A Large-scale Empirical Analysis of Browser Fingerprints Properties for Web Authentication

Nampoina Andriamilanto, Tristan Allard, Gaëtan Le Guelvouit, Alexandre Garel

A Large-scale Empirical Analysis of Browser Fingerprints Properties for Web Authentication

NAMPOINA ANDRIAMILANTO, Institute of Research and Technology b<>com, France and Univ Rennes, CNRS, IRISA, France
TRISTAN ALLARD, Univ Rennes, CNRS, IRISA, France
GAËTAN LE GUELVOUIT, Institute of Research and Technology b<>com, France
ALEXANDRE GAREL*, Institute of Research and Technology b<>com, France

Modern browsers give access to several attributes that can be collected to form a browser fingerprint. Although browser fingerprints have primarily been studied as a web tracking tool, they can contribute to improve the current state of web security by augmenting web authentication mechanisms. In this paper, we investigate the adequacy of browser fingerprints for web authentication. We make the link between the digital fingerprints that distinguish browsers, and the biological fingerprints that distinguish Humans, to evaluate browser fingerprints according to properties inspired by biometric authentication factors. These properties include their distinctiveness, their stability through time, their collection time, their size, and the accuracy of a simple verification mechanism. We assess these properties on a large-scale dataset of $4,145,408$ fingerprints composed of 216 attributes, and collected from $1,989,365$ browsers. We show that, by time-partitioning our dataset, more than 81.3% of our fingerprints are shared by a single browser. Although browser fingerprints are known to evolve, an average of 91% of the attributes of our fingerprints stay identical between two observations, even when separated by nearly 6 months. About their performance, we show that our fingerprints weigh a dozen of kilobytes, and take a few seconds to collect. Finally, by processing a simple verification mechanism, we show that it achieves an equal error rate of 0.61%. We enrich our results with the analysis of the correlation between the attributes, and of their contribution to the evaluated properties. We conclude that our browser fingerprints carry the promise to strengthen web authentication mechanisms.

Additional Key Words and Phrases: browser fingerprinting, web authentication

ACM Reference Format:
Nampoina Andriamilanto, Tristan Allard, Gaëtan Le Guelvouit, and Alexandre Garel. 2020. A Large-scale Empirical Analysis of Browser Fingerprints Properties for Web Authentication. 1, 1 (June 2020), 51 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Web authentication widely relies on the use of identifier-password pairs defined by the end user. The password authentication factor is easy to use and to deploy, but has been shown to suffer from severe security flaws when used without any additional factor. Real-life users indeed use common passwords[1], which paves the way to brute-force or guessing attacks [8]. Moreover, they tend to use

This paper is a major extension (more than 50% of new material) of work originally presented in [5].
*The author participated to the fingerprint collection and analysis when working at the institution, but is not anymore affiliated to it.
[1]https://www.troyhunt.com/86-of-passwords-are-terrible-and-other-statistics

Authors' addresses: Nampoina Andriamilanto, nampoina.andriamilanto@b-com.com, Institute of Research and Technology b<>com, 1219 avenue Champs Blancs, Cesson-Sévigné, France, 35510, Univ Rennes, CNRS, IRISA, 263 avenue du général Leclerc, Rennes, France, 35000; Tristan Allard, tristan.allard@irisa.fr, Univ Rennes, CNRS, IRISA, 263 avenue du général Leclerc, Rennes, France, 35000; Gaëtan Le Guelvouit, gaetan.leguelvouit@b-com.com, Institute of Research and Technology b<>com, 1219 avenue Champs Blancs, Cesson-Sévigné, France, 35510; Alexandre Garel, Institute of Research and Technology b<>com, 1219 avenue Champs Blancs, Cesson-Sévigné, France, 35510.

2020. XXXX-XXXX/2020/6-ART $15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

similar passwords across different websites [12], which increases the impact of successful attacks. Phishing attacks are also a major threat to the use of passwords. Over the course of a year, Thomas et al. [46] achieved to retrieve 12.4 million credentials stolen by phishing kits.

These flaws bring the need for supplementary security layers, primarily through multi-factor authentication [9], such that each additional factor provides an *additional security barrier*. However, this usually comes at the cost of *usability* (i.e., users have to remember, possess, or do something), and of *deployability* (i.e., implementers have to deploy dedicated hardware or software, teach users how to use them, and maintain the deployed solution).

In the meantime, *browser fingerprinting* [28] gains more and more attention. The seminal Panopticlick study [13] is the first work to highlight the possibility to build a *browser fingerprint* by collecting attributes from a browser (e.g., the `userAgent` property of the `navigator` JavaScript object). In addition to being widely used for web tracking purposes [14] (raising legal, ethical, and technical issues), browser fingerprinting is already used as an additional web authentication factor *in real-life*. The browser fingerprints constitute a supplementary factor that is verified at login with the other factors, as depicted in Figure 1 (see Appendix C for an example of an authentication mechanism that relies on browser fingerprints). Browser fingerprints are indeed a good *candidate* as an additional web authentication factor thanks to their distinctive power, their frictionless deployment (e.g., no additional software or hardware to install), and their usability (no secret to remember, no additional object to possess, and no supplementary action to carry out). As a result, companies like MicroFocus[2] or SecureAuth[3] include this technique into their authentication mechanisms.

However, to the best of our knowledge, no large-scale study rigorously evaluates the adequacy of browser fingerprints as an additional web authentication factor. On the one hand, most works about the use of browser fingerprints for authentication concentrate on the design of the authentication mechanism [17, 26, 34, 38, 43, 47]. On the other hand, the large-scale empirical studies on browser fingerprints focus on their effectiveness as a web tracking tool [13, 19, 29, 35]. Such a mismatch between the understanding of browser fingerprints for authentication – currently poor – and their ongoing adoption in real-life is a serious harm to the security of web users. The lack of documentation from the existing authentication tools (e.g., about the used attributes, about the distinctiveness and the stability of the resulting fingerprints) only adds up to the current state of ignorance, all this whereas security-by-obscurity directly contradicts the most fundamental security principles. Moreover, the distinctiveness of browser fingerprints that can be achieved when considering a wide-surface of fingerprinting attributes on a large population is, to the best of our knowledge, unknown. On the one hand, the studies that analyze browser fingerprints in a large-scale (more than $100,000$ fingerprints) consider fewer than thirty attributes [13, 19, 29, 48]. This underestimates the distinctiveness of the fingerprints (e.g., [19] reports a rate of 33.6% of unique fingerprints), as it increases the chances for browsers to share the same fingerprint. All this whereas more than a hundred attributes are accessible. On the other hand, the studies that consider more than fifty attributes either work on less than two thousands browsers [24, 35], or do not analyze the resulting fingerprints at all [3]. The current knowledge about the hundreds of accessible attributes (e.g., their stability, their collection time, their correlation) is, to the best of our knowledge, also incomplete. Indeed, previous studies consider few attributes [10, 13, 15, 19, 29, 33, 36, 41], or focus on a single aspect of them (e.g., their stability [48]).

Our contributions. We conduct the first *large-scale data-centric empirical study* of the *fundamental properties of browser fingerprints* when used as an additional web authentication factor. We

[2]https://www.microfocus.com/media/white-paper/device-fingerprinting-for-low-friction-authentication-wp.pdf
[3]https://docs.secureauth.com/pages/viewpage.action?pageId=33063454

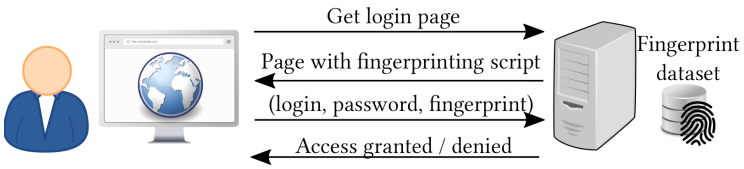

Fig. 1. A simplified web authentication mechanism that relies on browser fingerprinting.

base our findings on an in-depth analysis of a real-life fingerprint dataset collected over a period of 6 months, that contains 4,145,408 fingerprints composed of 216 attributes. In particular, our dataset includes nine *dynamic attributes* of three types, which values depend on instructions provided by the fingerprinter: five HTML5 canvases [10], three audio fingerprinting methods [36], and a WebGL canvas [33]. The dynamic attributes are used within state-of-the-art web authentication mechanisms to mitigate replay attacks [26, 38]. Each dynamic attribute has been studied singularly, but their fundamental properties have not yet been studied simultaneously on the same browser population. To the best of our knowledge, no related work considers a dataset of this scale, in terms of both fingerprints and attributes, together with various dynamic attributes. We formalize, and assess on our dataset, the properties necessary for paving the way to elaborate browser fingerprinting authentication mechanisms. We make the link between the digital fingerprints that distinguish browsers, and the biological fingerprints that distinguish Humans, to evaluate browser fingerprints according to properties inspired by biometric authentication factors [16, 31, 52]. We stress that we do not make any assumption on the inner working of the authentication mechanism, and consequently on the adversarial strategy. The properties aim at characterizing the adequacy and the practicability of browser fingerprints, independently of their use within future authentication mechanisms. In particular, we measure the size of the browser anonymity sets through time, the proportion of identical attributes between two observations of the fingerprint of a browser, the collection time of the fingerprints, their size, the loss of efficacy between device types, and the accuracy of a simple illustrative verification mechanism. To comprehend the obtained results on the complete fingerprints, we include an in-depth study of the contribution of the attributes to the fingerprint properties. Moreover, we discuss the correlation between the attributes, make a focus on the contribution of the dynamic attributes, and provide the exhaustive list of the attributes, together with their properties. To the best of our knowledge, no previous work analyzed browser fingerprinting attributes at this scale, in terms of the number of attributes, of the number of fingerprints, and of the variety of properties (e.g., stability, collection time).

In a nutshell, we make the following contributions:

(1) We formalize the fundamental properties that browser fingerprints should provide to be usable and practical as a web authentication factor.
(2) About their adequacy, we show that (1) considering a wide surface of 216 fingerprinting attributes on our large population provides a proportion of unique fingerprints – also called unicity rate – of 81.8% on our complete dataset, (2) by time-partitioning our dataset, the unicity rates are stable on the long term at around 81.3%, and 94.7% of our fingerprints are shared by 8 browsers or fewer, (3) on average, a fingerprint has more than 91% of identical attributes between two observations, even when separated by nearly 6 months, (4) our mobile browsers lack distinctiveness, as they show a unicity rate of 42%.
(3) About their practicability, we show that (1) the generated fingerprints weigh a dozen of kilobytes, (2) they are collected within seconds, (3) the accuracy of a simple illustrative verification mechanism is close to perfect, as it achieves an equal error rate of 0.61%.

(4) We enrich our results with (1) a precise analysis of the contribution of each attribute (a) to the distinctiveness, and show that 10% of the attributes provide a normalized entropy higher than 0.25, (b) to the stability, and show that 85% of the attributes stay identical for 99% of the consecutive fingerprints coming from the same browser, (c) to the collection time, and show that only 33 attributes take more than 5ms to collect, (d) to the fingerprint size, and show that only 20 attributes weigh more than 100 bytes, (2) a discussion about the correlation of the attributes, and show that only 49 attributes can completely be inferred when knowing another attribute, (3) a focus on the properties of the nine dynamic attributes.
(5) We provide an in-depth description of our methodology and our dataset with the goal of making our results reproducible. In particular, we include an exhaustive list of the collected attributes together with their properties, and a detailed description of the preprocessing of the fingerprints.

This paper is a major extension of work originally presented in [5], and brings more than 50% of new material. Specifically, we clarify the obtained results by discussing them further, we evaluate the accuracy of a simple illustrative verification mechanism, we highlight the contribution of the attributes to the properties, and we provide a comprehensive list of the attributes, together with their properties and their concrete implementation. In a summary, we make the following additions:

(1) Section 2 gains a description of the studied browser population. This includes the share of the families of browser and of operating system, and insights about the bias towards French browsers. We also add a description of the data preprocessing step, and a comparison between our dataset and the dataset of previous studies.
(2) Section 3 gains the accuracy of a simple illustrative verification mechanism as an additional performance property.
(3) Section 4 gets the results further discussed, and gains the results of the accuracy of the simple illustrative verification mechanism.
(4) Section 5 is entirely new. It discusses the contribution of the attributes to the fingerprint properties, the correlation between the attributes, and the properties of the dynamic attributes.
(5) Section 6 is also new, and provides related works about the use of browser fingerprinting for authentication.
(6) The appendices are also new. Appendix A describes the concrete implementation of the studied attributes for reproducibility. Appendix B provides the keywords used to classify the fingerprints. Appendix C describes how browser fingerprints can be integrated into a web authentication mechanism. Appendix D discusses a more complex verification mechanism that rely on distance functions on the attributes to compare fingerprints. Appendix E provides the complete list of the attributes with their properties (e.g., number of distinct values, stability).

The rest of the paper is organized as follows. Section 2 describes the dataset analyzed in this study. Section 3 presents and formalizes the properties evaluated in the analysis. Section 4 presents the experimental results. Section 5 breaks down the analysis to the attributes to comprehend the results on the complete fingerprints. Section 6 positions this study with the related works. Finally, Section 7 concludes.

2 DATASET

In this section, we describe the browser fingerprint dataset that is analyzed in this study. First, we present the conditions of the collection, and describe the precautions taken to protect the privacy

of the experimenters. Then, we detail the preprocessing steps to cleanse the raw dataset. Finally, we describe the working dataset, and compare it with the large-scale datasets of previous studies.

2.1 Fingerprints collection

To study the properties of browser fingerprints on a real-world browser population, we launched an experiment in collaboration with the authors of the Hiding in the Crowd study [19], together with an industrial partner that controls one of the top 15 French websites according to the site ranking service Alexa[4]. The authors of the Hiding in the Crowd study only consider the 17 attributes of their previous work [29], and focus on the issue of web tracking. On the contrary, we consider in this work more than one order of magnitude more attributes – 216 attributes – and focus on the use of browser fingerprinting as an additional web authentication factor.

2.1.1 Experiment. The experiment consisted into integrating a fingerprinting script on two general audience web pages that are controlled by our industrial partner, which subjects are political news and weather forecast. The script was active between December 7, 2016, and June 7, 2017, and fingerprinted the visitors who consented to the use of cookies in compliance with the European directives 2002/58/CE and 2009/136/CE. To differentiate two browsers in future analysis, we assigned them a unique identifier (UID) as a 6-months cookie, which was sent alongside fingerprints. Similarly to previous studies [13, 29], we coped with the issue of cookie deletion by storing a one-way hash of the IP address as well, computed by a secure cryptographic hash function. We refer the interested reader to Section 2.3 for more details on the measures taken to protect the privacy of the experimenters.

2.1.2 Browser fingerprinting attributes. The fingerprinting script used in the experiment includes 216 attributes divided into 200 JavaScript properties, together with their collection time, and 16 HTTP header fields. In particular, they include three types of dynamic attributes that comprise five HTML5 canvases [10], three audio fingerprinting methods [36], and a WebGL canvas [33].

