TLS Fingerprinting Techniques

Zlatina Gancheva, Patrick Sattler*, Lars Wüstrich*
*Chair of Network Architectures and Services, Department of Informatics
Technical University of Munich, Germany
Email: ga94vad@tum.de, sattler@net.in.tum.de, wuestrich@net.in.tum.de

***Abstract*—Internet security has become a key concern to society in the last couple of decades as more and more sensitive data are being transferred over the Internet. This has led to the adoption of cryptographic protocols such as Secure Sockets Layer protocol (SSL) and Transport Layer Security protocol (TLS), which serve to protect information sent across the Internet. However, even though encryption resolved many security problems, it raised another question and namely how to inspect network traffic while still complying with privacy restrictions.**

TLS Fingerprinting is a method, developed to assist network monitoring. This paper takes a closer look on how TLS Fingerprinting works and analyzes the advantages of it as a client identification method by reviewing different Fingerprinting implementations.

***Index Terms*—Transport Layer Security, Secure Socket Layer, Network monitoring, Client identification, Fingerprinting**

1. Introduction

Nowadays, Transport Layer Security protocol (TLS) is the cryptographic protocol that is used to encrypt the majority of the internet traffic. It creates a huge visibility gap, which serves to prevent third parties from observing user's communication. This, however, poses a challenge to network administrators, who are trying to analyze traffic and who do not necessarily have access to the endpoint devices. The traditional way of intercepting and decrypting traffic is no more applicable, since it does not comply with current privacy standards and causes lower network performance [1]. Therefore, there is a pressing need for a method to improve traffic analysis. A possible solution to this problem is the generation of TLS fingerprints to identify network clients.

TLS fingerprinting - a non-invasive method - meets all the following criteria for traffic monitoring. It aims to provide quick and successful client identification, while being compatible with existing technologies and preserving the integrity of the encrypted information [2], [3]. TLS fingerprinting a is completely passive and payload based approach, which works by capturing and analyzing the unencrypted messages exchanged during a TLS session initialization. In those messages both parties agree on various parameters such as protocol version and encryption keys, which will be used to establish the encrypted connection that follows. This procedure is defined by the term 'handshake' and it is subsequent to the TCP 3-Way Handshake. Most of the handshake parameters are specific enough for a unique client signature to be built and recorded into a database.

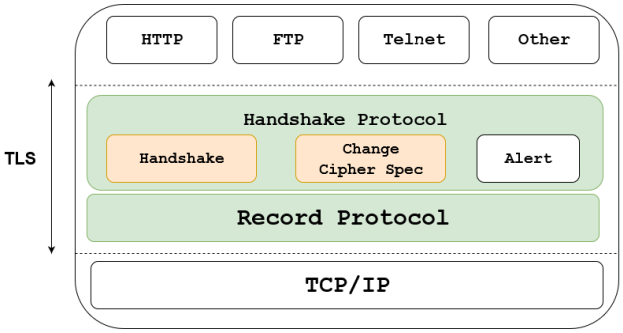

Figure 1: TLS protocol structure [4]–[7]

1.1. Outline

This paper is divided into 5 Sections. The second Section aims to explain TLS fingerprinting by making a detailed observation of the organization of the encryption it is based on. It provides a brief overview of TLS's history, current TLS versions in use and explains in detail important steps such as the TLS client-server Handshake. A detailed overview of different fingerprinting techniques is done in the third Section. The applicability of the results is discussed in Section four. Lastly, Section five concludes the paper.

2. Background

Transport Layer Security (TLS) is a cryptographic protocol, descendant of Secure Socket Layer protocol (SSL), and was first released in January 1999 [1]. Its most widespread version now is TLS 1.2 with more than 95 percent of the web servers supporting it as of February 2020 [8]. In 2018 however, a major TLS upgrade was made and TLSv1.3 was released. It aims to increase speed by reducing the number of handshake messages and improve security by not supporting outdated ciphers and hashing algorithms (e.g. SHA1, MD5, DES) [1]. The 1.3 version is relatively new, but the most popular browsers such as Chrome, Firefox and Opera already support it in their latest releases [9].

