	d5dcde95b8fa38b5062a128f7eff0737
10.0.2.15	172.217.23.225	ci3.googleusercontent.com	d5dcde95b8fa38b5062a128f7eff0737
10.0.2.15	172.217.23.229	mail.google.com	81d2604dcc31ff39cdddb6079692b0b0
10.0.2.15	216.58.201.106	www.googleapis.com	193c522402283ed9e84b8bb38137829f
10.0.2.15	216.58.201.106	www.googleapis.com	3d9a16cdc1b2a98f6046af1c833054b8
10.0.2.15	216.58.201.74	android.googleapis.com	ca75d9d90e40897206fa2a08d9100df0
10.0.2.15	216.58.201.97	ci4.googleusercontent.com	d5dcde95b8fa38b5062a128f7eff0737

3.4 Time Stability of JA3 Hashes

A very important issue related to the mobile apps fingerprinting is the stability of TLS fingerprints over time. We demonstrated, that a TLS fingerprint depends on TLS library and operating system, see Sect. 3.1. An update of the TLS library, adding new or excluding weaker ciphers can change the fingerprint. The longitudinal study of TLS fingerprints of Kotzias et al. [19] shows that the maximum duration of TLS fingerprints is 3 years and 4 months (median is 1 day, mean 158,8 days). If the app is not updated, it keeps its original TLS fingerprint.

Table 5. The number of TLS connections to Ad servers for selected Apps

App	All TLS handshakes	AD servers	Percentage
Accuweather	520	232	45%
BoomPlay Music	361	53	15%
Gmail	60	6	10%
Tor Browser	9	0	0%
Reddit	1892	840	44%
Muj vlak	451	195	43%
Viber	12	6	50%
Discord	24	0	0%
TitTok	203	20	10%
WhatsApp	243	30	12%
NextBike	444	39	9%
Facebook	178	13	7%
EquaBank	387	14	4%

[5] E.g., https://hosts-file.net/ad_servers.txt, https://pgl.yoyo.org/adservers/, or https://gitlab.com/ookangzheng/dbl-oisd-nl [April 2020].

The time instability means that for successful identification of mobile apps based on TLS fingerprints, we need to update the fingerprint database whenever a new version is released, otherwise the app will not be correctly identified. Nevertheless, our experiments show that the variability of TLS fingerprints is not so large and fingerprints stay the same even when an OS is updated.

4 Identification of Mobile Apps Using TLS Fingerprinting

This section describes how TLS fingerprints are created (learning phase) and used for mobile apps identification (detection phase).

4.1 Learning TLS Fingerprints

As mentioned above, the crucial task for mobile apps identification using TLS fingerprints is to create a reliable fingerprint database with unambiguous entries. Even if an app is running in the controlled environment like the Android Virtual Studio, the captured traffic contains a mixture of app traffic with communication of OS, pre-installed apps and plugins that are common to multiple apps. Here, we introduced a technique, how to clean up the captured traffic in order to receive only TLS handshake related to the given app, see Fig. 3.

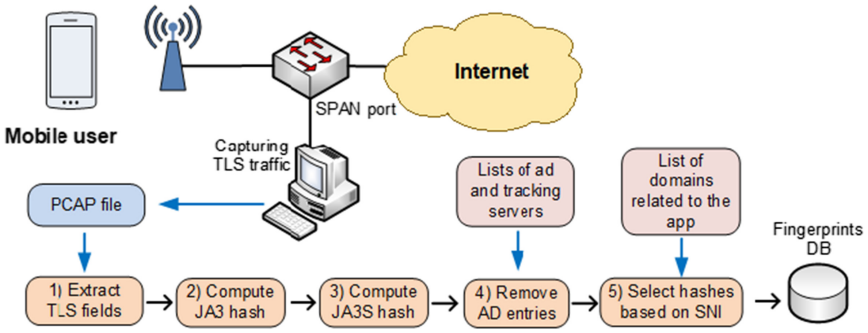

Fig. 3. Creating TLS fingerprints

First, we need to launch an app communication on the mobile device. We made experiments both with virtual devices running on the Android Virtual Studio (datasets MA2, MA3) and on real devices (datasets MA1, MA4). When using the virtual environment, we can capture network traffic on the interface connected to the virtual environment. However, there can also be communication of virtual OS and other applications installed on the system. When using real smart phones, we can create a WiFi connection only for this device and capture traffic on the WiFi interface. Fingerprint creation include the following steps:

1. Extract TLS Client Hello packets where `tls.handshake.type==1` and obtain the following data: source and destination IP address, source and destination port, TLS handshake type (client or server hello), SNI, a list of TLS cipher suites, extensions, supported groups, and EC point format. Exclude GREASE values, padding and renegotiation options, see Sect. 3.
2. Compute JA3 hash using TLS values in TLS Client Hello packet as explained in Sect. 3. Apply MD5 hash function on TLS version, a list of cipher suites, list of extensions, supported groups and EC point format. JA3 hash uses MD5 function with 32-bit output in hexadecimal format.
3. Compute JA3S hash using TLS values in a Server Hello packet. Link the JA3S hash with the JA3 hash using IP addresses and ports. The JA3S hash is a MD5 digest of the TLS version, cipher suite and extensions only.
4. Based on the list of ad servers and tracking servers, remove TLS fingerprints where SNI matches any domain name present in these lists.
5. From candidate TLS fingerprints select only those fingerprints that are related to the app based on matching SNI field with keywords related to the app. Keywords can be obtained manually or using the tools Lumen or AppVersion that show information about the app like domain names. An example of keywords that match the app SNI names is listed in Table 6. In most cases, keywords also include the app name. Formally, the keyword is the maximum common substring of all SNI names related to the app.

Table 6. Example of app keywords

Application	Keyword (s)	SNI
BoomPlay Music	boomplaymusic	source.boomplaymusic.com, android.boomplaymusic.com
Accuweather	accuweather, accu-weather	api.accuweather.com, cms.accuweather.com, vortex.accuweather.com
Viber	viber	content.cdn.viber.com
Discord	discord	best.discord.media, discordapp.com, dl.discordapp.net, gateway.discord.gg, ...
Mobilni Banka	mojebanka.cz, mobilnibanka.cz	www.mojebanka.cz, wa.mojebanka
KB klic	kb.cz	login.kb.cz, www.kb.cz
Nextbike	nextbike.net	api.nextbike.net, maps.nextbike.net, my.nextbike.net, static.nextbike.net, ...
EquaBank CZ	equa.cz, equamobile.cz	acs.equa.cz, ma.equamobile.cz
TitTok	tiktok	abtest-va-tiktok.byteoversea.com, mon.tiktokv.com
Duolingo	duolingo	android-api.duolingo.com, duolingo-leaderboard-prod.duolingo.com, ...
Youtube	youtube	www.youtube.com, youtubei.googleapis.com
Google Calendar	calendarsync	calendarsync-pa.googleapis.com
WhatsApp	whatsapp	media-prgl-l.cdn.whatsapp.net, pps.whatsapp.net, static.whatsapp.net
Gmail	mail.google.com, inbox.google.com	mail.google.com, inbox.google.com
Muj vlak	timetable.cz	ipws2.timetable.cz

This procedure does not guarantee uniqueness of the obtained fingerprints which is essential for successful detection. When analysing JA3 fingerprints learnt from our datasets, we noticed that there were 30 distinct fingerprints, however, many of them belonged to multiple applications. Only 21 JA3 hashes were assigned unambiguously reaching uniqueness of 70%. Thus, we added JA3S fingerprint to a feature set and obtained 122 distinct combinations with 114 unique fingerprints related to only one app. Remember that one app can have a set of unique fingerprints. Combination of JA3+JA3S increased uniqueness to 93,44%

but there were still several JA3+JA3S combinations that belonged to more than one app. After adding SNI to a feature set we received 154 distinct combinations with 153 combinations related to only one app. The results are given in Table 7.

Table 7. Uniqueness of features in the TLS fingerprint

Feature	Distinct items	Unique items	Uniqueness
JA3	30	21	70,00%
JA3+JA3S	122	114	93,44%
JA3+JA3S+SNI	154	153	99,35%

The number of unique fingerprints does not express, how many applications from our MA datasets we can cover with these fingerprints. We further analysed the sets of unique fingerprints to reveal this information. When using JA3 hashes only, we can uniquely identify only 33,33% of apps from our dataset. Remaining apps cannot be identified with JA3 hashes only (66,67%), because there are no unique JA3 hashes for these apps in our datasets. When adding JA3S hashes, we are able to cover 79,17% of apps from our datasets. Adding SNI to a feature set increase the coverage of apps to 91,67%. Using all three features, we were not able to cover 2 of 24 apps - Messenger and Telegram.