We sought to evaluate the properties of browser fingerprints when considering as many attributes as possible, to estimate more precisely what can really be achieved. Hence, we compiled the attributes from previous studies and open source projects. If the value of an attribute is not accessible, a flag explaining the reason is stored instead, as it is still exploitable information. Indeed, two browsers can be distinguished if they behave differently on the inaccessibility of an attribute (e.g., returning a `null` value is different from throwing an exception). We also configure a timeout after which the fingerprint is sent without waiting for every attribute to be collected. The attributes that were not collected are set to a specific flag. The complete list of attributes and their properties is available in Appendix E.

Client-side attributes only consist of JavaScript properties. No plugins (e.g., Flash, Silverlight) are used due to their removal and replacement by HTML5 functionalities[5]. Moreover, the fingerprinting script collects the HTTP headers from requests sent by JavaScript, hence the dataset contains no fingerprint of browsers having JavaScript disabled.

2.2 Browser population bias

The previously presented datasets were collected through dedicated websites, and are biased towards privacy-aware and technically-skilled persons [13, 29, 35]. Our dataset is more general audience oriented, and is not biased towards this type of population. Nevertheless, the website audience is mainly French-speaking users. This leads to biases of which we provide examples here.

[4]https://www.alexa.com/topsites/countries/FR
[5]https://theblog.adobe.com/adobe-flash-update

By matching the `userAgent` JavaScript property with manually selected keywords (see Appendix B), we obtain the share of each browser family. Firefox browsers are the most common, and constitute 31.37% of our desktop browsers, followed by Internet Explorer browsers (28.26%), and Chrome browsers (26.22%). Although French users use more Firefox as their desktop browser than the world average[6], the share of browsers is different from what is reported by Statcounter[7] with Chrome being the most common with 68.11% of the French browsers. The operating systems of our desktop browsers are mostly Microsoft ones, with 39.13% of Windows 10, 35.58% of Windows 7, and 10.21% of other Windows versions. Mac OS only represents 5.98% of our desktop browsers, and Linux-based browsers less than 1%. This can be explained by the population visiting the website being less technically savvy, hence using more common web environments (e.g., a Firefox on a Windows operating system) than technical environments (e.g., Linux-based operating systems).

For the mobile browsers, the vast majority of them run on an Android platform (84.42%), followed by Windows Phone (8.76%), and iOS (5.18%). The most common browser is Samsung Browser (45.10%), followed by Chrome (38.54%), Internet Explorer mobile (8.44%), and Safari (6.70%). The browsers running on Android devices tend to be more distinguishable than the ones running on iOS [29], due to the plurality of device vendors and models that embark the Android operating system.

The contextual attributes related to the time zone or the configured language are less distinctive than in previous studies (see Section 5.1.1). For example, the normalized entropy of the `Timezone` JavaScript property is of only 0.008, against 0.161 for the Panopticlick study [13], and 0.198 for the AmIUnique study [29]. These attributes also tend towards the typical French values: 98.48% of the browsers have a `Timezone` value of −1, 98.59% of them have the daylight saving time enabled, and `fr` is present in 98.15% of the value of the `Accept-Language` HTTP header.

The browsers of our dataset mostly belong to general audience French users. Counter-intuitively, considering an international population may not reduce the distinctiveness. Indeed, we can expect foreign users to have a combination of contextual attributes (e.g., timezone, languages) different from the French users, making them distinguishable even if the remaining attributes have identical values. Al-Fannah and Li [2] found out that browser families are not equally fingerprintable (e.g., Safari browsers are less distinguishable than Chrome browsers). Although the fingerprintability of the browser families of the studied population impacts the obtained distinctiveness, studying this aspect is out of the scope of this paper.

2.3 Privacy concerns

The browser fingerprints are sensible due to their identification capacity. We complied with the European directives 2002/58/CE[8] and 2009/136/CE[9] in effect at the time, and took additional measures to protect the participating users.

First, the script was set on two web pages of a single domain in a first-party manner, hence providing no extra information about the browsing of the users. The content of the web pages are generic, hence they do not leak any information about the interests of the users. Second, we restricted the collection to the users having consented to cookies, as required by the European directives 2002/58/CE and 2009/136/CE. Third, a UID was set as a cookie with a 6-months lifetime, corresponding to the duration of the experiment. Fourth, we deleted the fingerprints for which the `cookieEnabled` property was not set to `true`. Finally, we hashed the IP addresses by passing them through a HMAC-SHA256 using a key that we threw afterwards. It was done using the `secret`

[6]https://gs.statcounter.com/browser-market-share/desktop/worldwide/2020
[7]https://gs.statcounter.com/browser-market-share/desktop/france/2020
[8]https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX:02002L0058-20091219
[9]https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX:02009L0136-20091219

and the `hmac` libraries of Python3.6. These one-way hashed IP addresses are only used for the UIDs resynchronization (see Section 2.4.2), and are not used as an attribute in the working dataset.

2.4 Data preprocessing

Given the experimental aspect of browser fingerprints, and the scale of our collection, the raw dataset contains erroneous or irrelevant samples. That is why we perform several preprocessing steps before any analysis. The dataset is composed of entries in the form of (f, b, t) tuples so that the fingerprint f was presented by the browser b at the given time t. We talk here about entries (i.e., (f, b, t) tuples) and not fingerprints (i.e., only f) to avoid confusion. The preprocessing is divided into four steps: the cleaning, the UIDs resynchronization, the deduplication, and the derivation of the extracted attribute. Initially, we have $8,205,416$ entries in the raw dataset.

2.4.1 Dataset cleaning. The dataset cleaning step filters out $70,460$ irrelevant entries, following the method described here. The fingerprinting script prepares, sends, and stores the entries in string format consisting of the attribute values separated by semicolons. We remove 769 entries that have a wrong number of fields, mainly due to truncated or unrelated data (e.g., the body of a post request). We filter out $53,251$ entries that belong to robots, by checking that blacklisted keywords are present in the `UserAgent` HTTP header (see Appendix B for the keywords list). We reduce the entries that have multiple exact copies (down to the same moment of collection) to a single instance. Finally, we remove $18,591$ entries that have the cookies disabled, and $2,412$ entries that have a time of collection that falls outside the time window of the experiment.

2.4.2 Unique IDs resynchronization. The resynchronization step replaces $181,676$ UIDs with a total of $116,708$ other UIDs, following the method described here. The cookies are considered an unreliable browser identification solution, hence we undergo a cookie resynchronization step, similarly to the Panopticlick study [13]. We consider the entries that have the same (fingerprint, IP address hash) pair to belong to the same browser, and assign them the same UID. Similarly to the Panopticlick study, we do not synchronize the interleaved UIDs, that are the UIDs related to the entries having the same (fingerprint, IP address hash) pairs, but showing UID values b_1, b_2, then b_1 again.

2.4.3 Deduplication. The deduplication step constitutes the biggest cut in our dataset, and filters out $2,420,217$ entries. To avoid storing duplicates of the same fingerprint observed several times for a browser, the usual way is to ignore a fingerprint if it was already seen for a browser during the collection [13, 29]. Our script collects the fingerprint on each visit, no matter if it was already seen for this browser or not. To stay consistent with common methodologies, we deduplicate the fingerprints offline. For each browser, we hold the first entry that contains a given fingerprint, and ignore the following entries if they also contain this fingerprint. This method takes the interleaved fingerprints into account, that are the fingerprints so that we observe f_1, f_2, then f_1 again. For example, if a browser b has the entries $\{(f_1, b, t_1), (f_2, b, t_2), (f_2, b, t_3), (f_1, b, t_4)\}$, we only hold the entries $\{(f_1, b, t_1), (f_2, b, t_2), (f_1, b, t_4)\}$ after the deduplication step.

We hold the interleaved fingerprints to realistically simulate the state of the fingerprint of each browser through time. We find that 10.59% of our browsers showed at least one case of interleaved fingerprints. The interleaved fingerprints can come from attributes that switch between two values. An example is the screen size that changes when an external screen is plugged or unplugged. Previous studies discarded the fingerprints that were already encountered for a given browser [13, 19, 29], hiding the interleaved fingerprints.

Table 1. Comparison between the datasets of the studies Panopticlick (PTC), AmIUnique (AIU), Hiding in the Crowd (HitC), Long-Term Observation (LTO), and this study. As the LTO study tests various attribute sets, we present the ranges of the results obtained by their attribute selection method. - denotes missing information. * denotes deduced information. Fps is the short for fingerprints. The attributes comprise the derived attributes, and the fingerprints are counted after data preprocessing.

	PTC [13]	AIU [29]	HitC [19]	LTO [35]	**This study**
Collection period	3 weeks	3-4 months*	6 months	3 years	**6 months**
Attributes	8	17	17	305	**262**
Browsers	-	-	-	-	**1,989,365**
Fingerprints	470,161	118,934	2,067,942	88,088	**4,145,408**
Distinct fingerprints	409,296	142,023[10]	-	9,822–16,541	**3,578,196**
Ratio of desktop fps	-	0.890*	0.879	0.697*–0.707*	**0.805**
Ratio of mobile fps	-	0.110*	0.121	0.293–0.303	**0.134**
Unicity of overall fps	0.836	0.894	0.336	0.954–0.958	**0.818**
Unicity of mobile fps	-	0.810	0.185	0.916–0.941	**0.399**
Unicity of desktop fps	-	0.900	0.357	0.974–0.978	**0.884**

2.4.4 Extracted attributes. We derive 46 extracted attributes of two types from 9 original attributes. First, we have the extracted attributes that are parts of an original attribute, like an original attribute that is composed of 28 triplets of RGB (Red Green Blue) color values that we split into 28 single attributes. Then, we have the extracted attributes that are derived from an original attribute, like the number of plugins derived from the list of plugins. The extracted attributes do not increase the distinctiveness as they come from an original attribute, and they are, at most, as distinctive as their original attribute. However, the extracted attributes can offer a higher stability than their original attribute, as the latter is impacted by any little change among the extracted attributes. For example, if exactly one of the 28 RGB values changes between two fingerprint observations, the original attribute is counted as having changed, but only one of the extracted attributes will be.

2.4.5 Working dataset. The working dataset obtained after the preprocessing steps contains 5,714,738 entries (comprising the identical fingerprints that are interleaved for each browser), with 4,145,408 fingerprints (comprising no identical fingerprint for each browser), and 3,578,196 distinct fingerprints. They are composed of 216 original attributes and of 46 extracted attributes, for a total of 262 attributes. The fingerprints come from 1,989,365 browsers, 27.53% of which have multiple fingerprints. Table 1 displays a comparison between the dataset of the studies Panopticlick [13], AmIUnique [29], Hiding in the Crowd [19], Long-Term Observation [35], and this study.

2.5 Comparison with previous studies

We compare our working dataset with the datasets of previous studies in Table 1, notably by the unicity rate, which is the proportion of the fingerprints that have been observed for a single browser only.

We have a lower unicity rate compared to previous studies [13, 29] due to our larger population. Even if our unicity rate is lower than that of AmIUnique, the gap is not extremely wide. This is due to the fact that considering a larger browser population leads to higher chances of collision,

[10]This number is displayed in Figure 11 of [29] as the number of distinct fingerprints, but it also corresponds to the number of raw fingerprints collected. Every fingerprint would be unique if the number of distinct and of collected fingerprints are equal. Hence, we are not confident in this number, but it is the number provided by the authors.

reducing the unicity rate. However, by considering a wider surface of fingerprinting attributes, we reduce these chances. These two effects produce this slight decrease of the unicity rate. We also observe a lower unicity rate for the fingerprints of mobile browsers compared to the fingerprints of desktop browsers, confirming the findings of previous studies [13, 19, 29, 42].

The authors of the Hiding in the Crowd study [19] worked on fingerprints collected from the same experiment and browser population than ours. However, to stay consistent with their previous study [29], they consider the same set of 17 attributes. This explains the higher number of fingerprints, as two browsers that have different fingerprints for a given set of attributes can come to the same fingerprint if a subset of these attributes is considered. Hence, they remove more duplicated fingerprints than us due to the higher chances for a browser to present the same fingerprint for 17 attributes than for 216 attributes. Our unicity rate is also higher, being at 81.8% for the complete dataset against 33.6% for [19]. This is due to the larger set of considered attributes that distinguish browsers more efficiently, as each additional attribute can provide a way to distinguish browsers. Our little drops on the proportion of desktop and mobile browsers come from a finer-grained classification, as we have 4.8% of the browsers that are classified as belonging to tablets, smart TVs, and game consoles.

3 AUTHENTICATION FACTOR PROPERTIES

Biometric authentication factors and browser fingerprints share strong similarities. They both work by extracting features from a unique entity, which is a person for the former and a browser for the latter, that can be used for identification or authentication. Although the entity is unique, the extracted features are a digital representation of the entity, that can lead to imperfections (e.g., the fingerprints of two different persons can show similar representations). Previous studies [16, 31, 52] identified the properties for a biometric characteristic to be *usable*[11] as an authentication factor, and the additional properties for a biometric authentication scheme to be *practical*. We evaluate browser fingerprints according to these properties, because of their similarity with biometric authentication factors.

The four properties needed for a biometric characteristic to be *usable* as an authentication factor are the following.

- *Universality*: the characteristic should be present in everyone.
- *Distinctiveness*: two distinct persons should have different characteristics.
- *Permanence*: the same person should have the same characteristic over time. We rather use the term *stability*.
- *Collectibility*: the characteristic should be collectible and measurable.

The three properties that a biometric authentication scheme requires to be *practical* are the following.

- *Performance*: the scheme should be accurate, consume few resources, and be robust against environmental changes.
- *Acceptability*: the users should accept to use the scheme in their daily lives.
- *Circumvention*: it should be difficult for an attacker to deceive the scheme.

The properties that we study are the *distinctiveness*, the *stability*, and the *performance*. We consider that the *universality* and the *collectibility* are satisfied, as the HTTP headers that are automatically sent by browsers constitute a fingerprint. However, we stress that a loss of distinctiveness occurs when no JavaScript attribute is available. About the *circumvention*, we refer the reader to Laperdrix et al. [26] that analyzed the security of an authentication mechanism based on browser

[11] Here, *usable* refers to the adequacy of the characteristic to be used for authentication, rather than the ease of use by the users.

fingerprints. We let the evaluation of the *acceptability* as future works, but we stress that such mechanisms are already used in real-life [50].

3.1 Distinctiveness

To satisfy the *distinctiveness* property, the browser fingerprints should distinguish two different browsers. The two extreme cases are every browser sharing the same fingerprint, which makes them indistinguishable from each other, and no two browsers sharing the same fingerprint, making every browser distinguishable. The distinctiveness falls between these extremes, depending on the attributes and the browser population. We consider the use of browser fingerprinting as an additional authentication factor. Hence, we do not require a perfect distinctiveness, as it is used in combination with other authentication factors to improve the overall security.