TLS should provide [10]–[12]:

- *Authentication* - The server always authenticates itself to the client.

doi: 10.2313/NET-2020-04-1_04

- *Data integrity* - After the connection establishment the data cannot be tampered with by attackers without detection.
- *Confidentiality* - After the connection establishment the data is only visible to the endpoints.

The TLS protocol enables client - server applications to communicate over the network in a manner, designed to prevent interference and eavesdropping. This done by encapsulating and encrypting data from the application layer, which ensures end-to-end security [4]. Hence, TLS is typically implemented on top of TCP with regard to the TCP/IP model (Figure 1). It is the encryption protocol currently standardized [13] for securing the most widespread network protocols, such as HTTP, FTP, SMTP and takes part in VoIP and VPN protocols [14]. As can be seen from Figure 1, the TLS protocol consists of two parts: the Handshake protocol and the Record protocol. The fist one ensures that the communicating parties authenticate themselves and is responsible for the negotiation of cryptographic parameters and key establishment [7]. The second one utilizes the parameters negotiated during the handshake. The Record protocol splits the transferred data into records, which are then individually protected [6]. For the purpose of TLS fingerprinting this paper is going to focus on the Handshake protocol, since as mentioned in the Introduction section only there the information exchanged between the client and the server is in plain format.

However, before going into detail about the handshake procedure, it must be pointed out that TLS uses a combination of both Symmetric and Asymmetric encryption [15]. Asymmetric encryption uses a public-private key pairing, so that data that is encrypted using the public key can only be decrypted using the private key and the other way around [16]. Its purpose is to authenticate the identity of the website's origin server. This is also known as public key encryption. Symmetric encryption on the other hand, uses only one key for encrypting and decrypting data. During the Handshake information is exchanged using asymmetric encryption, until the two sides are finished generating the session keys. Afterwards the session is encrypted using Symmetric encryption.

The Handshake process for TLSv1.3 is graphically presented in Fig 2 and explained as follows [17] :

1) The client calculates a few private/public keypairs for key exchange and requests a TLS Handshake by sending a *ClientHello* message that contains the following cryptographic information:
   - *Preferred TLS version* (TLS 1.3, 1.2, 1.1, etc.)
   - *Client random variable*, which represents a 32 byte string, used to prevent valid data transmission from repetition or delay with malicious intent.
   - *Session ID*, that has the default value of null if this is the first time connecting to this server [10].
   - *Cipher suites list* (e.g. ECDHE, RSA, PSK), ordered by preference of the client. A Cipher suite is a collection of encryption algorithms used to establish a secure connection [**?**].
   - *Compression methods*, used to decrease the bandwidth.
   - *List of Extensions*, which specify additional parameters (e.g. server name, padding). There are about 20 extensions, but among the most prominent ones are Signature Algorithms, Key Share, Elliptic Curves and Elliptic Curve Point Format [18]. They could also be included in the ClientHello fingerprint in order to bring more diversity [19]:
   - *List of public keys*, which contains a list of public keys that the server might find suitable for key exchange
2) The server calculates his private/public key-pair and answers the client with several messages. First is a *ServerHello* message that contains the negotiated protocol version, the chosen cipher suite, the session ID, another random byte string, compression method as well as the public key. The client and the server then both calculate the the shared session key that will be used to encrypt the rest of the handshake, using their private keys and the public key they have received from their partner.
3) The second message from the server is a *ChangeCipherSpec*, which serves to inform the client that from now on the all the messages will be encrypted with the shared key. (however in TLSv1.3 this message is sent simply as a middlebox compatibility mode [20])
4) A *Wrapper* message follows, that comprises of the *Server Encrypted Extensions*, *Server Cerificate*, *Server Certificate Verify* and *Server Finished* messages. The emphasis here falls on the fact that the rest of the handshake communication is encrypted, which is new in TLSv1.3 / is a major upgrade to TLSv1.2.
5) The client also sends a *ChangeCipherSpec*, which has the same purpose as the one send from the servers.
6) Finally the client also sends a *Wrapper* message containing the *Client Finished* message, informing that the handshake was successful for the client.