It seems that the combination of JA3, JA3S and SNI provides unique and reliable TLS fingerprints of mobile apps. This statements is not always true. It depends on the function of the mobile app. Most mobile apps communicate only with a limited number of servers related to the app. Some apps, for instance web browsers, communicate with an open set of destinations based on user activity. These apps cannot be identified by JA3S and SNI because these features depend on the destination which changes by each connection. Also for applications that connect to servers with random or anonymized domain names, only JA3 hash can be employed for app identification. Interestingly, the JA3 hash of Tor app was unambiguous and sufficient for successful identification. Most of the apps, however, require the full combination of JA3, JA3S and SNI features.

4.2 Detection of Mobile Applications Using TLS Fingerprints

The above written procedure describes a generation of TLS fingerprints from captured TLS traffic. The process includes TLS data pre-processing and refinement that produces a unique TLS fingerprint composed of JA3 hash only, combination of JA3+JA3S or JA3+JA3S+SNI. Having such fingerprints, we can monitor unknown network traffic, retrieve selected values from the TLS Hello packets, and compute JA3 and JA3S hashes. By comparison with the fingerprint database, we can identify an app that initiated this communication.

4.3 Stability and Reliability of TLS Fingerprinting

As mentioned in Sect. 3, the stability of TLS fingerprints of a mobile app depends on an app version, TLS library, and operating system. When using JA3S hashes,

it also depends on the server version and its TLS library. This means that we have to update our fingerprint database whenever a new version of the app is released. The fingerprint can be generated using the procedure described in Sect. 4.1. In some cases, a new version may keep the same fingerprint as the previous one. Fingerprint stability is demonstrated on experiments with dataset MA3 where we observed TLS fingerprints of four apps on Android 7.1, 8.1 and 9. The results are presented in Table 8. The first column represents the number of unique values of JA3, JA3S and SNI in the TLS fingerprint of a given mobile app under Android 7. Columns Android 8.1 and 9 show the number of features that were added or missing in comparison to the previous version. We can see that SNI for CP app and Mujvlak are stable across versions. JA3S hash of CP app was changed when migrating from version 7 to 8 but it stayed unchanged to version 9. Adding new values does not negate stability of the fingerprint because the original fingerprint can still identify the app. If there are more additions, it may happen that the fingerprint of the older version would not match newly added features (false negative). However, when updating the fingerprint database by a new fingerprint, the accuracy of the identification is preserved.

Table 8. Stability of TLS fingerprints over OS version

Mobile App	Android 7	Android 8.1		Android 9	
Feature	present	added	missing	added	missing
CP JA3	1	1	1	0	0
CP JA3S	2	2	2	0	0
CP SNI	2	0	0	0	0
Mujvlak JA3	1	1	1	0	0
Mujvlak JA3S	1	0	0	0	0
Mujvlak SNI	1	0	0	0	0
Reddit JA3	1	2	1	0	0
Reddit JA3S	1	2	1	0	0
Reddit SNI	9	3	2	4	0
Seznam CZ JA3	3	3	3	2	1
Seznam CZ JA3S	11	0	0	2	0
Seznam CZ SNI	12	0	1	1	0

5 Experiments

This section describes our experiments with fingerprinting mobile apps. First, we describe our datasets and then achieved results of mobile apps identification.

5.1 Datasets

This section introduces datasets used in the experiments. First we observed available datasets with the mobile traffic. ReCon dataset[6] [25] was created to observe leakage of personal identifiers through mobile communication. The dataset contains HTTP(s) logs of 512 mobile apps. The logs do not contain TLS headers that are important for TLS fingerprinting. However, for a given mobile app, we can

[6] See https://recon.meddle.mobi/appversions/ [April 2020].

extract a list of sites the app usually connects to. For example, for `accuweather` app, we get fonts.googleapis.com or ssl.google-analytics.com (noise servers), and vortex.accuweather.com or accuwxturbo.accu-weather.com (app related domain names). These domain names can be traced in SNI extension during the TLS analysis which is important for creating unambiguous fingerprints.