The dataset entries are composed of a fingerprint, the source browser, and the moment of collection in the form of a Unix timestamp in milliseconds. We denote B the domain of the unique identifiers, F the domain of the fingerprints, and T the domain of the timestamps. The fingerprint dataset is denoted D, and is formalized as:

$$D = \{(f, b, t) \mid f \in F, b \in B, t \in T\} \tag{1}$$

We use the size of the browser anonymity sets to quantify the distinctiveness, as the browsers that belong to the same anonymity set are indistinguishable. We denote $\mathcal{B}(f, \mathcal{D})$ the function that returns the browsers that provided the fingerprint f in the dataset D. It is formalized as:

$$\mathcal{B}(f, \mathcal{D}) = \{b \in B \mid \forall (g, b, t) \in \mathcal{D}, f = g\} \tag{2}$$

We denote $\mathcal{A}(\epsilon, \mathcal{D})$ the function that provides the fingerprints that have an anonymity set of size ϵ (i.e., that are shared by ϵ browsers) in the dataset D. It is formalized as:

$$\mathcal{A}(\epsilon, \mathcal{D}) = \{f \in F \mid \mathrm{card}(\mathcal{B}(f, \mathcal{D})) = \epsilon\} \tag{3}$$

A common measure of the fingerprint distinctiveness is the *unicity rate* [13, 19, 29], which is the proportion of the fingerprints that were observed for a browser only. We denote $\mathcal{U}(\mathcal{D})$ the unicity rate of the dataset $\mathcal{D}$, which is formalized as:

$$\frac{\mathrm{card}(\mathcal{A}(1, \mathcal{D}))}{\mathrm{card}(F)} \tag{4}$$

Previous studies measured the anonymity set sizes on the whole dataset [13, 19, 29]. We measure the anonymity set sizes on the fingerprints currently in use by each browser, and not on their whole history. Two different browsers, sharing the same fingerprint on different time windows, are then distinguishable (e.g., a browser was updated before the other). Moreover, a browser that runs in a fancy web environment (e.g., having a custom font), and that has several fingerprints in the dataset (e.g., fifty), will bloat the proportion of unique fingerprints, biasing the study (e.g., fifty fingerprints are unique whereas they come from a single browser).

We evaluate the anonymity set sizes on the time-partitioned datasets composed of the last fingerprint seen for each browser at a given time. Let $\mathcal{S}_\tau(\mathcal{D})$ be the time-partitioned dataset originating from $\mathcal{D}$ that represents the state of the fingerprint of each browser after τ days. With t_τ the last timestamp of this day, we have:

$$\mathcal{S}_\tau(\mathcal{D}) = \{(f_i, b_j, t_k) \in \mathcal{D} \mid \forall (f_p, b_q, t_r) \in \mathcal{D}, b_j = b_q, t_r \leq t_k \leq t_\tau\} \tag{5}$$

3.2 Stability

To satisfy the *stability* property, the fingerprint of a browser should stay sufficiently similar between two observations to be recognizable. Browser fingerprints have the particularity of evolving

through time, due to changes in the web environment, like a software update or a user configuration. We measure the stability by the average similarity between the consecutive fingerprints of browsers, given the elapsed time between their observation. The two extreme cases are every browser holding the same fingerprint through its life, and the fingerprint changing completely on each observation. A lack of stability makes it harder to recognize the fingerprint of a browser between two observations.

We denote $C(\Delta, \mathcal{D})$ the function that provides the pairs of consecutive fingerprints of $\mathcal{D}$ that are separated by a time-lapse comprised in the Δ time range. It is formalized as:

$$\begin{aligned} C(\Delta, \mathcal{D}) = \{(f_i, f_p) \mid \forall((f_i, b_j, t_k), (f_p, b_q, t_r)) \in \mathcal{D}^2, b_j = b_q, t_k < t_r, (t_r - t_k) \in \Delta, \\ \nexists(f_c, b_d, t_e) \in \mathcal{D}, b_d = b_j, f_c \neq f_i, f_c \neq f_p, t_k < t_e < t_r\} \end{aligned} \tag{6}$$

We consider the Kronecker delta $\delta(x, y)$, being 1 if x equals y, and 0 otherwise. We consider the set Ω of the n used attributes. We denote $f[\omega]$ the value taken by the attribute ω for the fingerprint f. Let $\mathrm{sim}(f, g)$ be a simple similarity function between the fingerprints f and g, which is formalized as:

$$\mathrm{sim}(f, g) = \frac{1}{n} \sum_{\omega \in \Omega} \delta(f[\omega], g[\omega]) \tag{7}$$

We define the function $\mathrm{avsim}(\Delta, \mathcal{D})$ that provides the average similarity between the pairs of the consecutive fingerprints, for a given time range Δ and a dataset D. It is formalized as:

$$\mathrm{avsim}(\Delta, \mathcal{D}) = \frac{\sum_{(f,g) \in C(\Delta, \mathcal{D})} \mathrm{sim}(f, g)}{\mathrm{card}(C(\Delta, \mathcal{D}))} \tag{8}$$

3.3 Performance

We consider four aspects of the *performance* of browser fingerprints for web authentication: their *collection time*, their *size* in memory, the *loss of efficacy* between different device types, and the *accuracy* of a simple illustrative verification mechanism.

Browser fingerprinting can easily be deployed by adding a script on the authentication page, and by preparing the servers to handle the reception, the storage, and the verification of the fingerprints. The users solely rely on their regular web browser, and do not have to run any dedicated application, nor possess a specific hardware, nor undergo a configuration step. The main additional load is on the supplementary consumption of memory and time resources. Moreover, the web environment differ between device types (e.g., mobile browsers have more limited functionalities than desktop browsers) and through time (e.g., modern browsers differ from the browsers from ten years ago), impacting the efficacy of browser fingerprinting. Finally, we evaluate the accuracy of the verification under fingerprints evolution, considering a simple illustrative verification mechanism.

3.3.1 Collection time. The browser fingerprints can be solely composed of passive attributes (e.g., HTTP headers) that are transmitted along with the communications with the server. In this case, the fingerprints are collected without the user perceiving any collection time, but major attributes are put aside. The client-side properties collected through JavaScript provide more distinctive attributes, at the cost of an additional collection time. We measure the collection time of the fingerprints considering only the JavaScript attributes, and ignore the HTTP headers that are transmitted passively.

3.3.2 Size. Browser fingerprinting consumes memory resources on the clients during the buffering of the fingerprints, on the wires during their sending, and on the servers during their storage. The memory consumption depends on the storage format of the fingerprints. For example, a canvas

can be stored as an image encoded in a base64 string, or as a hash, which is shorter. A trade-off has to be done between the quantity of information and the memory consumption. The less memory-consuming choice is to store the complete fingerprint as a single hash. However, the fingerprints evolve through time – even more when they are composed of many attributes – which results in the use of a hash of the complete fingerprint being impractical. Due to the unspecified size of the attributes (e.g., the `UserAgent` specification does not define a size limit[12]), we measure their *size* given a fingerprint dataset.

3.3.3 Loss of efficacy. The *loss of efficacy* is the loss of stability, of distinctiveness, or of performance of the fingerprints. It can occur either for a group of browsers (e.g., mobile browsers), or resulting from changes brought to web technologies.

First, previous works showed differences in the properties of the fingerprints coming from mobile and desktop devices [19, 29, 42], notably a limited distinctiveness for the mobile browsers. Following these findings, we compare the properties shown by the mobile and the desktop browsers. We match keywords on the `userAgent` JavaScript property (see Appendix B) to differentiate the browsers running on a desktop or a laptop (referred to as *desktops*), from the browsers running on a mobile phone (referred to as *mobiles*).

Second, browser fingerprinting is closely dependent on the evolution of web technologies. As new technologies are integrated into browsers, new attributes are accessible, and conversely for removal. Similarly, functionality updates can lead to a change in the fingerprint properties. For example, Kurtz et al. [25] detected an iOS update by the sudden instability of an attribute that provides the iOS version. Following their finding, we verify whether the evolution of web technologies provokes major losses in the properties of the fingerprints.

3.3.4 Accuracy of a simple verification mechanism. We evaluate the accuracy of the verification under fingerprints evolution, considering a simple illustrative verification mechanism. This mechanism counts the identical attributes between the presented fingerprint and the stored fingerprint, and considers the evolution legitimate if this number is above a threshold Θ. The simplicity of this mechanism gives us an idea of the accuracy that can be easily achieved, without having to engineer more complex rules. More elaborate mechanisms can obviously be designed (see Appendix D).

4 EVALUATION OF BROWSER FINGERPRINTS PROPERTIES

In this section, we evaluate the browser fingerprints of our dataset according to the distinctiveness, the stability, and the performance properties. We present here the results on the complete fingerprints, and let Section 5 provide insights on the contribution of the attributes to each property. We show that, by time-partitioning our dataset, our fingerprints provide a unicity rate of more than 81.3%, which is stable through time. However, our fingerprints of mobile browsers are less distinctive than our fingerprints of desktop browsers, with a respective unicity rate of 42% against 84%, considering the time-partitioned datasets. We also show that, on average, a fingerprint has more than 91% of identical attributes between two observations, even when separated by nearly 6 months. About their performance, we show that our fingerprints weigh a dozen of kilobytes, and are collected within seconds. The accuracy of the simple illustrative verification mechanism is close to perfect, as it achieves an equal error rate of 0.61%. This results from most of the consecutive fingerprints coming from a browser having at least 234 identical attributes, whereas most of the fingerprints of different browsers have fewer.

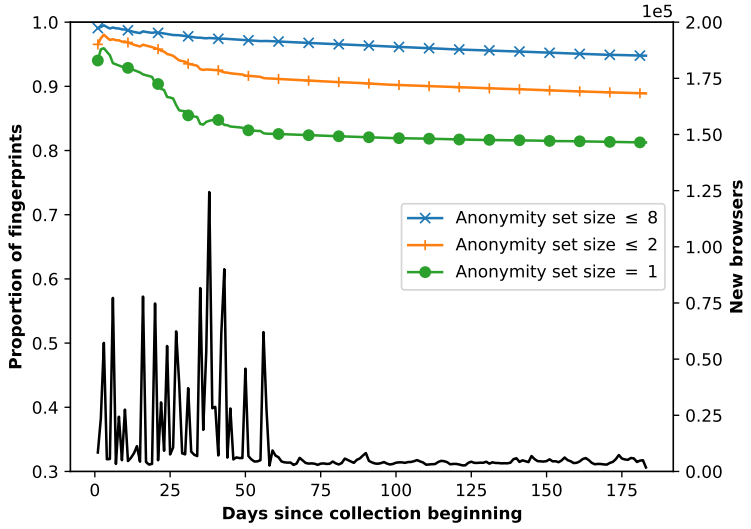

Fig. 2. Anonymity set sizes, and frequency of browser arrivals, through the time-partitioned datasets obtained after each day. The new browsers are displayed in hundreds of thousands.

4.1 Distinctiveness

4.1.1 Overall distinctiveness. Figure 2 presents the size of the anonymity sets alongside the frequency of browser arrival for the time-partitioned datasets. The time-partitioned datasets are designed so that each browser has the last fingerprint observed at the end of the τ-th day. The overall fingerprints have a stable unicity rate of more than 81.3% for the partitioned-datasets, and more than 94.7% of the fingerprints are shared by 8 browsers or fewer. The overall fingerprints comprise those collected from desktop and mobile browsers, but also those of tablets, consoles, and smart TVs. The comparisons are done using fingerprint hashes, resulting in 4 collisions, which we deem negligible.

The anonymity sets tend to grow as more browsers are encountered, due to the higher chances of collision. However, the fingerprints tend to stay in small anonymity sets, as can be seen by the growth of the anonymity sets of size 2 being more important than the growth of the anonymity sets of size 8 or higher. The unicity rate of the time-partitioned datasets (81.3%) is lower than the unicity rate of the complete dataset (81.8%). This is due to browsers having multiple unique fingerprints in the complete dataset, which typically occurs when a browser having a unique web environment is fingerprinted multiple times. Considering the time-partitioned datasets removes this over-counting effect.

New browsers are encountered continually. However, starting from the 60th day, the arrival frequency stabilizes around $5,000$ new browsers per day. Before this stabilization, the arrival frequency is variable, and has major spikes that seem to correspond to events that happened in France. These events could lead to more visits, hence explaining these spikes. For example, the spike on the 38th day corresponds to a live political debate on TV, and the spike on the 43rd correlates with the announcement of a cold snap.

4.1.2 Distinctiveness of desktop and mobile browsers. Figure 3 presents the unicity rate through the time-partitioned datasets for the overall, the mobile, and the desktop browsers. The fingerprints

[12]https://tools.ietf.org/html/rfc7231#section-5.5.3

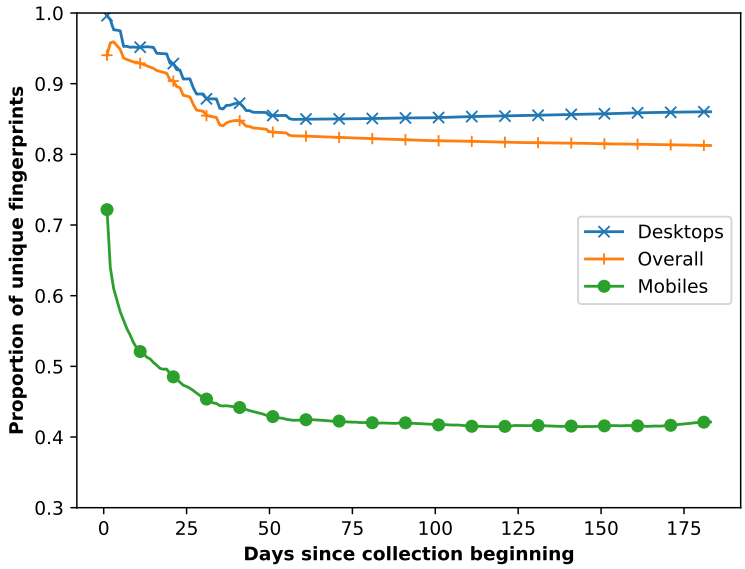

Fig. 3. Unicity rate for the overall, the mobile, and the desktop browsers, through the time-partitioned datasets obtained after each day.

of the mobile browsers are more uniform than the fingerprints of the desktop browsers, with a unicity rate of approximately 42% against 84%, for the time-partitioned datasets. The unicity rate of the desktop browsers slightly increases by 1.04 points from the 60th to the 183th day, from 84.99% to 86.03%. On the contrary, the unicity rate of the mobile browsers slightly decreases by 0.29 points on the same period, from 42.42% to 42.13%.