Now for the duration of the TLS session the server and client can send each other data that are encrypted symmetrically with the shared session key.

3. Fingerprinting

Previously a client used to be identified by the browser User-Agent found in the HTTP header. This is application layer information, which is now encrypted when the client uses cryptographic protocol such as TLS. Nevertheless, careful examination of network traffic has shown that clients can still be identified by capturing the unique set of plain text parameters from the Client- and ServerHello messages. It is important to point out that the elements of the Client- and ServerHello messages stay static throughout different sessions, which allows for previously known clients to be easily recognized. All the client records

doi: 10.2313/NET-2020-04-1_04

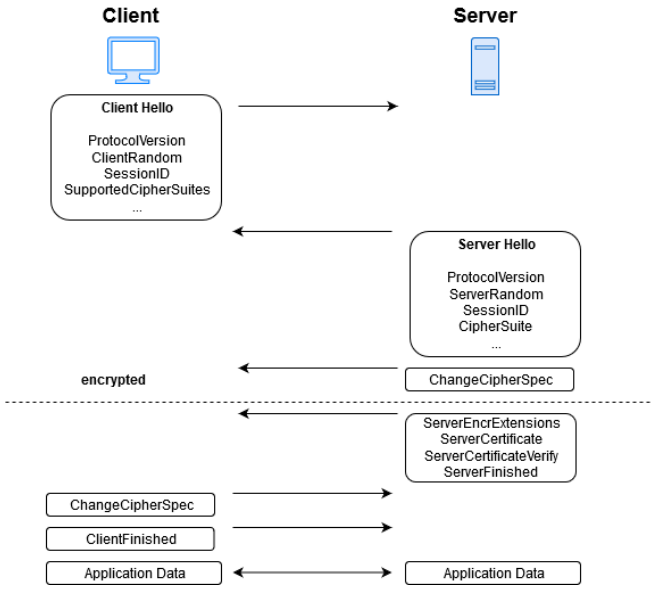

Figure 2: TLS Handhake Steps [1], [17]

are stored in a dictionary database, as this serves to quickly identify known TLS connections and fingerprint new unknown ones. In addition to that, clients with odd behavior can be tracked and discriminated if they are found to be malware applications. Since malware is known to use quite unique/custom parameters when they use TLS communication (normally old or obsolete TLS versions and/or small number of extensions or cipher suites) [2c03], a blacklist with their fingerprints can be composed to aid various TLS fingerprinting implementations.

3.1. Fingerprinting Methodologies

Numerous studies have worked with ClientHello packets to fingerprint TLS. In 2009 Ristić et al. [21] analyzed how to fingerprint SSL/TLS clients by evaluating the Handshake parameters, including the Cipher suites and Extensions list [21] [19]. In this Section three fingerprinting techniques using TLS implementations will be described.

Network-based HTTPS Client identification – this traffic analysis technique achieves proper client identification by creating a dictionary, where the Cipher suite list of the client is paired with their respective User-Agent. The list of Cipher suites is chosen over other elements from the TLS handshake, for it is the most diverse amongst the parameters, supposedly specific enough to identify a client. Other elements of the Handshake have only a few different values and are therefore found not suitable.

This method is based on a combination of two approaches [4]:

- *Host-based* - based on server monitoring - measures connections using the decrypted information from a HTTPS connection, such as the HTTP header, once it is received on the server side. The main advantage of this technique is that it provides results with high accuracy and is applicable in a controlled environment. It is however, also dependable on the amount of clients accessing the monitored server and there is no guarantee for diverse enough traffic. This could result in an insufficient amount of produced pairs.
  Essentially the accuracy of the data depends on the attractiveness of the server [4].
- *Flow-based* - based on network monitoring – this method works on the precondition that clients use both HTTP and HTTPS protocols when they communicate with the server. Thus it scans the traffic for connections that share the same IP source address. Then select a cipher suite list from the HTTPS connection and pairs it with the User-Agent from the HTTP connection, which is the closest in time [4], [14]. As opposed to the host-based approach, the flow-based one is not limited to a single server, so it provides more diverse pairs. Key weakness of the flow-based method is that is could provide ambiguous/perplexing results, because there are usually more than one User-Agent corresponding to a Cipher suite list [4]. Normally the User-Agents should have only slight differences, like software version. So the ones that deviate notably are supposedly connections, forged by web crawlers, pretending to be a legitimate clients. Therefore only the most similar User-Agents sub-strings were taken [14]. There are some possibilities to improve this method [14]. The first one is to manually inspect the pairs, which is still an approach prone to errors and time consuming. The second option is to repeat the measurement. Yet repeating it in a different time window or with different network settings would not necessarily provide complimentary results. Another way to fixing this shortcoming is to extend/spread the fingerprint to the TCP/IP layer.

Combined, the host- and flow- based approaches were proven to be sufficient for the creation of a usable dictionary. Such dictionaries must contain about 300 cipher suit lists with their assigned User-Agents in order to be reliable [14]. After careful examination of the results provided both methods show that the top 10 cipher suite lists covered more that 68 percent of the network traffic and the top 31 cipher suite lists are enough to represent about 90 percent of the traffic. This shows that using both method it is feasible to identify clients with high accuracy.

JA3/JA3S fingerprinting - this technique is a project from Salesforce [1], which utilizes both the ClientHello and the ServerHello to fingerprint the negotiation between client and server, using MD5 hash to produce an 32 character fingerprint, that is easy to digest. [22], [23].

Initially there was JA3, where only the client side of the TLS session establishing messages where exploited. It composes a client fingerprint by collects the decimal values of the bytes for the following fields in the ClientHello packet: Protocol version, Accepted Ciphers, List of Extensions, Elliptic Curves, and Elliptic Curve Formats [24]. It then joins those values together in a string, ordered as listed above, using commas to separate the field and a dash to part each value in each field. If there are no TLS Extensions in the ClientHello, the fields are left empty [23]. Those values are captured at the earliest possible stage, even before the server responds. This results in a

very large fingerprint, which is why the strings are hashed using MD5 hash.

Alone JA3 is not always enough to create an unique fingerprint, because when client applications use the same OS sockets or common libraries their JA3 fingerprints will be identical. A resolution to this shortcoming is the extension of JA3 - JA3S [1].

JA3S essentially does the same thing JA3 does, but with the server response – it uses the ServerHello packet to gather information from the following fields: Version, Accepted Ciphers, and List of Extensions [22], [23]. Then it concatenates them. This is useful, since servers reply to different clients differently, but to one client the same way in every session. JA3/JA3S provides additional benefits to the detection of malware. For example, if the JA3 fingerprint of the malicious application looks indistinguishable from a JA3 fingerprint of a legitimate application, so it can only be recognized from the server's response. Hence, the combined usage of JA3+JA3S contributes to a highly trustworthy identification / results in a more accurate malware detection [1], [22], [23].

Lastly, JA3/JA3S also has some disadvantages. The first one is that the MD5 hash has become obsolete [1]. It is important to clarify that back in 2017-2018 developers chose this hash type, because then it was supported by current technologies. However, as mentioned in the Background section, this is no longer the case, for MD5 is no longer supported in TLSv1.3, which urges the hash type of the JA3/JA3S to be changed. Additionally the JA3/JA3S technology is blacklist-based, which implies that its trustworthiness depends on how often the blacklist is updated.

Markov Chain Fingerprinting – this traffic classifications technique is conducted on the server side and is designed based on the message type sequence that emerges in a single-direction flow form the server to the client. It can use first-order or second-order homogeneous Markov chains to model statistical TLS fingerprint of different applications [25].