An interesting mobile apps dataset is Panoptispy[7] [22] that was created to study media permissions and leaks from Android apps. The dataset consists of network traffic that have instances of media in an HTTP requests body. Besides dumps of HTTP requests, it contains a list of apps with a package name, version, app name and app md5. However, this is not sufficient for TLS fingerprinting.

Andrubis and Cross Platform datasets mentioned in [12] were not located by the authors of this paper. Nevertheless, we explored Mirage dataset[8] [2] which contains mobile app traffic for ground-truth evaluation. The captured traffic is stored in a JSON format and contains the bi-flows with src/dst ports, number of bytes, inter-arrival times, TCP window size, L4 raw data, and various statistics. Since the TLS header is hidden in byte-wise raw L4 payload, it is not easy to extract TLS values that are interesting for our research. However, we plan to use this data for a ground-truth evaluation of our method presented in the paper.

For our experiments we created our own dataset that contains communication of selected mobile apps, see Table 9. We ran the app five to ten times on the same platform in order to receive a representative sample of the traffic. In case of MA3 database we ran each app 10 to 20 times on the three Android's versions with and without cache (after restarting a device) in order to observe the impact on the temporary data saved in the cache. That is why dataset MA3 is larger.

Table 9. Mobile apps communication dataset

Dataset	Apps	Device	OS	PCAP (MB)	Packets	TLS Handshakes	List of Apps
Web browsers (WB)	3	PC	Linux, MacOS, Win 10	193	224.717	2.621	Chrome, Firefox, Opera
Mobile Apps I (MA1)	5	Sony, Huawei	Android 9, 7.0	2	5.700	79	Discord, Messenger, Slack, Telegram, WhatsApp
Mobile Apps II (MA2)	4	Android Studio	Android 6	35	50.820	595	Accuweather, Gmail, Tor, Viber
Mobile Apps III (MA3)	4	Android Studio	Android 7.1, 8,1, 9	827	642.919	3.180.345	Cestovne Poriadky, Mujvlak, Reddit, Seznam
Mobile Apps IV (MA4)	14	Tecno	Android 5.1	446	578.812	5.308	Boomplay Music, Chrome, EquaBank, Facebook, ...
Total	*30*			*1504*	*1.502.968*	*3.188.948*	

Even if using a small number of apps, our datasets are sufficient for studying typical features of TLS mobile apps fingerprints: uniqueness, stability and reliability, and for training and detection of mobile apps. These datasets are available in PCAP format in github[9] and contain five parts:

Web Browsers (WB). The first dataset consists of TLS communication of web browsers Chrome v80, Firefox 68.2, Firefox 73.0, Firefox 70.0, Opera 66.0 and

[7] See https://recon.meddle/mobi/panoptispy [April 2020].

[8] See https://ieee-dataport.org/open-access/mirage-mobile-app-traffic-capture-and-ground-truth-creation [April 2020].

[9] See https://github.com/matousp/ja3s-fingerprinting [July 2020].

Opera 67.0. These browsers were running under four different operating systems: Kali Linux, Mac OS, Windows 10 and Linux Ubuntu. During experiments we requested 10 different URLs. We created TLS fingerprints for all browsers based on TLS handshakes related to requested URLs. The dataset contains 2.621 TLS handshakes. It does not contain mobile traffic. It was used to observe an impact of TLS libraries on JA3 fingerprints, see Sect. 3.1.

Mobile Apps I (MA1). The second dataset includes five mobile applications: Discord v16.3, Messenger v253.0, Slack v20.03, Telegram v6.0 and WhatsApp v2.20. The applications were installed on two mobile devices: Sony Xperia X71 Compact with Android 9 (API level 28) and Huawei P9 with Android 7 and EMUI 5.0.1 (API level 24). Devices were connected to a PC and TLS data captured using `tshark`. To make sure that the packets include initial handshake, the tested application were restarted using ADB commands. The dataset contains 79 TLS handshakes.