4.2 Stability

Figure 4 displays the average similarity between the pairs of consecutive fingerprints as a function of the time difference, together with the number of compared pairs for each time difference. The ranges Δ are expressed in days, so that day d on the x-axis represents the fingerprints that are separated by $\Delta = [d; d+1[$ days. We ignore the comparisons of the time ranges that have less than 10 pairs, to have samples of sufficient size without putting too many comparisons aside. We also ignore the bogus comparisons that have a time difference higher than the limit of our experiment (182 days). These two sets of ignored comparisons account for less than 0.03% of each group. The results are obtained by comparing a total of $3,725,373$ pairs of consecutive fingerprints, that include $2,912,860$ pairs for the desktop browsers, and $594,591$ pairs for the mobile browsers. Two consecutive fingerprints are necessarily different as we remove the duplicated consecutive fingerprints (see Section 2.4.3). Considering (f_1, f_2, f_3) the fingerprints collected for a browser, ordered by the time of collection, the set of consecutive fingerprints is $\{(f_1, f_2), (f_2, f_3)\}$. Our stability results are a lower bound, as the consecutive fingerprints are necessarily different (i.e., their similarity is strictly lower than 1).

Our fingerprints are stable, as on average more than 91% of the attributes are expected to not change, considering up to 174 elapsed days (nearly 6 months) between two observations. We assume that the users return more frequently, and otherwise would accept to undergo the account recovery process (see Appendix C). Moreover, the fingerprints of the mobile browsers are generally more stable than the fingerprints of the desktop browsers, as suggests their respective similarity

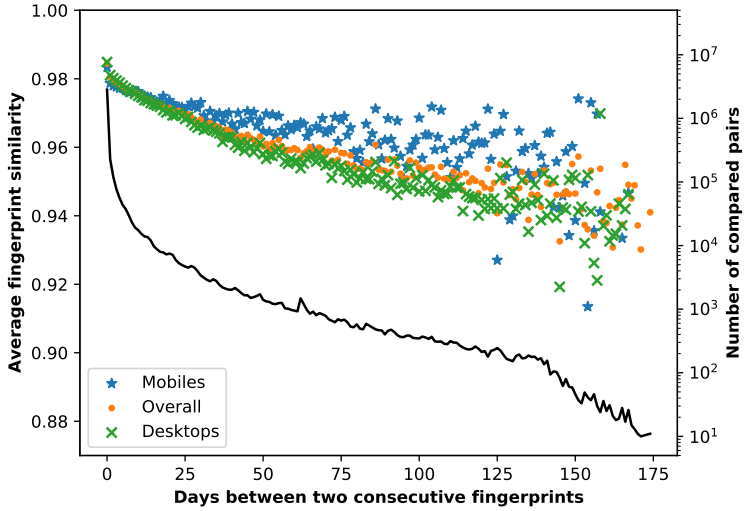

Fig. 4. Average similarity between the pairs of consecutive fingerprints as a function of the time difference, with the number of compared pairs, for the overall, the mobile, and the desktop browsers.

curve. Few attributes of our script are highly unstable. They are discussed in Section 5.1.2. Getting rid of these attributes could reduce the distinctiveness of the fingerprints, but would improve their stability.

4.3 Performance

4.3.1 Time consumption. Figure 5 displays the cumulative distribution of the collection time of our fingerprints in seconds, with the outliers removed. We measure the collection time by the difference between two timestamps, one recorded at the starting of the script, and the other just before sending the fingerprint. Some values are extremely high, taking from several hours to days. They can come from a web page put in background, or accessed after a long time. We limit the results to the fingerprints that take less than 30 seconds to collect, and consider the higher values as outliers. They account for less than 1% of each group.

Half of our fingerprints are collected in less than 2.92 seconds, and the majority (95%) in less than 10.42 seconds. The time to collect the fingerprints is lower for the desktop browsers than for the mobile browsers. Half of the fingerprints of the desktop browsers are collected in less than 2.64 seconds, and the majority (95%) in less than 10.45 seconds. These numbers are respectively of 4.44 seconds and 10.16 seconds for the mobile browsers. The median collection time of our fingerprints is less than the estimated median time taken by web pages to load completely [6], being at 6.5 seconds for the desktop browsers, and 19.7 seconds for the mobile browsers, at the date of May 1, 2020. The mobile devices have generally less computing power than the desktop devices, which can explain the longer collection time. The collection time of the fingerprints of the mobile browsers has less variance than for the desktop browsers, which can be explained by the former having more uniform computing power than the latter. This is supported by the presence in our dataset of desktop browsers running on old systems like Windows Vista or Windows XP.

Our script takes several seconds to collect the attributes composing the fingerprints. However, we stress that this script is purely experimental, and was developed to collect many attributes to get closer to what a fingerprinter can achieve in real-life. The attributes that are longer to collect and that are less distinctive can be removed. For example, our method to detect an advertisement

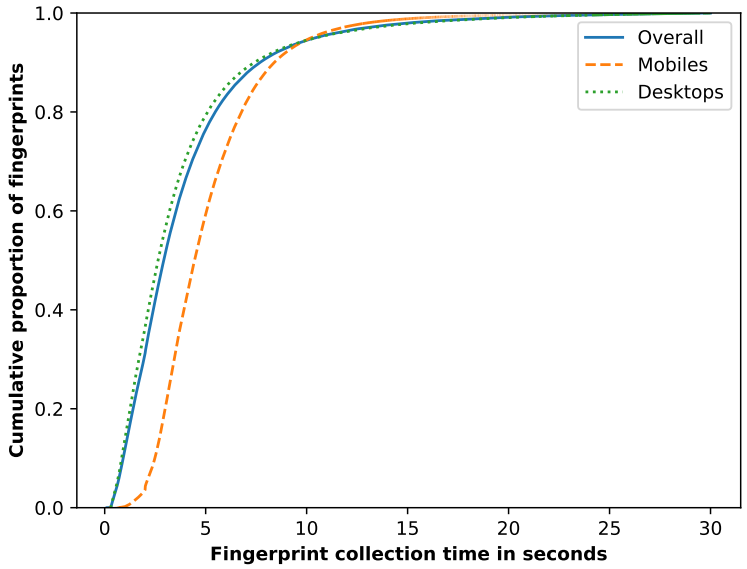

Fig. 5. Cumulative distribution of the collection time of the fingerprints in seconds.

blocker waits a few seconds for a simulated advertisement to be removed, but only provides a Boolean value. We discuss these cases in Section 5.1.3. Our script can also be updated to leverage the most advanced web technologies, like the `OffscreenCanvas` API[13] that migrates the generation of the canvases off the main thread to another thread. More generally, we can use the `Service Workers` API[14] to collect the attributes concurrently in the background, reducing the perceived collection time. These APIs are available on the modern browsers, the collection time should still be monitored for the older browsers.

4.3.2 Memory consumption. Figure 6 displays the cumulative distribution of the size of our fingerprints in bytes, with the outliers removed. Our fingerprints are encoded in UTF-8 (with only ASCII characters), hence one character takes one byte, and the results can be expressed in both units. The canvases are stored as sha256 hashes. The fingerprint sizes comprise the value of the 262 attributes without the metadata fields (e.g., the UID, the timestamp). The average fingerprint size is of $\mu = 7,692$ bytes, and the standard deviation is of $\sigma = 2,294$. We remove 1 fingerprint from a desktop browser considered an outlier due to its size being greater than $\mu + 15 \cdot \sigma$.

The memory consumption takes place on three components: on the client during the buffering of the fingerprints, on the wire during their sending, and on the server during their storage. Half of our fingerprints take less than $7,550$ bytes, 95% less than 12 kilobytes, and all of them less than 22 kilobytes. This is negligible given the current storage and bandwidth capacities. We observe a difference between the fingerprints of mobile and desktop browsers, with 95% of fingerprints weighing respectively less than $8,020$ bytes and $12,082$ bytes. This is due to heavy attributes being lighter on mobiles, like the list of plugins or of mime types that are most of the time empty. We discuss these cases furtherly in Section 5.1.4.

4.3.3 Accuracy of the simple verification mechanism. The accuracy of the simple illustrative verification mechanism is measured according to the following methodology. First, we split our dataset

[13] https://developers.google.com/web/updates/2018/08/offscreen-canvas
[14] https://developer.mozilla.org/en-US/docs/Web/API/Service_Worker_API

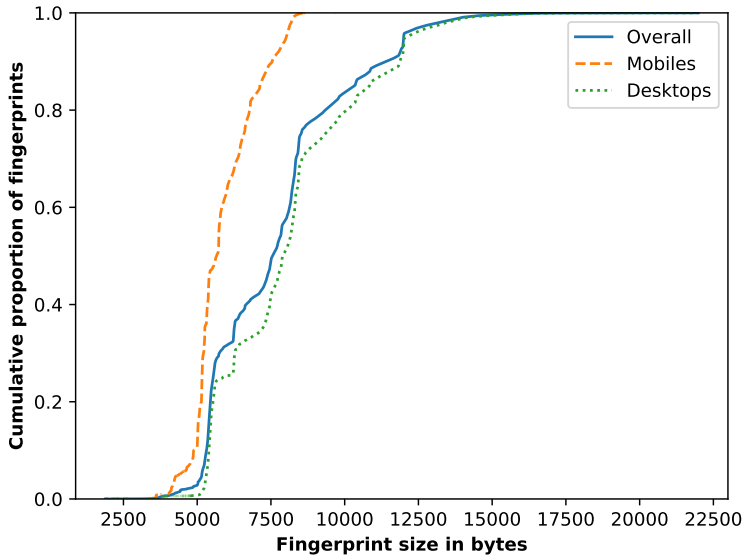

Fig. 6. Cumulative distribution of the fingerprint size in bytes, for the overall, the mobile, and the desktop browsers.

into *six samples*, one for each month. We assume that a user would spend at most one month between two connections, and otherwise would accept to undergo a heavier fingerprint update process. Two sets are afterward extracted from each sample, and are composed of pairs of compared fingerprints, also called *comparisons*. The *same-browser* comparisons are composed of the consecutive fingerprints of each browser, and the *different-browsers* comparisons are composed of two randomly picked fingerprints of different browsers. After constituting the same-browser comparisons for each month, we sample the different-browsers comparisons to have the same size as the same-browser comparisons. The month sampling also helps the different-browsers comparisons to be realistic by pairing fingerprints that are separated by at most one month. Both the two sets of comparisons contain a total of $3,467,289$ comparisons.

Figure 7 displays the distribution of the identical attributes between the same-browser comparisons and the different-browsers comparisons, starting from 34 identical attributes as there are no observed value below. Figure 8 presents a focus that starts from 227 identical attributes, below which there are less than 0.005 of the same-browser comparisons. We can observe that the two sets of comparisons are well separated, as 99.05% of the same-browser comparisons have at least 234 identical attributes, and 99.68% of the different-browsers comparisons have fewer. The different-browsers comparisons have generally a fewer, and a more diverse, number of identical attributes compared to the same-browser comparisons. The different-browsers comparisons have between 34 and 253 identical attributes, with an average of 127.41 attributes, and a standard deviation of 44.06 attributes. The same-browser comparisons have between 72 and 252 identical attributes, with an average of 248.64 attributes, and a standard deviation of 3.91 attributes.

Figure 9 displays the false match rate (FMR), which is the proportion of the same-browser comparisons that are classified as different-browsers comparisons, and the false non-match rate (FNMR), which is the inverse. The displayed results are the average for each number of identical attributes among the six samples. As there are few same-browser comparisons that have less than 234 identical attributes, the FNMR is null until this value. However, after exceeding this threshold,

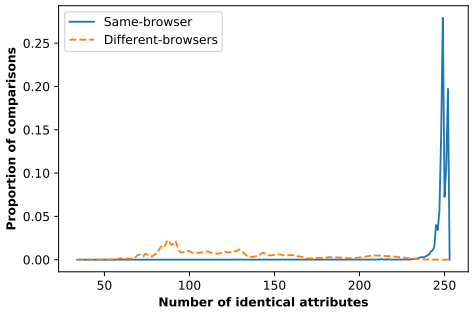

Fig. 7. The number of identical attributes between the same-browser comparisons and the different-browsers comparisons.

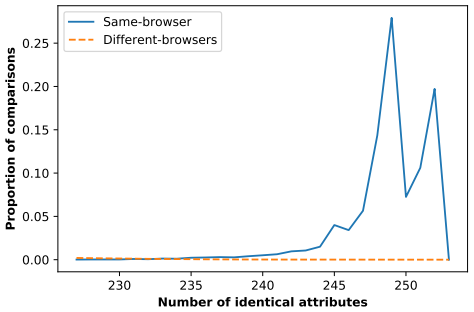

Fig. 8. The number of identical attributes between the same-browser comparisons and the different-browsers comparisons, starting from 227 attributes.

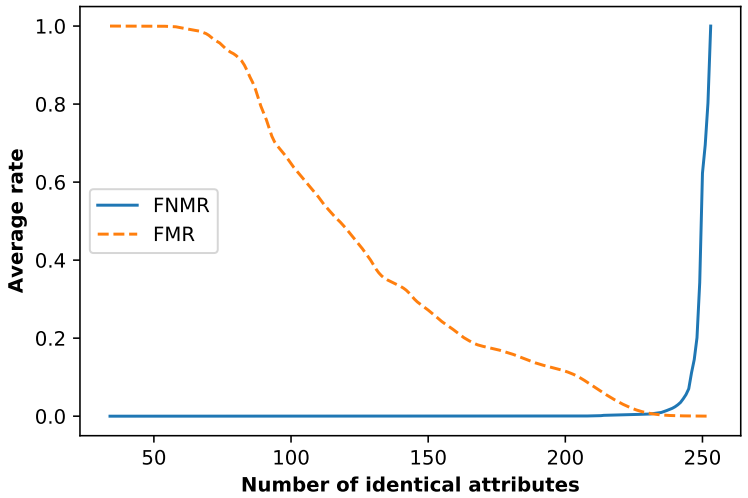

Fig. 9. False match rate (FMR) and false non-match rate (FNMR) given the required number of identical attributes, averaged among the six samples.

the FNMR increases as the same-browser comparisons begin to be classified as different-browsers comparisons. The equal error rate, which is the rate where both the FMR and the FNMR are equal, is of 0.61% and is achieved for 232 identical attributes. Although the verification mechanism does not have a perfect accuracy of 100%, this is acceptable. Indeed, a user getting his browser unrecognized can undergo the fallback authentication process [32, 37]. Moreover, we consider the use of browser fingerprinting as an additional authentication factor, hence the other factors can prevent a falsely recognized browser. Both these cases are expected to rarely occur as can be seen by the low equal error rate.

These results are tied to the distinctiveness and the stability of the fingerprints. Indeed, as more than 94.7% of the fingerprints are shared by less than 8 browsers, two random fingerprints have little chances to match. Moreover, more than 96.64% of the attributes are identical between the consecutive fingerprints of a browser, on average and when separated by less than 31 days. This results in more than 244 identical attributes on average, which is consistent with the 248.64 identical attributes on average among the same-browser comparisons.