Fundamentally, Markov chains are utilized when computing the probabilities of certain events by viewing them as states transitioning into new or past states [26]. At the beginning of the method development, researchers used first-order homogeneous Markov chain model for computational simplicity [25]. This technique operates under the assumption that the parameters of each TLS session differ considerably and therefore the fingerprint of each application is distinctive enough. However, due to the limited amount of states during a TLS session, it could happen, that many applications contain alike transitions in their fingerprints, which could cause them to be misclassified. This problem could be avoided if the technique is upgraded to second-order Markov chains that are able to capture more diverse application features, which further balances the relationship between complexity and truthfulness [25].

There are a couple limitations to this method. The first one being, that applications change their TLS session initialization parameters over time, which means that for higher accuracy levels, it is advised that application fingerprints must to be updated periodically [27]. The second one being, that the technique struggles to recognize applications that have not taken part in the training stage. To resolve this issue, new application fingerprints must be incorporated in the existing database accordingly regularly [25].

Overall, the Markov chain fingerprinting technique results in proper applications discrimination, which can come from one of the following reasons [27]:

- incorrect and diverse implementation practices
- the misuse of TLS protocol
- various server configurations
- the application nature.

This leads us to the conclusion that proper classification could possibly be avoided by omitting implementation mistakes or creating the secure layer on a limited set [27].

4. Discussion

TLS Fingerprinting amongst other ways of client fingerprinting is a reliable method for traffic analysis and client identification [28]. It is passive, payload based and requires no endpoint agent data [29]. Nevertheless there could occur some inconveniences, such as collisions.

Fingerprint collision is the event of two fingerprints, belonging to different applications, overlapping [29]. The solution to collision avoidance is to take as many parameters from the ClientHello message as possible. Suitable candidates for that are extensions, such as Signature algorithms, Elliptic curves and Elliptic curve point format [29]. This offers greater variety in comparison to assessing cipher suites alone.

A different kind of inconvenience is the fact that TLS Fingerprinting implementations can be avoided or redirected. Based on the researches of Husak et al. [14] and Frolov et al. [19], a client can prevent TLS Fingerprinting in the following ways: by using a proxy, by manual change in the Cipher suite list or by mimicking popular TLS implementations.

- Usage of proxy redirects the TLS fingerprinting technology to fingerprint the cipher suite of the proxy instead of the one of the client [14]. However, the cipher suite list of a proxy could already exist in the fingerprint database and thus be recognized and associated accordingly.
- Manual changes in the client Cipher suite list are usually done by forced reducing of the list [14]. The client continues to communicate with a reduced Cipher suite list and is therefore not recognized as an existing record. So the Fingerprinting technique fails to find the corresponding User-Agent. But reducing it so much as to forge another client's cipher suite list is a quite difficult.
- Mimicking TLS implementations such as browsers. Mimicking also has its challenges - it is difficult to keep up with the rapidly-changing TLS browser implementations and their many features. It is also difficult to know what types of fingerprints to mimic.

Generally fingerprint collisions and TLS fingerprinting avoidance techniques are not a obstacle for the majority of fingerprinting tools. The biggest concern of TLS fingerprinting remains the database that the tools use, because TLS fingerprinting is only as good as the database

supporting it. Dictionaries may turn into disadvantage, if they are hard to maintain and update. Currently the process of the creation of data set collection was manual, but there are ongoing researches, attempting to automatize it in the future. [30].

Classification tools usually require to be trained on a particular data set consisting of benign traffic, which must be updated regularly to ensure novelty data. Especially hard to maintain and update is the collection of malware samples [2c03].

5. Conclusion

Encryption of data is crucial when aiming to protect the privacy of users. In modern networks, the TLS protocol is the current encryption standard for data transferred over the Internet. Although it is used to mask the plain text information from the application layer, TLS also provides a set of unique observable parameters that allow many conclusions to be made about both the client and the server [1].