Mobile Apps II (MA2). The third dataset includes communication of four mobile applications: Accuweather, Gmail, Tor and Viber. For tests, we used Android Emulator which is a part of Android Studio. In the Android Emulator, we created two virtual devices: Google Pixel C with Android 8.1 and Google Nexus 10 with Android 6.0. Using ADB interface we installed the above mentioned mobile applications on the virtual device and simulated user behavior using the command-line tool *Monkey*. The Monkey emulates user behavior on a given app, thus the captured traffic is initiated by this app only. An example of emulating Viber app on the virtual device is below. The dataset contains 595 TLS handshakes.

```
$adb shell monkey -p viber -v 500  # emulate 500 events on package viber
```

Mobile Apps III (MA3). This dataset includes communication of the following mobile applications: Cestovne Poriadky (Time Table), Muj vlak (My train), Reddit and Seznam. TLS fingerprints of these apps were obtained using Virtual Box where these apps were installed. The apps were tested on Android version 7.1, 8.1 and 9. On each Android system, the app was repeatedly launched and the communication captured. We also observed if the application cache has influence on the communication, so each app was running twenty times without cache and twenty times with cache on each system. Together, we obtained 3.180.245 TLS handshakes.

Mobile Apps IV (MA4). The last dataset was focused on a variety of mobile apps installed on a real device Tecno J8 with Android 6.1. The dataset includes the following apps: BoomPlay Music, Chrome Browser, Equa Bank app, Facebook app, Gmail app, Google calender, KB klic, Messenger, Mobilni Banka app, NextBike, Telegram, TikTok, WhatsApp and Youtube app. Each app was running five times on the restarted device so that the captured communication

corresponds to a typical usage. We extracted 5.308 TLS handshakes from the captured traffic.

5.2 Results

We evaluated our TLS fingerprinting method using datasets MA3 and MA4 as they contain multiple runs for each application and can be divided into training and testing sets. In order to get the balanced sets we used reduced dataset MA3 (11 runs per app). We observed communication of 16 apps captured in distinguished time windows. We selected one run of each application for testing and the others we used for training. For training, we used the procedure described in Sect. 4.1. The testing set contained 244 TLS handshakes in total. These handshakes were classified as belonging to some application or unknown traffic.

Table 10. Detection of mobile apps based on JA3 hash

		Real values																
		A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	X
Predicted values	A	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	B	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	C	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	14
	D	0	0	0	4	0	0	0	0	0	0	0	0	0	0	0	0	12
	E	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	F	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	G	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	H	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	I	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	J	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	K	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0
	L	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	13
	M	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	N	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	O	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	P	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	X	5	5	0	0	1	6	1	4	9	1	1	0	13	6	1	1	139

Table 10 shows the confusion matrix of TLS handshakes classification based on JA3 hash only. Letters A to P represent mobile apps as follows: BoomPlay Music (A), EquaBank (B), Facebook (C), Gmail (D), Google Calendar (E), Chrome App (F), KB Klic (G), Mobilni Banka (H), NextBike (I), TikTok (J), WhatsApp (K), Youtube (L), Seznam CZ (M), Reddit (N), Muj vlak (O) and Cestovne Poriadky (P). Letter X describes unknown traffic. The rows contain predicted values, columns represent real values.

We can see the limits of JA3 fingerprinting that works well with apps C (Facebook), D (Gmail), K (WhatsApp) and L (Youtube) but other apps have JA3 hashes same as unknown traffic (X class). By adding JA3S hash to TLS fingerprint, the number of correctly classified apps increases, see Table 11. However, there is still a high number of false positives (column X). Table 12 presents classification results for three features JA3+JAS3+SNI. We can see that the classification is more accurate when using all these features with the exception of apps Chrome (F) and Youtube (L). Table 13 shows the accuracy, precision and

recall of classification. JA3 hash is reliable only for specific apps and produces many false negatives (row X). JA3+JA3S classification has the comparable accuracy but better recall. This means that it produces a lot of false positives. The best result shows a combination of JA3+JA3S+SNI. It also places some samples into the X (unknown app) category, however, this can be improved by extending a list of keywords and inserting additional SNIs into the fingerprint database.