4.4 Conclusion

About the distinctiveness, and considering the time-partitioned datasets, our fingerprints provide a unicity rate of more than 81.3%, which is stable on the long-run. Moreover, more than 94.7% of our fingerprints are shared by at most 8 browsers. About the stability, and on average, a fingerprint has more than 91% of its attributes that stay identical between two observations, even when separated by nearly 6 months. About the performance, the majority (95%) of our fingerprints weigh less than 12 kilobytes, and are collected in less than 10.42 seconds. We do not remark any significant loss in the properties offered by our fingerprints through the 6 months of our experiment. However, we fall to the same conclusion as previous studies [42] about the fingerprints of mobile browsers lacking distinctiveness. Their unicity rate in the time-partitioned datasets falls down to approximately 42%. We remark that, in our dataset, the consecutive fingerprints of a browser have at least 234 identical attributes, whereas the majority of the fingerprints of different browsers have fewer. This results in our simple verification mechanism achieving an equal error rate of 0.61%.

5 ATTRIBUTE-WISE ANALYSIS

In this section, we discuss the contribution of the attributes to the properties of the fingerprints, show that most of the attributes are correlated with at least another one (i.e., they provide less than 1 bit of entropy when the other one is known), and focus on the properties of the dynamic attributes (e.g., their collection time). We refer the reader to Appendix A for more information about the implementation of each attribute.

5.1 Contribution of particular attributes

In this section, we discuss the contribution of the attributes to the fingerprint properties of distinctiveness, stability, and performance. We express the stability of the attributes as the proportion of the consecutive fingerprints where the value of the attribute stays identical, that we call the *sameness rate*. Appendix E provides the exhaustive list of the attributes, together with their properties.

5.1.1 Attributes distinctiveness. We measure the distinctiveness of the attributes as the *normalized entropy* for comparability with previous studies. The normalized entropy was proposed by Laperdrix et al. [29] to cope with the problem of comparing the entropy of attributes between fingerprint datasets of dissimilar sizes. The normalized entropy $H_n(X)$ is defined as the ratio of the entropy $H(X)$ of the attribute to the maximum entropy $H_M = \log_2(N)$, with N being the number of fingerprints. Hence, the entropy can be calculated by $H(X) = H_n(X) * H_M$, with $H_M = 21.983$ bits in our case. For reference, an entropy of 1 bit is equivalent to a normalized entropy of 0.045.

Figure 10 displays the cumulative distribution of the normalized entropy, and of the entropy in bits, among the attributes. We have 10% of the attributes that provide a low normalized entropy of less than 0.003, and another 10% that provide a normalized entropy between 0.25 and 0.42. The majority of the attributes (80%) provide a normalized entropy comprised between 0.003 and 0.25. The most distinctive attributes of previous studies [13, 29] also belong to the most distinctive attributes of our study. The three *canvases* are in the 10 most distinctive attributes. Our designed canvas in PNG has a normalized entropy of 0.420, the canvas similar to the canvas presented in the Morellian study [26] has a normalized entropy of 0.385, and the canvas inspired by the AmIUnique study [29] has a normalized entropy of 0.353. The *userAgent* collected from the JavaScript property is more distinctive than its HTTP header counterpart, as they respectively have a normalized entropy of 0.394 and 0.350. Finally, the *list attributes* are also highly distinctive. The list of plugins has a normalized entropy of 0.394, the list of supported mime types has a normalized entropy of 0.311, and the list of fonts has a normalized entropy of 0.305.

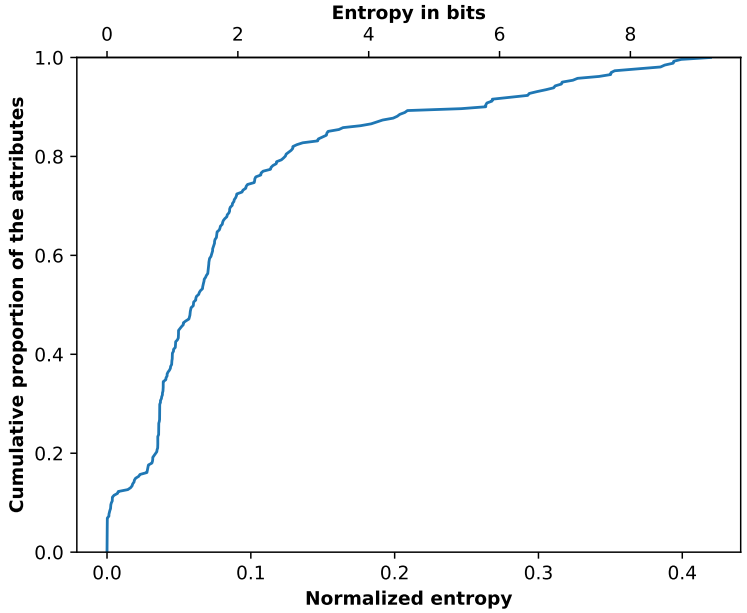

Fig. 10. Cumulative distribution of the normalized entropy, and of the entropy in bits, among the attributes.

Table 2 compares the normalized entropy between the studies Panopticlick, AmIUnique, Hiding in the Crowd, and this study, ranked by the most distinctive attributes of this study. The normalized entropy of our attributes are lower than what is reported in previous studies [13, 29]. This loss of normalized entropy can be explained by the following factors. First, the maximum entropy H_M increases with the number of fingerprints. However, an attribute that has n possibilities (e.g., a Boolean attribute has only two possible values) will have a normalized entropy of at most $\log_2(n)$. As the number N of fingerprints increases, the normalized entropy decreases due to the entropy of the attribute being capped at $\log_2(n)$ whereas the ratio is to $\log_2(N)$. Second, contextual attributes are biased towards the French-population, as described in Section 2.2, hence they provide a lower normalized entropy. For example, the time zone and the `Accept-Language` HTTP header (named Content language in Table 2) provide a respective normalized entropy of 0.008 and 0.124, against 0.198 and 0.351 for the AmIUnique study [29]. The third reason is the evolution of web technologies since the Panopticlick and the AmIUnique study. For example, the list of plugins is less distinctive due to the replacement of plugins[15] by HTML5 functionalities or extensions. Another example is the list of fonts that could be collected through plugins [13], but now it has to be inferred from the size of text elements [15]. These three reasons partly explain the lower normalized entropy that our attributes provide compared to previous studies. We emphasize that although the 17 attributes in common have a lower normalized entropy, we supplement them with more than a hundred attributes, resulting in the fingerprints showing a unicity rate above 80%.

Interestingly, four attributes unreported by the previous large-scale studies [13, 19, 29] are found to be highly distinctive. The `innerHeight` and `outerHeight` properties of the `windows` JavaScript object, mentioned by [40, 49] but without any distinctiveness measure, have a respective normalized entropy of 0.388 and 0.327. The size of bounding boxes was used by [15] as a method of font fingerprinting. From the entropy reported by the authors, we obtain a normalized entropy of 0.761.

[15]https://theblog.adobe.com/adobe-flash-update

Table 2. Comparison of the normalized entropy between the studies Panopticlick (PTC), AmIUnique (AIU), Hiding in the Crowd (HitC), and this study, ranked by the most distinctive attributes of this study.

Attribute	PTC [13]	AIU [29]	HitC [19]	**This study**
Canvas (PNG)	-	0.491	0.407	**0.420**
Canvas (JPG)	-	-	0.391	**0.399**
List of plugins	0.817	0.656	0.452	**0.394**
User agent	0.531	0.580	0.341	**0.350**
List of fonts	0.738	0.497	0.329	**0.305**
Content language	-	0.351	0.129	**0.124**
List of HTTP headers	-	0.249	0.085	**0.095**
Renderer WebGL	-	0.202	0.264	**0.089**
Do Not Track	-	0.056	0.091	**0.085**
Vendor WebGL	-	0.127	0.109	**0.080**
Platform	-	0.137	0.057	**0.068**
Accept	-	0.082	0.035	**0.028**
Content encoding	-	0.091	0.018	**0.019**
Timezone	0.161	0.198	0.008	**0.008**
AdBlock	-	0.059	0.002	**0.002**
Use of local storage	-	0.024	0.002	**0.001**
Use of session storage	-	0.024	0.002	**0.000**
Cookies enabled	0.019	0.015	0.000	**0.000**
Maximum entropy H_M	18.843	16.859	20.980	**21.983**
Fingerprints	470,161	118,934	2,067,942	**4,145,408**

This attribute has a normalized entropy of 0.369 in our dataset. To the best of our knowledge, no previous study use the width and the position of a newly created `div` element as a fingerprinting attribute. However, it is highly distinctive as it achieves a normalized entropy of 0.324 in our dataset. All the mentioned attributes provide a sameness rate higher than 90%, except for the size of bounding boxes that goes down to 47%. However, when looking at its extracted parts, we observe that they have a sameness rate higher than 90%, at the exception to the height of the first bounding box that has a sameness rate of 49.04%. This illustrates the necessity of breaking down some attributes to parts, as removing this part from the original attribute would drastically increase its sameness rate.

5.1.2 Attributes sameness rate. Figure 11 displays the cumulative distribution of the sameness rate among the attributes. Only 6 attributes (2.29% of the attributes) provide a lower sameness rate than 85%, 5.7% of the attributes provide a sameness rate comprised in $[85; 95]\%$, 10.7% provide a sameness rate comprised in $[95, 99]\%$, and more than 80% of the attributes have a sameness rate higher than 99%. The attributes that have a sameness rate lower than 85% are the size of bounding boxes that was previously explained, three extracted attributes derived from the bounding boxes, and two other attributes that we describe here. The `Cache-Control` HTTP header allows the browser to specify the cache policy used during requests, and is the second most unstable attribute with a sameness rate of 70.63%. This is due to the fact that this header is not always sent, and some values contain the `max-age` parameter that can vary between requests. The third most unstable attribute is the WebRTC fingerprinting method that has a sameness rate of 76.46%. This is due to three factors. First, the experimental state of this attribute leads it to be unsupported, undefined,

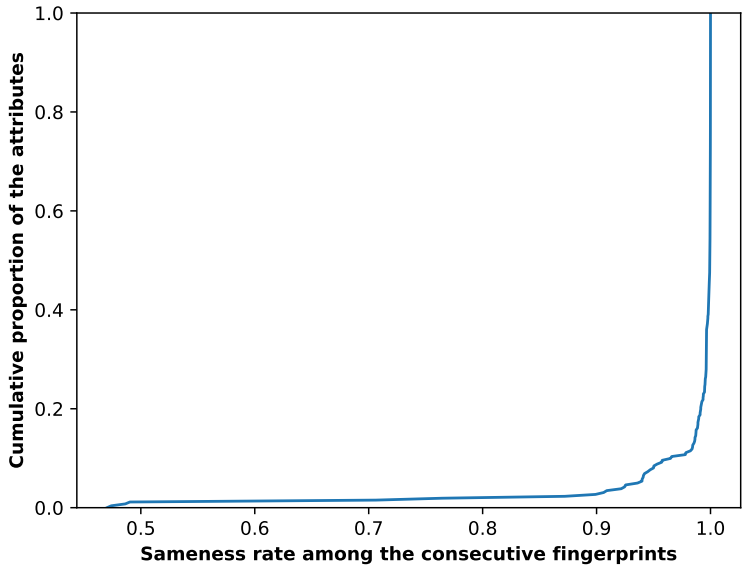

Fig. 11. Cumulative distribution of the sameness rate of the attributes among the consecutive fingerprints.

or erroneous for 75.23% of the observed fingerprints. Then, it contains local IP addresses[16] which can change between two observations. Finally, it is composed by numerous information about an instance of a current WebRTC connection, hence the change of any information changes the value of the whole attribute.

5.1.3 Attributes collection time. Most of the attributes are HTTP headers or JavaScript properties that are collected in a negligible amount of time. Only 33 attributes have a median collection time (MCT) higher than 5ms. Their median collection time is presented in Figure 12, ranked from the slowest to the fastest to collect for the overall browsers. These attributes can be separated into three classes: extension detection, browser component, and media related. These attributes wait for the web page to render, or execute heavy processes (e.g., graphical computation), which explains their high collection time. Note that most of these attributes are collected asynchronously, hence the total collection time of the fingerprint is not their sum.

The first class of attributes that take time to be collected are the methods of *extension detection*. The 9 slowest attributes detect an extension by the changes it brings to the content of the web page [44]. They have a MCT of approximately 2.2 seconds, because of the waiting time before checking the changes on the web page. There is a clear difference between the desktop and the mobile browsers, with a respective MCT of approximately 2 seconds and 3.4 seconds. The 22nd to the 29th slowest attributes detect an extension based on web-accessible resources [41], which consist into checking if an image embarked in an extension is accessible or not. They have a MCT of around 60ms, which is lower than the method that relies on detecting changes brought to the web page.

The second class of attributes that take time to be collected infer the availability of *browser components*. The list of speech synthesis voices is ranked 14th, and has a MCT of 568ms. This is due to the collection method that, for some browsers, requires to be done during an `onvoiceschanged`

[16]Our script hashes these local IP addresses in MD5 directly on the client, see [45] for more information about how the WebRTC API gives access to this information.

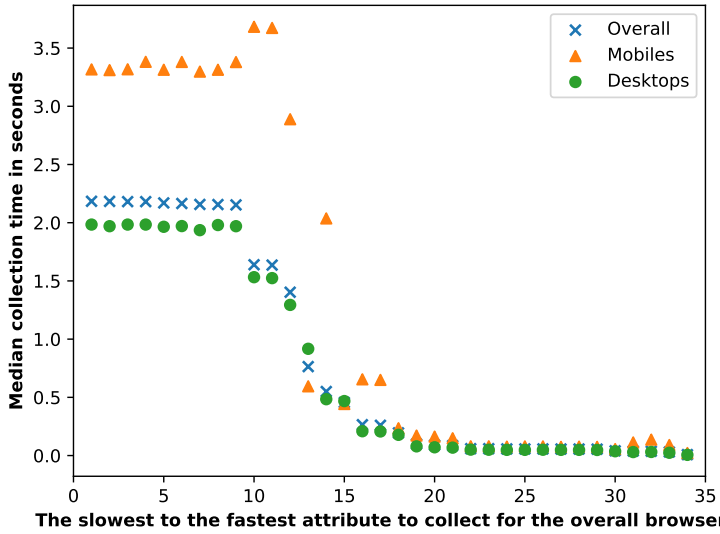

Fig. 12. Median collection time of the 33 attributes that have a median collection time higher than 5ms, ranked from the slowest to the fastest to collect for the overall browsers, in seconds.

event, which takes time to be triggered. The list of fonts, and the inference of the default font, are ranked 15th and 19th with a respective MCT of 471ms and 103ms. This is due to the detection method that measures the size of newly created text boxes [15]. The size of bounding boxes, the browser component colors, and the width and position of a newly created `div` element are ranked 18th, 20th, and 21st, with a MCT ranging from 60ms to 200ms. This is due to their manipulation of the web page that takes time. The WebRTC fingerprinting method is ranked 13th with a MCT of 783ms. This is due to the creation of a dummy connection that is needed to gather information about the WebRTC configuration.