In this paper we have reviewed the three most widely spread/diverse techniques used for TLS fingerprinting, starting with the simplest one – Network-based HTTPS Client identification – essentially divided into two approaches, which are both based on the extraction of the the most varied components from the TLS session initialization messages and writing them down in a database. The second one being the JA3/JA3S that is partially based on the Network-based identification as it upgrades it through memory optimization, hashing the values into 32-character unique fingerprint, making it quicker for malware software to be recognized. The last and most complicated method is the creation of a fingerprint using homogeneous Markov chains (either first or second order) so as to simulate the time-varying message sequence that occurs during the TLS session initialization. Vital characteristic trait of this method is that conducted on the server side and focuses mainly on detecting abnormal TLS sessions and improving discrimination practices. All of these techniques can identify clients with high accuracy while sustaining their privacy. A comparison based on the statistical accuracy of these techniques is hard to derive, because experiments with each one of them has been done individually, over different amounts of time, using different traffic samples.

In the future it would be interesting to conduct an experiment to test how these three techniques would perform under the same set of conditions (e.g. time window, network and servers).

Overall, TLS fingerprinting is a subsection of passive client identification and traffic. There are other methods for client fingerprinting, that may partially incorporate the TLS technology (for example OS fingerprinting [31], web browser fingerprinting, website fingerprinting, signal fingerprinting, cookies [32]) that are efficient as well.

References

[1] B. Anderson, S. Paul, and D. McGrew, "Deciphering malware's use of TLS (without decryption)." [Online]. Available: http://arxiv.org/abs/1607.01639

[2] L. Brotherston, "synackpse/tls-fingerprinting," accessed: 2020-01-23. [Online]. Available: https://github.com/synackpse/tls-fingerprinting

[3] The generation and use of TLS fingerprints. Accessed: 2020-01-23. [Online]. Available: https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=539893

[4] M. Husak, M. Cermak, T. Jirsik, and P. Celeda, "Network-based HTTPS client identification using SSL/TLS fingerprinting," in *2015 10th International Conference on Availability, Reliability and Security*. IEEE, pp. 389–396. [Online]. Available: http://ieeexplore.ieee.org/document/7299941/

[5] Transport layer security protocol | microsoft docs. Accessed: 2020-01-18. [Online]. Available: https://docs.microsoft.com/en-us/previous-versions/windows/it-pro/windows-server-2012-r2-and-2012/dn786441(v\%3Dws.11)

[6] M. D. Center. TLS record protocol - win32 apps. Accessed: 2020-01-23. [Online]. Available: https://docs.microsoft.com/en-us/windows/win32/secauthn/tls-record-protocol

[7] ——. TLS handshake protocol - win32 apps. Accessed: 2020-01-23. [Online]. Available: https://docs.microsoft.com/en-us/windows/win32/secauthn/tls-handshake-protocol

[8] Qualys SSL labs - SSL pulse. Accessed: 2020-02-23. [Online]. Available: https://www.ssllabs.com/ssl-pulse/

[9] Can i use... support tables for HTML5, CSS3, etc. Accessed: 2020-02-23. [Online]. Available: https://caniuse.com/#feat=tls1-3

[10] RFC 8446 - the transport layer security (TLS) protocol version 1.3. Accessed: 2019-12-13. [Online]. Available: https://tools.ietf.org/html/rfc8446#section-4.1.2

[11] RFC 5246 - the transport layer security (TLS) protocol version 1.2. Accessed: 2019-12-13. [Online]. Available: https://tools.ietf.org/html/rfc5246

[12] L. Brotherston, "Lee brotherston's work," accessed: 2019-12-13. [Online]. Available: https://github.com/synackpse/tls-fingerprinting

[13] P. Kotzias, A. Razaghpanah, J. Amann, K. G. Paterson, N. Vallina-Rodriguez, and J. Caballero, "Coming of age: A longitudinal study of TLS deployment," in *Proceedings of the Internet Measurement Conference 2018 on - IMC '18*. ACM Press, pp. 415–428, accessed: 2019-11-18. [Online]. Available: http://dl.acm.org/citation.cfm?doid=3278532.3278568