Table 11. Detection of mobile apps based JA3+JA3S

		Real values																
		A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	X
Predicted values	A	5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	B	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	C	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	13
	D	0	0	0	4	0	0	0	0	0	0	0	0	0	0	0	0	18
	E	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	F	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	4
	G	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	H	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
	I	0	0	0	0	0	0	0	0	6	0	0	0	0	0	0	0	0
	J	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	K	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0
	L	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	10
	M	0	0	0	0	0	0	0	0	0	0	0	0	10	0	0	0	2
	N	0	0	0	0	0	0	0	0	0	0	0	0	0	6	0	0	0
	O	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
	P	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
	X	0	2	0	0	1	4	1	3	3	1	2	0	3	0	0	0	131

Table 12. Detection of mobile apps based on JA3+JA3S+SNI

		Real values																
		A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	X
Predicted values	A	5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	B	0	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	C	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	8
	D	0	0	0	4	0	0	0	0	0	0	0	0	0	0	0	0	0
	E	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
	F	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	G	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
	H	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0
	I	0	0	0	0	0	0	0	0	7	0	0	0	0	0	0	0	0
	J	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	K	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0
	L	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
	M	0	0	0	0	0	0	0	0	0	0	0	0	12	0	0	0	0
	N	0	0	0	0	0	0	0	0	0	0	0	0	0	6	0	0	0
	O	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
	P	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
	X	0	1	0	0	0	6	0	1	2	1	1	1	1	0	0	0	170

Table 13. Evaluation of combination of TLS features

	Total items	Accuracy	Precision	Recall
JA3	244	61,89%	23,53%	18,18%
JA3+JA3S	244	72,54%	49,46%	69,70%
JA3+JA3S+SNI	244	90,98%	86,67%	78,79%

6 Application to Digital Forensics

The identification of smartphone apps can be applied to digital forensics as a complementary method to obtain forensically valuable information. First, the background traffic of installed apps can be analyzed to identify a communicating device [28]. This can distinguish a smartphone model from different vendors based on a pre-installed set of apps [13] and the background traffic. Next, by the identification of communicating applications, we can observe user-specific information, habits and interests as well [8].

Most digital investigations that include mobile device analysis use a logical extraction to access the existing files such as call history, text messages, web browsing history, pictures and other files available on the smartphone. However, logical extraction requires the possession of a device and bypassing the password. With smartphones better protected against the unauthorized access, the passive monitoring of their activities stands for the complementary data source for forensic analysis. The possibility to identify the mobile app, and therefore the device or even the user of the device is applicable in the following scenarios:

- Forensic analysis. LEAs may send a preservation order to the ISP to collect the communication for a specific device. The captured information can be used for learning about the activities of a suspect at different points in time. Creating a profile of the suspect and correlating identified activities with the information obtained from the other sources can bring an important insight to the case being investigated. Even if most of the app traffic is encrypted, the presented method can detect installed applications. The presence and usage of a specific application at a given point of time may reveal the intention of the suspect. Later, after the device is physically available to an investigator the information obtained from the monitoring phase can be corroborated with the findings of the logical acquisition outcomes to support the reliability of the evidence. Criminals aware of secrecy provided by IM apps can use them for communication to protect against traditional call record analysis [32]. However, if we can identify the activity of mobile apps, the traffic generated by the communicating IM hosts can be used to record communication between suspects. Also, posts published under the anonymous social network account can be revealed by comparing the time of the public posts with the time of the actions as inferred by the application usage aiding to hate crime investigations.
- Intelligence operations. The agencies may be able to trace certain individuals on basis of tracking the communication characteristic of the apps installed on their smartphones. To be feasible, the amount of information that needs to be collected and processed has to be limited. For instance, NetFlow-based monitoring is considered as suitable technique for this purpose [33]. The advantage of TLS fingerprinting can be applied at massive scale. Adding TLS fingerprinting to existing NetFlow monitoring requires to include TLS fingerprints to NetFlow records, which many existing monitoring solutions already provide for detection of security threats that use encrypted communication.

The presented cases consider the scenarios when the method is applied as a part of legal investigation done under strict law requirements applied to data collection and analysis. Unfortunately, as for most communication monitoring techniques collecting sensitive information possibly revealing user privacy, also this method can be misused. For instance, the totalitarian states can easily adapt the method to block selected apps and because it is computational inexpensive they can apply it on a large scale. Avoiding this situation requires the modification of TLS parameters of applications to make their JA3 hashes indistinguishable or intractable, e.g., by adding randomly generated parameters in the extension list and avoiding the use of SNI field.