The third class of attributes that take time to be collected generate a media file (e.g., a sound, an image), and are discussed in more detail in Section 5.3. The methods of advanced audio fingerprinting are ranked 10th, 11th, and 12th. Our designed canvases are ranked 16th and 17th, the canvases inspired by the AmIUnique study [29] are ranked 31st and 33rd, and the canvas similar to the canvas of the Morellian study [26] is ranked 32nd.

5.1.4 Attributes size. Most of our attributes have a negligible median size (MS): 137 attributes have a MS of less than 5 bytes, 105 attributes have a MS between 5 bytes and 100 bytes, and 20 attributes have a MS of more than 100 bytes. Figure 13 displays the median size of the 20 heaviest attributes in bytes, ranked from the heaviest to the lightest for the overall browsers. In this section, we discuss these 20 heaviest attributes.

The heaviest attributes are composed of *list attributes* and *verbose textual attributes*, the three heaviest attributes being list attributes. The list of the properties of the `navigator` object is ranked 1st with a MS of 502 bytes, the list of the colors of layout components is ranked 2nd with a MS of 492 bytes, and the list of WebGL extensions is ranked 3rd with a MS of 401 bytes. Examples of verbose textual attributes are the `appVersion` that is ranked 18th with a MS of 107 bytes, and the `userAgent` JavaScript property that is ranked 14th with a MS of 115 bytes, which is more verbose than its HTTP header counterpart that is ranked 19th with a MS of 108 bytes.

On mobile browsers, some list attributes are most of the time empty due to their lack of customization (e.g., plugins are mostly unsupported). They are the list of available synthesis voices

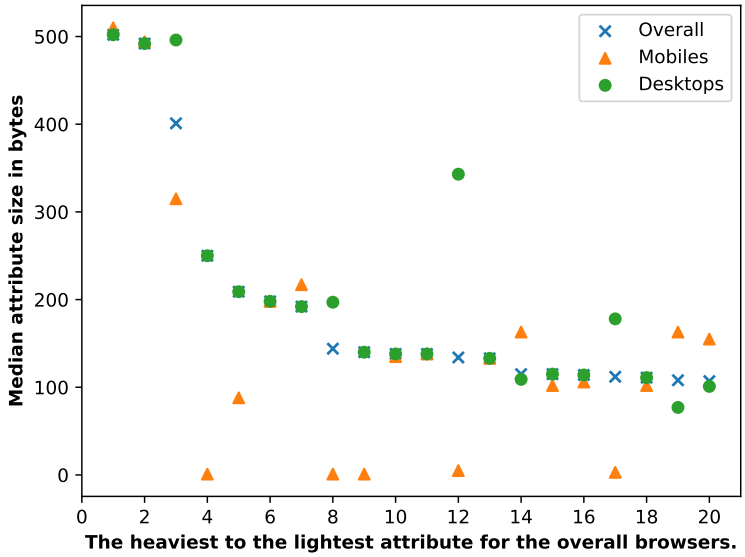

Fig. 13. Median size of the 20 heaviest attributes in bytes, ranked from the heaviest to the lightest for the overall browsers.

that is ranked 4th, the list of the constraints supported by the `mediaDevices` object that is ranked 8th, the list of plugins that is ranked 12th, and the list of supported mime types that is ranked 17th. On the contrary, the verbose attributes are slightly heavier on the mobile browsers, which is explained by the presence of additional information like the device model.

5.2 Correlation between attributes

We can expect to have correlations occurring between the attributes when considering more than 250 attributes. We provide here an overview of the correlation between the attributes, that include the nine dynamic attributes, and refer the reader to Appendix E for insight into the correlation of each attribute.

For comparability with the results of the distinctiveness of the attributes, we express the correlation by the conditional entropy of an attribute a_i when another attribute a_j is known, normalized to the maximum entropy H_M. We call this measure the *normalized conditional entropy* (NCE). It is comprised between 0.0 if knowing a_i allows to completely infer a_j, and the normalized entropy of a_j if knowing a_i provides no information on the value of a_j (i.e., they are independent). We denote V_i the domain of the attribute a_i, and e_v^i the event that the attribute a_i takes the value v. We consider the relative frequency p of the attribute values among the considered fingerprints. The measure of the conditional entropy $\mathrm{H}(a_j|a_i)$ of a_j given a_i is expressed as

$$\mathrm{H}(a_j|a_i) = -\sum_{v \in V_i, w \in V_j} p(e_v^i, e_w^j) \log \frac{p(e_v^i, e_w^j)}{p(e_v^i)} \tag{9}$$

Finally, we normalize the conditional entropy $\mathrm{H}(a_j|a_i)$ by the maximum entropy H_M.

Figure 14 displays the minimum, the average, and the maximum NCE of an attribute when the value of another one of the attributes is known, ordered by the average NCE. We ignore 9 source attributes from which the extracted attributes are derived, and the comparison of an attribute with itself. These cases are irrelevant, as the extracted attributes are completely correlated with their

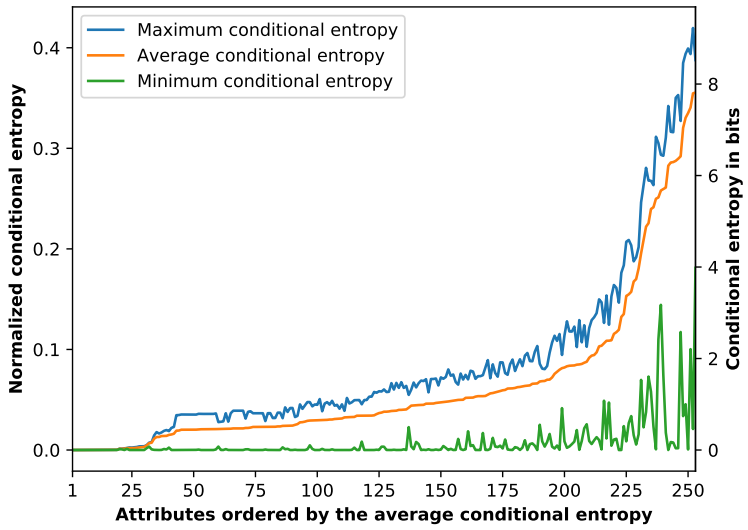

Fig. 14. Minimum, average, and maximum normalized conditional entropy of an attribute when the value of another attribute is known, ordered by the average normalized conditional entropy.

source attribute, and an attribute is completely correlated with itself. We obtain a total of 253 attributes. We observe that the maximum NCE of an attribute is always equal to the normalized entropy of this attribute, due to the `cookieEnabled` property that is always `true`. Hence, it provides a null entropy, and knowing its value does not provide any information on the value of the other attributes. We recall that the maximum entropy is $H_M = 21.983$ bits, and that a conditional entropy of 1 bit is equivalent to a NCE of 0.045.

We can see three parts in this figure. First, 19 attributes have a low normalized entropy of less than 10^{-3}, and the NCE when knowing another attribute is at most as much. Few different values have been observed for these attributes. Then, 194 attributes have an average NCE between 10^{-3} and 10^{-1}. The minimum NCE of these attributes is near 0.0, hence there exists another attribute that can be used to efficiently infer their value. Finally, 40 attributes have an average NCE higher than 10^{-1}, and generally have a strictly positive minimum NCE. These attributes help to efficiently distinguish browsers, and are less correlated to other attributes.

The minimum normalized conditional entropy (MNCE) is an interesting indicator of the efficiency to infer the value of an attribute if the value of another attribute is known. We have 49 attributes that have a null MNCE, which can completely be inferred when another attribute is known. Moreover, 192 attributes have a MNCE comprised in the range $]0; 0.045]$, hence knowing the value of another attribute helps to infer their value, but not completely. Finally, 12 attributes have a MNCE higher than 0.045, which is equivalent to having a minimum conditional entropy higher than 1 bit. They are displayed in Table 3. They consist of highly distinctive attributes (see Section 5.1.1) that concern the size of the screen (e.g., W.screenX) or the window (e.g., W.innerHeight), browser components (e.g., fonts, plugins), and verbose information about the browser (e.g., N.userAgent) or about external components (e.g., WebRTC fingerprinting).

5.3 Focus on dynamic attributes

Previous studies highlight the possibility to fingerprint browsers by rendering media files inside the browser, given instructions that are provided by the fingerprinter (e.g., WebGL canvas [33],

Table 3. The attributes that have a minimum normalized conditional entropy (MNCE) higher than 0.045, or a minimum conditional entropy higher than 1 bit, together with their MNCE and their minimum conditional entropy in bits. W refers to the `window` JavaScript object, N refers to the `navigator` object, WG refers to an initialized WebGL context, and [...] is a truncated part.

Attribute	MNCE	Minimum conditional entropy
W.innerHeight	0.181	3.98
WebRTC fingerprinting	0.144	3.17
W.outerHeight	0.117	2.58
Presence of fonts	0.110	2.42
N.plugins	0.100	2.21
W.outerWidth	0.074	1.63
WG.[...].UNMASKED_RENDERER_WEBGL	0.073	1.61
Height of first bounding box	0.070	1.53
S.availHeight	0.058	1.27
W.screenY	0.049	1.08
W.screenX	0.047	1.04
N.userAgent	0.046	1.00

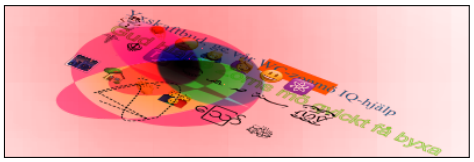

Fig. 15. Our designed HTML5 canvas in PNG format.

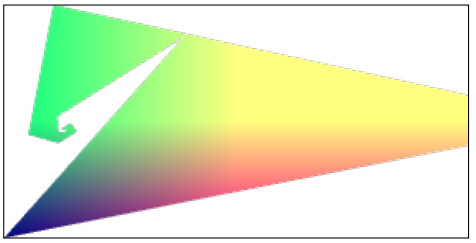

Fig. 16. Our designed WebGL canvas in PNG format.

HTML5 canvas [10], audio fingerprinting [36]). Later on, these attributes were integrated within challenge-response mechanisms [26, 38] that mitigate replay attacks. We call these attributes the *dynamic attributes*, and include nine of them in our script: five HTML5 canvases, three audio fingerprinting methods, and a WebGL canvas. To the best of our knowledge, no study evaluate the properties of several dynamic attributes on a browser population, together with the evaluation of various set of instructions. In this section, we seek to fill this gap, and focus on the properties offered by the nine dynamic attributes of our fingerprinting script.

5.3.1 HTML5 canvas. The HTML5 canvas consists into asking the browser to draw an image by using the canvas API, within the two-dimensional context called 2d. This method is already studied in several works, whether it is about its efficacy [10, 19, 29], its use on the web [14, 30], or the distinctiveness provided by various sets of instructions [26]. However, to the best of our knowledge, no study evaluates the different properties (e.g., distinctiveness, stability, collection time) offered by various sets of complex instructions (i.e., mixes of texts, emojis, mathematical curves, drawn using different colors). We seek to fill this gap, and evaluates the properties of five HTML5 canvases, generated following three sets of instructions and in two image formats (PNG and JPEG).

We name the canvases given the set of instructions used and their format. We call *AmIUnique canvas* the canvas generated by the set of instructions inspired by the AmIUnique[17] study [29], and extracted in PNG format. We call *Morellian canvas* the canvas generated by the set of instructions that is similar to the Morellian study [26], which is extracted in PNG and JPEG formats. We call *custom canvas* the canvas generated by a set of instructions that we designed, which is extracted in PNG and JPEG formats. We collect the canvases by using the `toDataURL` function. The `quality` parameter of this function goes from 0.0 to 1.0, and allow us to control the level of compression of the JPEG versions. As we seek to compare the PNG canvases that are compressed without loss, with their JPEG counterparts by using a high level of compression, we set the `quality` to 0.1 for the JPEG versions. An example of the custom canvas in PNG format is displayed in Figure 15. Examples of the AmIUnique canvas, and of the Morellian canvas, are provided in Appendix A.8.

We observe that canvases are less distinctive in JPEG format than in PNG format, but are more stable. For example, the custom canvas has a normalized entropy of 0.420 when exported in PNG, against 0.399 for the JPEG version. However, the PNG version has a sameness rate of 92.16%, against 93.59% for the JPEG version. These differences are due to distinct images in PNG format ending up the same after the lossy compression of the JPEG format. Acar et al. [1] assumes that a canvas generated in a lossy compression format as JPEG is not an indicator of a fingerprinting attempt. Although the JPEG version provides a lower distinctiveness than the PNG version, we show that it is still highly distinctive when generated from a complex set of instructions. Indeed, it is the second most distinctive attribute among ours (see Section 5.1.1). The time overhead induced by the additional extraction in JPEG format is negligible. For example, the custom canvas has a median collection time (MCT) increased by 3ms for the JPEG version, compared to the PNG version that has a MCT of 266ms. We do not account the size differences as both formats are hashed, and the resulting hashes weigh 64 bytes. The PNG and the JPEG versions are also highly correlated, as knowing the value in the PNG format of the custom canvas leaves a normalized conditional entropy (NCE) of 5.28×10^{-4} on the value of the JPEG format, whereas the inverse provides a NCE of 0.021.

The properties of the three PNG canvases differ, with the custom canvas being the most distinctive. First, the Morellian canvas is an enhanced version of the AmIUnique canvas, with additional curves. This enhancement provides an increase of the distinctiveness, with a normalized entropy of 0.385 for the Morellian canvas against 0.353 for the AmIUnique canvas, but also a decrease of the sameness rate, with a respective sameness rate of 94.71% against 98.64%. Then, the custom canvas is more complex than the Morellian canvas, as it includes several emojis, two strings including Swedish letters, many overlapping ellipses, a color gradient background, and a rotation of all these elements if the functionality is available. These improvements provide a higher normalized entropy of 0.420, but also a lower sameness rate of 92.16%. The main drawback of adding more complexity to the custom canvas is the temporal cost, as it has a MCT of 266ms, against 37ms for the Morellian canvas, and 32ms for the AmIUnique canvas. However, it represents less than 10% of the total median collection time of the fingerprints, which is 2.92 seconds, and less than 5% of the median loading time of a web page on a desktop browser [6], which is 6.5 seconds. Finally, knowing the value of the custom canvas leaves less variability on the value of the two other canvases than the opposite. When knowing the value of the Morellian canvas or of the AmIUnique canvas, the custom canvas has a respective NCE of 0.079 and 0.103. However, knowing the value of the custom canvas, the Morellian canvas has a NCE of 0.044, and the AmIUnique canvas has a NCE of 0.037. To conclude, adding more instructions for the canvas drawing usually provides

[17]Contrarily to the AmIUnique study [29], we only have one sentence and one smiling face emoji instead of two, we do not draw a colored rectangle, and the sentence is drawn using a color gradient.

more distinctiveness, as each instruction can induce a difference between browsers, at the cost of additional computation time. Moreover, each additional instruction can constitute an instability factor, that negatively impacts the sameness rate.