[14] M. Husák, M. Čermák, T. Jirsík, and P. Čeleda, "HTTPS traffic analysis and client identification using passive SSL/TLS fingerprinting," vol. 2016, no. 1, p. 6. [Online]. Available: https://doi.org/10.1186/s13635-016-0030-7

[15] An overview of the SSL or TLS handshake. Accessed: 2019-12-14. [Online]. Available: www.ibm.com/support/knowledgecenter/en/ssfksj_7.1.0/com.ibm.mq.doc/sy10660_.htm

[16] Comparative study of symmetric and asymmetric cryptography techniques | semantic scholar. Accessed: 2019-12-13. [Online]. Available: https://www.semanticscholar.org/paper/Comparative-Study-of-Symmetric-and-Asymmetric-Tripathi-Agrawal/e0e4810c5276f9c05cc82425fcf911f206c52bef

[17] The illustrated TLS 1.3 connection: Every byte explained. Accessed: 2020-01-18. [Online]. Available: https://tls13.ulfheim.net/

[18] TLSfingerprint.io - extensions. Accessed: 2019-12-13. [Online]. Available: https://tlsfingerprint.io/top/extensions

[19] S. Frolov and E. Wustrow, "The use of TLS in censorship circumvention," in *Proceedings 2019 Network and Distributed System Security Symposium*. Internet Society. [Online]. Available: https://www.ndss-symposium.org/wp-content/uploads/2019/02/ndss2019_03B-2-1_Frolov_paper.pdf

[20] Middlebox compatibility mode. Accessed: 2020-01-18. [Online]. Available: https://www.ibm.com/support/knowledgecenter/en/ssw_ibm_i_74/rzain/rzainmiddlebox.htm

[21] Ivan ristić: HTTP client fingerprinting using SSL handshake analysis. Accessed: 2019-12-11. [Online]. Available: https://blog.ivanristic.com/2009/06/http-client-fingerprinting-using-ssl-handshake-analysis.html

[22] Open sourcing JA3 - salesforce engineering. Accessed: 2019-12-13. [Online]. Available: https://engineering.salesforce.com/open-sourcing-ja3-92c9e53c3c41

[23] TLS fingerprinting with JA3 and JA3s - salesforce engineering. Accessed: 2019-12-13. [Online]. Available: https://engineering.salesforce.com/tls-fingerprinting-with-ja3-and-ja3s-247362855967

[24] B. Vasudevan, "Elliptic curves in transport layer security (TLS) - a presentation tutorial," p. 4.

[25] M. Shen, M. Wei, L. Zhu, and M. Wang, "Classification of encrypted traffic with second-order markov chains and application attribute bigrams," vol. 12, no. 8, pp. 1830–1843.

[26] K. Chan, C. Lenard, and T. Mills, "An introduction to markov chains."

[27] M. Korczynski and A. Duda, "Markov chain fingerprinting to classify encrypted traffic," in *IEEE INFOCOM 2014 - IEEE Conference on Computer Communications*. IEEE, pp. 781–789. [Online]. Available: http://ieeexplore.ieee.org/document/6848005/

[28] T. Bujlow, V. Carela-Español, J. Solé-Pareta, and P. Barlet-Ros, "Web tracking: Mechanisms, implications, and defenses." [Online]. Available: http://arxiv.org/abs/1507.07872

[29] SquareLemon. Accessed: 2019-11-17. [Online]. Available: https://blog.squarelemon.com/tls-fingerprinting/

[30] TLS fingerprinting in the real world. Accessed: 2019-12-13. [Online]. Available: https://blogs.cisco.com/security/tls-fingerprinting-in-the-real-world

[31] [1706.08003] OS fingerprinting: New techniques and a study of information gain and obfuscation. Accessed: 2019-12-13. [Online]. Available: https://arxiv.org/abs/1706.08003

[32] R. Upathilake, Y. Li, and A. Matrawy, "A classification of web browser fingerprinting techniques," in *2015 7th International Conference on New Technologies, Mobility and Security (NTMS)*, pp. 1–5, ISSN: 2157-4960.