7 Conclusion

Mobile apps fingerprinting can be considered a practical method with potential applications in digital forensics. In this paper, we have presented a study on the reliability of JA3-based methods for mobile apps identification. The advantage of this method is that it only depends on the TLS handshake information that is obtained during the establishment of the secure channel.

We have shown that using JA3 is not sufficient for the accurate identification of mobile apps. More reliable results are obtained by a combination of JA3, JA3S and SNI features that can be easily computed from the TLS handshake messages. We have also considered the issues of TLS fingerprint volatility. Based on our experiments the variability of TLS fingerprints is not so large. Also, when a new major version of the application is released, it is not difficult to obtain a new fingerprint in the virtual environment and update the fingerprint database.

The presented results are valid for existing TLS versions that provide access to the source information necessary for computing the fingerprints. However, ongoing work on TLS protocol suggests an increase of user privacy by hiding some of currently visible fields, e.g., SNI[10], or even the encryption of TLS Hello message. Addressing these emerging challenges is a topic for our future work.

Acknowledgement. This work was supported by project "Integrated platform for analysis of digital data from security incidents", 2017–2020, No. VI20172020062, granted by Ministry of Interior of the Czech Republic. The authors would like to give thanks to students Matej Meluš, Radovan Babic and Alberts Saulitis who participated in the generation of datasets for the research experiments.

References

1. Abel, R.: SSL/TLS fingerprint tampering jumps from thousands to billions. SC Magazine (2019)
2. Aceto, G., Ciuonzo, D., Montieri, A., Persico, V., Pescapé, A.: MIRAGE: mobile-app traffic capture and ground-truth creation. In: 2019 4th International Conference on Computing, Communications and Security (ICCCS), pp. 1–8 (2019)

[10] See Internet Draft at https://tools.ietf.org/html/draft-ietf-tls-esni-06 [March 2020].

3. Anderson, B., McGrew, D.: TLS Beyond the browser: combining end host and network data to understand application behavior. In: Proceedings of the Internet Measurement Conference, pp. 379–392 (2019)
4. Anderson, B., Paul, S., McGrew, D.: Deciphering malware's use of TLS (without decryption). J. Comput. Virol. Hacking Tech. (2018)
5. Benjamin, D.: Applying Generate Random Extensions And Sustain Extensibility (GREASE) to TLS Extensibility. IETF RFC 8701, January 2020
6. Böttinger, K., Schuster, D., Eckert, C.: Detecting fingerprinted data in TLS traffic. In: Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security, pp. 633–638. ASIA CCS 2015. Association for Computing Machinery, New York (2015)
7. Castelluccia, C., Kaafar, M.A., Tran, M.D.: Betrayed by your ads!. In: Fischer-Hübner, S., Wright, M. (eds.) Privacy Enhancing Technologies, pp. 1–17. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-31680-7_1
8. Conti, M., Mancini, L.V., Spolaor, R., Verde, N.V.: Can't you hear me knocking: identification of user actions on android apps via traffic analysis. In: Proceedings of the 5th ACM Conference on Data and Application Security and Privacy, CODASPY 2015, pp. 297–304. New York, NY, USA (2015)
9. Danezis, G.: Traffic Analysis of the HTTP Protocol over TLS (2009). https://pdfs.semanticscholar.org/9d75/9184cdc524624fe551b9fc15de9a4cd199fa.pdf
10. Dierks, T., Rescorla, E.: The Transport Layer Security (TLS) Protocol Version 1.2. IETF RFC 5246, August 2008
11. Eckersley, P.: How unique is your web browser? In: Proceedings of the 10th International Conference on Privacy Enhancing Technologies, PETS 2010, pp. 1–18. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-14527-8_1
12. van Ede, T., et al.: FlowPrint: semi-supervised mobile-app fingerprinting on encrypted network traffic. In: NDSS (2020)
13. Gamba, J., Rashed, M., Razaghpanah, A., Tapiador, J., Vallina-Rodriguez, N.: An analysis of pre-installed android software. In: 41st IEEE Symposium on Security and Privacy. IEEE (2020)
14. Govindaraj, J., Verma, R., Gupta, G.: Analyzing mobile device ads to identify users. DigitalForensics 2016. IAICT, vol. 484, pp. 107–126. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-46279-0_6
15. Hoffman, P., McManus, P.: DNS Queries over HTTPS. RFC 8484, October 2018
16. Hu, Z., Zhu, L., Heidemann, J., Mankin, A., Wessels, D., Hoffman, P.: Specification for DNS over Transport Layer Security (TLS). IETF RFC 7858, May 2016
17. Hupperich, T., Maiorca, D., Kührer, M., Holz, T., Giacinto, G.: On the robustness of mobile device fingerprinting: can mobile users escape modern web-tracking mechanisms? In: Proceedings of the 31st ACSAC, pp. 191–200. New York, USA (2015)
18. Husák, M., Čermák, M., Jirsík, T., Čeleda, P.: Https traffic analysis and client identification using passive SSL/TLS fingerprinting. EURASIP J. Inf. Secur. (2016)
19. Kotzias, P., Razaghpanah, A., Amann, J., Paterson, K.G., Vallina-Rodriguez, N., Caballero, J.: Coming of age: a longitudinal study of TLS deployment. In: Proceedings of the Internet Measurement Conference 2018, pp. 415–428 (2018)
20. Kurtz, A., Gascon, H., Becker, T., Rieck, K., Freiling, F.: Fingerprinting mobile devices using personalized configurations. Proc. Privacy Enhancing Technol. **1**, 4–19 (2016)
21. Kwakyi, G.: How Do Mobile Advertising Auction Dynamics Work? Incipia Blog (2018). https://incipia.co/post/app-marketing/how-do-mobile-advertising-auction-dynamics-work/