5.3.2 WebGL canvas. The WebGL canvas is an image that is also drawn using the HTML5 API, but within the `webgl` or `webgl2` contexts. These contexts use the WebGL library[18] that leverages hardware accelerations to render and manipulate two-dimensional graphics, but also three-dimensional scenes. Canvas fingerprinting was first introduced by Mowery et al. [33] by using the `webgl` context, but afterwards most studies focused on the `2d` context [2, 7, 10, 48]. This can result from the unreliability of the method encountered by Laperdrix et al. [29], for which Cao et al. [11] proposed a remedy by setting specific parameters.

Our WebGL canvas consists into sequential triangles with a color gradient, and is simpler than our designed HTML5 canvas. It provides a normalized entropy of 0.263 against 0.420 for the custom HTML5 canvas, is more stable that the other canvases with a sameness rate of 98.97%, and has a median collection time of 29ms against 266ms for the custom HTML5 canvas. Figure 16 displays an example of our WebGL canvas.

5.3.3 Web Audio. Web Audio fingerprinting was discovered by Englehardt et al. [14] when assessing the use of web tracking methods on the web, and more recently studied thoroughly by Queiroz et al. [36]. It consists into processing audio data into the browser, and getting the rendered result. Similarly to canvases, this processing relies on software and hardware components, and variations occur between different component stacks. It works by creating an `Audio Context`, which in our case is the `OfflineAudioContext`[19], into which we manipulate `Audio Nodes`. The `Audio Nodes` are of three types. Source nodes generate an audio signal (e.g., from a microphone or an audio stream), destination nodes send the signal to be rendered (e.g., by speakers), and manipulation nodes manipulate the signal (e.g., increasing the volume). These nodes are linked together to form a network that goes from source nodes to destination nodes, and passes through manipulation nodes. We refer the reader to [36] for a broader description of the main audio nodes.

We have three audio fingerprinting attributes. The *audio fp simple* (AFS) attribute relies on a simple process. The attributes *audio fp advanced* (AFA) and *audio fp advanced frequency data* (AFA-FD) rely on a more advanced process. Their concrete implementation is described in Appendix A. The most distinctive audio attribute is the AFA-FD that has a normalized entropy of 0.161, followed by the AFS (0.153), and the AFA (0.147). They all have a sameness rate of approximately 95%. Their values are the string representations of floating-point numbers, hence they have a median size of 17 bytes. The simple process has a median collection time (MCT) of 1.4 seconds, and the advanced process has a MCT of 1.7 seconds. The AFA-FD is collected from the advanced version, and has a negligible 4ms additional MCT compared to the AFA.

6 RELATED WORKS

In this section, we present related works about the use of browser fingerprinting for authentication. In addition, Section 2.5 compares our dataset with the previously studied large-scale datasets [13, 19, 29, 35], and Section 5.1.1 compares the distinctiveness of our attributes with their distinctiveness reported in previous studies [13, 19, 29].

We stress that most studies about browser fingerprinting focus on their use for identification. The use of browser fingerprints for identification and for authentication differ in the objective

[18] https://get.webgl.org

[19] The advantage of using OfflineAudioContext is the manipulation of audio signal without playing any sound on a real audio output.

when given a presented fingerprint f. In identification, we seek to find the identity (e.g., an account, an advertisement profile) to which f belongs, among a pool of N candidate identities. If f is unrecognized, it is associated to a new identity that is then added to the candidate identities. Hence, in identification we operate a 1-N comparison. On the contrary, in authentication the identity is already given (e.g., the claimed account), and we seek to verify that f legitimately belongs to this identity. Hence, in authentication we operate a 1-1 comparison[20]. The studies about the use of browser fingerprints for identification usually evaluates the threat posed to privacy by browser accessible information [13, 15, 19, 29, 33, 36, 44], propose counter-measures to avoid being tracked by this technique [7, 27], or measure their usage in the wild [1, 14].

The first works about the use of browser fingerprinting for authentication focused on its use for continuous authentication [34, 43, 47]. The objective is to verify that the authenticated session is not hijacked. These studies focus on the integration of browser fingerprinting into an authentication mechanism, and provide few insights about the properties of the fingerprints (e.g., only [43] analyze fingerprints, and focus on their classification efficacy). On the contrary, we identify several properties used to evaluate authentication factors, against which we evaluate real-life browser fingerprints, and explain the results by highlighting the contribution of single attributes. Moreover, these studies only consider a small fraction of the hundreds of available attributes, that do not include the dynamic attributes that can be used in a challenge-response mechanism to thwart replay attacks. We include more than 200 attributes – including 9 dynamic attributes – for which we provide the implementation and the properties (e.g., number of distinct values, normalized entropy).

Alaca et al. [4] provided a classification of the fingerprinting attributes, given properties that include the stability, the distinctiveness, and the resource usage. They qualitatively estimate the stability, the distinctiveness, and the resource usage of the attributes given their nature. For some attributes, they provide the entropy measured in previous studies, and acknowledge that further study is needed on this subject. We emphasize that comparing the entropy of the attributes between datasets of different sizes leads to comparability problems as explained by [29] (e.g., the attribute of a dataset of N fingerprints provides at most an entropy of $\log_2(N)$ bits). We provide the qualitative measure of these properties on our attributes from the analysis of real-life fingerprints, and also evaluate the accuracy of a simple verification mechanism. About the attributes considered by [4], they include attributes that require interactions from the user, like the geolocation, or that are related to the network protocol, like the TCP/IP stack fingerprinting that analyzes the response to specially crafted messages. Moreover, their classification leads to different attributes being grouped under a single designation, like the "major software and hardware details" that englobes the attribute family of the JavaScript properties that we describe in Appendix A. The attributes of this family can show diverse properties, like the `UserAgent` that provides a normalized entropy of 0.394 and a sameness rate of 0.98%, whereas the `screenX` property of the `window` object provides a normalized entropy of 0.125 and a sameness rate of 0.93%. We show that more than 200 attributes are easily accessible (i.e., they require few lines of JavaScript), and do not require any interaction from the user, which would reduce the usability (e.g., the user being asked permissions several times). Moreover, we provide the exhaustive list of the attributes, with their concrete implementation and their properties. We also evaluate the properties of distinctiveness, stability, and resource usage on the complete fingerprints that combine as many attributes.

[20]If users register n browsers to their account, as described in Appendix C, f is compared to the fingerprint of these n browsers that are already identified. Users are expected to register less than ten devices [39], hence n is expected to be smaller than N (e.g., a website having a thousand accounts registered). It would also be possible to find the right fingerprint among the n possibilities by leveraging the UserAgent to recognize which browser the user is using.

Markert et al. [32] recently presented a *work in progress* about the long-term analysis of fallback authentication methods. They plan to measure the recovery rate of the evaluated methods, after an elapsed time of 6, 12, and 18 months. These methods include the browser fingerprinting, for which they acknowledge that "not much about browser fingerprinting-based security systems is known".

Rochet et al. [38] and Laperdrix et al. [26] proposed challenge-response mechanisms that rely on dynamic attributes, especially the HTML5 canvas. The instructions provided to the canvas API are changed on each authentication, making the generated image vary on each fingerprinting. Trivial replay attacks then fails as the awaited canvas image changes each time. In this study, we propose the evaluation of nine dynamic attributes that include five HTML5 canvases, one WebGL canvas, and three audio fingerprinting methods.

7 CONCLUSION

In this study, we conduct the first large-scale empirical study of the properties of browser fingerprints when used for web authentication. We make the link between the digital fingerprints that distinguish Humans, and the browser fingerprints that distinguish browsers, to evaluate the latter according to properties inspired by biometric authentication factors. We formalize and evaluate the properties for the browser fingerprints to be usable and practical in an authentication context. They include the distinctiveness of the fingerprints, their stability, their collection time, their size, the loss of efficacy among browser types, and the accuracy of a simple illustrative verification mechanism. We evaluate these properties on a real-life large-scale fingerprint dataset collected over a period of 6 months, that contains $4,145,408$ fingerprints composed of 216 attributes. The attributes include nine dynamic attributes, which are used in state-of-the-art authentication mechanisms to mitigate replay attacks. We thoroughly describe the preprocessing steps to prepare the dataset, and the browser population that, contrary to most of previous studies, is not biased towards technically-savvy users. About the results, we show that our browser fingerprints provide a unicity rate above 81.3%, considering our time-partitioned datasets. A rate that is stable over the 6 months. Moreover, more than 94.7% of the fingerprints are shared by at most 8 browsers. However, we observe a loss of distinctiveness from mobile browsers that show a lower unicity rate of 42%. About the stability, and on average, more than 91% of the attributes are identical between two observations of the fingerprint of a browser, even when they are separated by nearly 6 months. About the memory and the time consumption of our fingerprints, we show that they weigh a dozen of kilobytes, and take a few seconds to collect. Our simple verification mechanism achieves an equal error rate of 0.61%, thanks to most of the consecutive fingerprints of a browser having at least 234 identical attributes, whereas most of the fingerprints of different browsers have fewer. To better comprehend the results on the complete fingerprints, we evaluate the contribution of the attributes to each fingerprint property. We show that, although the attributes show a lower distinctiveness compared to previous studies, 10% of the attributes provide a normalized entropy higher than 0.25. Moreover, four attributes unconsidered by the previous large-scale studies are part of the most distinctive of our attributes. They concern the size and position of elements of the browser interface. About the stability, 85% of the attributes stay identical between 99% of the pairs of consecutive fingerprints coming from the same browser. Few attributes consume a high amount of resources, as only 33 attributes take more than 5ms to collect, and only 20 attributes weigh more than 100 bytes. When looking at the correlation between the attributes, we find that 49 attributes can completely be inferred when knowing the value of another attribute. We remark that the dynamic attributes are part of the most time-consuming attributes, but also the most distinctive attributes. We also show the importance of the set of instructions, as it influences both the distinctiveness and the stability of the generated values (e.g., the canvas image). We conclude that

the browser fingerprints, obtained from the combination of the studied browser population and the large surface of fingerprinting attributes, carry the promise to strengthen web authentication mechanisms.

ACKNOWLEDGMENTS

We would like to thank Benoît Baudry and David Gross-Amblard for their valuable comments.

REFERENCES

[1] Gunes Acar, Christian Eubank, Steven Englehardt, Marc Juarez, Arvind Narayanan, and Claudia Diaz. 2014. The Web Never Forgets: Persistent Tracking Mechanisms in the Wild. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security (CCS '14)*. ACM, New York, NY, USA, 674–689. https://doi.org/10.1145/2660267.2660347

[2] Nasser Mohammed Al-Fannah and Wanpeng Li. 2017. Not All Browsers Are Created Equal: Comparing Web Browser Fingerprintability. (March 2017). arXiv:1703.05066 http://arxiv.org/abs/1703.05066

[3] Nasser Mohammed Al-Fannah, Wanpeng Li, and Chris J Mitchell. 2018. Beyond cookie monster amnesia: Real world persistent online tracking. In *Proceedings of ISC 2018*. Springer-Verlag.

[4] Furkan Alaca and P. C. van Oorschot. 2016. Device Fingerprinting for Augmenting Web Authentication: Classification and Analysis of Methods. In *Proceedings of the 32Nd Annual Conference on Computer Security Applications (ACSAC '16)*. ACM, New York, NY, USA, 289–301. https://doi.org/10.1145/2991079.2991091

[5] Nampoina Andriamilanto, Tristan Allard, and Gaëtan Le Guelvouit. [n.d.]. "Guess Who?" Large-Scale Data-Centric Study of the Adequacy of Browser Fingerprints for Web Authentication. In *Innovative Mobile and Internet Services in Ubiquitous Computing* (Cham, 2021) *(Advances in Intelligent Systems and Computing)*, Leonard Barolli, Aneta Poniszewska-Maranda, and Hyunhee Park (Eds.). Springer International Publishing, 161–172. https://doi.org/10.1007/978-3-030-50399-4_16

[6] The HTTP Archive. 2020. *Median Loading Time of Web Pages.* https://httparchive.org/reports/loading-speed#ol accessed 2020-06-16.

[7] Peter Baumann, Stefan Katzenbeisser, Martin Stopczynski, and Erik Tews. 2016. Disguised Chromium Browser: Robust Browser, Flash and Canvas Fingerprinting Protection. In *Proceedings of the 2016 ACM on Workshop on Privacy in the Electronic Society* (New York, NY, USA, 2016) *(WPES '16)*. ACM, 37–46. https://doi.org/10.1145/2994620.2994621

[8] Joseph Bonneau. 2012. The science of guessing: analyzing an anonymized corpus of 70 million passwords. In *Security and Privacy (SP), 2012 IEEE Symposium on*. IEEE, 538–552.

[9] Joseph Bonneau, Cormac Herley, Paul C. van Oorschot, and Frank Stajano. 2015. Passwords and the Evolution of Imperfect Authentication. *Commun. ACM* 58, 7 (June 2015), 78–87. https://doi.org/10.1145/2699390

[10] Elie Bursztein, Artem Malyshev, Tadek Pietraszek, and Kurt Thomas. 2016. Picasso: Lightweight Device Class Fingerprinting for Web Clients. (2016). https://research.google.com/pubs/pub45581.html

[11] Yinzhi Cao, Song Li, and Erik Wijmans. 2017. (Cross-)Browser Fingerprinting via OS and Hardware Level Features. https://doi.org/10.14722/ndss.2017.23152

[12] Anupam Das, Joseph Bonneau, Matthew Caesar, Nikita Borisov, and XiaoFeng Wang. 2014. The Tangled Web of Password Reuse. In *NDSS*, Vol. 14. 23–26.

[13] Peter Eckersley. 2010. How unique is your web browser?. In *Privacy Enhancing Technologies*.

[14] Steven Englehardt and Arvind Narayanan. 2016. Online Tracking: A 1-million-site Measurement and Analysis. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS '16)*. ACM, New York, NY, USA, 1388–1401. https://doi.org/10.1145/2976749.2978313

[15] David Fifield and Serge Egelman. 2015. Fingerprinting web users through font metrics. 107–124.

[16] Marco Gamassi, Massimo Lazzaroni, Mauro Misino, Vincenzo Piuri, Daniele Sana, and Fabio Scotti. 2005. Quality assessment of biometric systems: a comprehensive perspective based on accuracy and performance measurement. 54, 4 (2005), 1489–1496.

[17] Tom Goethem, Wout Scheepers, Davy Preuveneers, and Wouter Joosen. 2016. Accelerometer-Based Device Fingerprinting for Multi-factor Mobile Authentication. In *Proceedings of the 8th International Symposium on Engineering Secure Software and Systems - Volume 9639 (ESSoS 2016)*. Springer-Verlag New York, Inc., New York, NY, USA, 106–121. https://doi.org/10.1007/978-3-319-30806-7_7

[18] Maximilian Golla, Theodor Schnitzler, and Markus Dürmuth. 2018. "Will Any Password Do?" Exploring Rate-Limiting on the Web. In *Who are you? Adventures in Authentication Workshop 2018* (Baltimore, MD, USA). USENIX Association.