22. Pan, E., Ren, J., Lindorfer, M., Wilson, C., Choffnes, D.: Panoptispy: characterizing audio and video exfiltration from Android applications. Proc. Privacy Enhancing Technol. 33–50 (2018)
23. Benjamin, D.: A Transport Layer Security (TLS) ClientHello Padding Extension. IETF RFC 7685, October 2015
24. Razaghpanah, A., Niaki, A.A., Vallina-Rodriguez, N., Sundaresan, S., Amann, J., Gill, P.: Studying TLS usage in Android apps. In: Proceedings of the 13th Conference on Emerging Networking Experiments and Technologies, pp. 350–362. New York (2017)
25. Ren, J., Rao, A., Lindorfer, M., Legout, A., Choffnes, D.: ReCon: revealing and controlling PII leaks in mobile network traffic. In: Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services, pp. 361–374 (2016)
26. Rescorla, E.: The Transport Layer Security (TLS) Protocol Version 1.3. IETF RFC 8446, August 2018
27. Rescorla, E., Ray, M., Dispensa, S., Oskov, N.: Transport Layer Security (TLS) Renegotiation Indication Extension. IETF RFC 5746, February 2010
28. Stöber, T., Frank, M., Schmitt, J., Martinovic, I.: Who do you sync you are? Smartphone fingerprinting via application behaviour. In: Proceedings of the Sixth ACM Conference on Security and Privacy in Wireless and Mobile Networks, pp. 7–12 (2013)
29. Sun, G., Xue, Y., Dong, Y., Wang, D., Li, C.: An novel hybrid method for effectively classifying encrypted traffic. In: 2010 IEEE Global Telecommunications Conference GLOBECOM 2010, pp. 1–5 (2010)
30. Taylor, V.F., Spolaor, R., Conti, M., Martinovic, I.: Appscanner: automatic fingerprinting of smartphone apps from encrypted network traffic. In: IEEE European Symposium on Security and Privacy (EuroS&P), pp. 439–454 (2016)
31. Taylor, V.F., Spolaor, R., Conti, M., Martinovic, I.: Robust Smartphone App Identification Via Encrypted Network Traffic Analysis. CoRR abs/1704.06099 (2017). http://arxiv.org/abs/1704.06099
32. Tsai, F., Chang, E., Kao, D.: Whatsapp network forensics: discovering the communication payloads behind cybercriminals. In: 2018 20th International Conference on Advanced Communication Technology (ICACT), p. 1 (2018)
33. Verde, N.V., Ateniese, G., Gabrielli, E., Mancini, L.V., Spognardi, A.: No nat'd user left behind: fingerprinting users behind nat from netflow records alone. In: IEEE 34th International Conference on Distributed Computing Systems, pp. 218–227 (2014)