[19] Alejandro Gómez-Boix, Pierre Laperdrix, and Benoit Baudry. 2018. Hiding in the Crowd: an Analysis of the Effectiveness of Browser Fingerprinting at Large Scale. In *The Web Conference*.

[20] Marti A. Hearst, Susan T Dumais, Edgar Osuna, John Platt, and Bernhard Scholkopf. 1998. Support vector machines. *Intelligent Systems and their applications* (1998).

[21] Dan Jurafsky and James H. Martin. 2008. *Speech and Language Processing*.

[22] Nian-hua KANG, Ming-zhi CHEN, Ying-yan FENG, Wei-ning LIN, Chuan-bao LIU, and Guang-yao LI. 2017. Zero-Permission Mobile Device Identification Based on the Similarity of Browser Fingerprints. *DEStech Transactions on Computer Science and Engineering* (July 2017). https://doi.org/10.12783/dtcse/cst2017/12531

[23] Soroush Karami, Panagiotis Ilia, Konstantinos Solomos, and Jason Polakis. 2020. Carnus: Exploring the Privacy Threats of Browser Extension Fingerprinting. https://www.ndss-symposium.org/ndss-paper/carnus-exploring-the-privacy-threats-of-browser-extension-fingerprinting

[24] Amin Faiz Khademi, Mohammad Zulkernine, and Komminist Weldemariam. 2015. An Empirical Evaluation of Web-Based Fingerprinting. 32, 4 (2015), 46–52. https://doi.org/10.1109/MS.2015.77

[25] Andreas Kurtz, Hugo Gascon, Tobias Becker, Konrad Rieck, and Felix Freiling. 2016. Fingerprinting mobile devices using personalized configurations. *Proceedings on Privacy Enhancing Technologies* 2016, 1 (2016), 4–19.

[26] Pierre Laperdrix, Gildas Avoine, Benoit Baudry, and Nick Nikiforakis. 2019. Morellian Analysis for Browsers: Making Web Authentication Stronger With Canvas Fingerprinting. In *16th Conference on Detection of Intrusions and Malware & Vulnerability Assessment* (2019).

[27] Pierre Laperdrix, Benoit Baudry, and Vikas Mishra. 2017. FPRandom: Randomizing core browser objects to break advanced device fingerprinting techniques. In *ESSoS 2017 - 9th International Symposium on Engineering Secure Software and Systems*. Bonn, Germany, 17. https://hal.inria.fr/hal-01527580

[28] Pierre Laperdrix, Nataliia Bielova, Benoit Baudry, and Gildas Avoine. 2020. Browser Fingerprinting: A Survey. 14, 2 (2020), 8:1–8:33. https://doi.org/10.1145/3386040

[29] Pierre Laperdrix, Walter Rudametkin, and Benoit Baudry. 2016. Beauty and the Beast: Diverting modern web browsers to build unique browser fingerprints. In *37th IEEE Symposium on Security and Privacy*.

[30] H. Le, F. Fallace, and P. Barlet-Ros. 2017. Towards accurate detection of obfuscated web tracking. In *2017 IEEE International Workshop on Measurement and Networking (M N)* (2017-09). 1–6. https://doi.org/10.1109/IWMN.2017.8078365

[31] Davide Maltoni, Dario Maio, Anil K. Jain, and Salil Prabhakar. 2003. *Handbook of Fingerprint Recognition* (1 ed.). Springer-Verlag. https://doi.org/10.1007/b97303

[32] Philipp Markert, Maximilian Golla, Elizabeth Stobert, and Markus Dürmuth. 2020. A Comparative Long-Term Study of Fallback Authentication - Work in Progress. https://www.ndss-symposium.org/ndss-paper/auto-draft-30/

[33] Keaton Mowery and Hovav Shacham. 2012. Pixel Perfect: Fingerprinting Canvas in HTML5. In *Proceedings of W2SP 2012*. 1–12.

[34] Davy Preuveneers and Wouter Joosen. 2015. SmartAuth: Dynamic Context Fingerprinting for Continuous User Authentication. In *Proceedings of the 30th Annual ACM Symposium on Applied Computing (SAC '15)*. ACM, New York, NY, USA, 2185–2191. https://doi.org/10.1145/2695664.2695908

[35] Gaston Pugliese, Christian Riess, Freya Gassmann, and Zinaida Benenson. 2020. Long-Term Observation on Browser Fingerprinting: Users' Trackability and Perspective. 2 (2020), 558–577.

[36] Jordan S. Queiroz and Eduardo L. Feitosa. 2019. A Web Browser Fingerprinting Method Based on the Web Audio API. (22 Jan. 2019). https://doi.org/10.1093/comjnl/bxy146

[37] Nils Quermann, Marian Harbach, and Markus Dürmuth. 2018. The state of user authentication in the wild. In *Who are you* (2018).

[38] Florentin Rochet, Kyriakos Efthymiadis, FranÃ§ois Koeune, and Olivier Pereira. 2019. SWAT: Seamless Web Authentication Technology. In *The World Wide Web Conference* (New York, NY, USA, 2019) *(WWW '19)*. ACM, 1579–1589. https://doi.org/10.1145/3308558.3313637

[39] Bardia Safaei, Amir Mahdi Monazzah, Milad Bafroei, and Alireza Ejlali. 2017. Reliability Side-Effects in Internet of Things Application Layer Protocols. https://doi.org/10.1109/ICSRS.2017.8272822

[40] Michael Schwarz, Florian Lackner, and Daniel Gruss. 2019. JavaScript Template Attacks: Automatically Inferring Host Information for Targeted Exploits. In *NDSS* (2019).

[41] Alexander Sjösten, Steven Van Acker, and Andrei Sabelfeld. 2017. Discovering Browser Extensions via Web Accessible Resources. In *Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy (CODASPY '17)*. ACM, New York, NY, USA. https://doi.org/10.1145/3029806.3029820

[42] Jan Spooren, Davy Preuveneers, and Wouter Joosen. 2015. Mobile Device Fingerprinting Considered Harmful for Risk-based Authentication. In *Proceedings of the Eighth European Workshop on System Security*.

[43] Jan Spooren, Davy Preuveneers, and Wouter Joosen. 2017. Leveraging Battery Usage from Mobile Devices for Active Authentication. (25 Oct. 2017). https://doi.org/10.1155/2017/1367064

[44] Oleksii Starov and Nick Nikiforakis. 2017. XHOUND: Quantifying the Fingerprintability of Browser Extensions. https://doi.org/10.1109/SP.2017.18

[45] K. Takasu, T. Saito, T. Yamada, and T. Ishikawa. 2015. A Survey of Hardware Features in Modern Browsers. In *2015 9th International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing* (2015-07). 520–524. https://doi.org/10.1109/IMIS.2015.72

[46] Kurt Thomas, Frank Li, Ali Zand, Jacob Barrett, Juri Ranieri, Luca Invernizzi, Yarik Markov, Oxana Comanescu, Vijay Eranti, Angelika Moscicki, et al. 2017. Data breaches, phishing, or malware?: Understanding the risks of stolen credentials. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 1421–1434.

[47] T. Unger, M. Mulazzani, D. Frühwirt, M. Huber, S. Schrittwieser, and E. Weippl. 2013. SHPF: Enhancing HTTP(S) Session Security with Browser Fingerprinting. In *2013 International Conference on Availability, Reliability and Security*. 255–261. https://doi.org/10.1109/ARES.2013.33

[48] Antoine Vastel, Pierre Laperdrix, Walter Rudametkin, and Romain Rouvoy. 2018. FP-STALKER: Tracking Browser Fingerprint Evolutions. IEEE, 14. https://hal.inria.fr/hal-01652021/document

[49] Antoine Vastel, Walter Rudametkin, Romain Rouvoy, and Xavier Blanc. 2020. FP-Crawlers: Studying the Resilience of Browser Fingerprinting to Block Crawlers. In *NDSS Workshop on Measurements, Attacks, and Defenses for the Web (MADWeb'20)* (2020).

[50] Stephan Wiefling, Luigi Lo Iacono, and Markus Dürmuth. 2019. Is This Really You? An Empirical Study on Risk-Based Authentication Applied in the Wild. In *ICT Systems Security and Privacy Protection* (Cham, 2019) *(IFIP Advances in Information and Communication Technology)*. Springer International Publishing, 134–148. https://doi.org/10.1007/978-3-030-22312-0_10

[51] Wenjia Wu, Jianan Wu, Yanhao Wang, Zhen Ling, and Ming Yang. 2016. Efficient Fingerprinting-Based Android Device Identification with Zero-Permission Identifiers. *IEEE Access* PP (Nov. 2016), 1–1. https://doi.org/10.1109/ACCESS.2016.2626395

[52] Vasilios Zorkadis and P Donos. 2004. On biometrics-based authentication and identification from a privacy-protection perspective. (2004).

A BROWSER FINGERPRINTING ATTRIBUTES

In this section, we describe the 216 *source attributes* that are included in our script, and the 46 *extracted attributes* that are derived from source attributes. We group these attributes into families, and provide references to related studies. Their name is sufficient to retrieve the corresponding browser property, and when needed, we provide a brief description of the method for reproducibility. We focus here on the description of the method, and provide a *complete list* of the attributes and their property in Appendix E.

A.1 JavaScript properties

Most attributes are *properties* that are accessed through common *JavaScript objects*. The `navigator` object provides information on the browser (e.g., version), its customization (e.g., language), the underlying system (e.g. operating system), and supported functionalities (e.g., list of available codecs). The `screen` object provides information on the screen size, the orientation, the pixel density, and the available space for the web page. The `window` object provides information on the window containing the web page, like its size or the support of storage mechanisms. The `document` object gives access to the web page content, but also to a few properties. Such properties are already included in previous studies [13, 19, 29] or open source projects[21], but are usually limited to less than 15 properties.

A.2 HTTP headers

We include 16 attributes that are collected from *HTTP headers*, most of which are already used in previous studies [13, 19, 29]. Among these 16 attributes, 15 consist of the value of an explicitly specified header, and the last attribute stores any remaining fields as pairs of name and value.

[21] https://github.com/Valve/fingerprintjs2

A.3 Enumeration or presence of browser components

One attribute family that provides a high diversity is the *browser components* that are installed in the browser. The presence of some components can directly be accessed (e.g., the installed plugins [22]), whereas the presence of others have to be inferred (e.g., the installed fonts). The list of components that are given in this section have the components separated by a comma.

A.3.1 List attributes. Previous studies already identified the *list of plugins* and the *list of fonts* as highly distinctive [13, 29], hence we include these two attributes in our fingerprinting script. We enumerate the list of plugins, and check the size of text boxes [15] to infer the presence of 66 fonts. Additionally, we get the *list of mime types* (i.e., the supported data format), and the *list of speech synthesis voices*.

A.3.2 Support of video, audio, and streaming codecs. We infer the *support of video codecs* by creating a `video` element and checking if it can play a given type using the `canPlayType()` function. The *support of audio codecs* is done using the same method, but for an `audio` element. We infer the *support of streaming codecs* by calling the `isTypeSupported()` function of the `window.[WebKit, moz, ms, ∅]MediaSource` object, using both the audio codecs and the video codecs for which we check the presence. We apply the same method to infer the *support of recording codecs*, but on the `MediaRecorder` object instead.

A.3.3 List of video codecs. The 15 *video codecs* for which we infer the presence are the following: video/mp2t; codecs="avc1.42E01E,mp4a.40.2", video/mp4; codecs="avc1.42c00d", video/mp4; codecs="avc1.4D401E", video/mp4; codecs="mp4v.20.8", video/mp4; codecs="avc1.42E01E", video/mp4; codecs="avc1.42E01E, mp4a.40.2", video/mp4; codecs="hvc1.1.L0.0", video/mp4; codecs="hev1.1.L0.0", video/ogg; codecs="theora", video/ogg; codecs="vorbis", video/webm; codecs="vp8", video/webm; codecs="vp9", application/dash+xml, application/vnd.apple.mpegURL, audio/mpegurl.

A.3.4 List of audio codecs. The 9 *audio codecs* for which we infer the presence are the following: audio/wav; codecs="1", audio/mpeg, audio/mp4; codecs="mp4a.40.2", audio/mp4; codecs="ac-3", audio/mp4; codecs="ec-3", audio/ogg; codecs="vorbis", audio/ogg; codecs="opus", audio/webm; codecs="vorbis", audio/webm; codecs="opus".

A.3.5 List of detected fonts. The 66 *fonts* for which we infer the presence are the following: Andale Mono; AppleGothic; Arial; Arial Black; Arial Hebrew; Arial MT; Arial Narrow; Arial Rounded MT Bold; Arial Unicode MS; Bitstream Vera Sans Mono; Book Antiqua; Bookman Old Style; Calibri; Cambria; Cambria Math; Century; Century Gothic; Century Schoolbook; Comic Sans; Comic Sans MS; Consolas; Courier; Courier New; Garamond; Geneva; Georgia; Helvetica; Helvetica Neue; Impact; Lucida Bright; Lucida Calligraphy; Lucida Console; Lucida Fax; LUCIDA GRANDE; Lucida Handwriting; Lucida Sans; Lucida Sans Typewriter; Lucida Sans Unicode; Microsoft Sans Serif; Monaco; Monotype Corsiva; MS Gothic; MS Outlook; MS PGothic; MS Reference Sans Serif; MS Sans Serif; MS Serif; MYRIAD; MYRIAD PRO; Palatino; Palatino Linotype; Segoe Print; Segoe Script; Segoe UI; Segoe UI Light; Segoe UI Semibold; Segoe UI Symbol; Tahoma; Times; Times New Roman; Times New Roman PS; Trebuchet MS; Verdana; Wingdings; Wingdings 2; Wingdings 3.

A.4 Extension detection

The installed *browser extensions* cannot be directly accessed, but their presence can be inferred by changes brought to the page content by the extension [44], or by the presence of web accessible resources [23, 41]. We check the changes that can be brought to the page content by 8 extensions

[22] At the exception of the Firefox browsers that now only display the Flash plugin.

Table 4. Extensions detected by the changes they bring to the page content.

Extension	Page content change
Privowny	W.privownyAddedListener[EXT] is supported
UBlock	D.head has `display: none !important;` and `:root` as style
Pinterest	D.body.data-pinterest-extension-installed is supported
Grammarly	D.body.data-gr-c-s-loaded is supported
Adguard	W.AG_onLoad is supported
Evernote	Element with `style-1-cropbar-clipper` as `id` exists
TOTL	W.ytCinema is supported
IE Tab	W.ietab.getVersion() is supported

Table 5. Extensions detected by the availability of their web accessible resource. C stands for chrome, and R stands for resource.

Extension	Web accessible resource
Firebug	C://firebug/skin/firebugBig.png
YahooToolbar	R://635abd67-4fe9-1b23-4f01-e679fa7484c1/icon.png
EasyScreenshot	C://easyscreenshot/skin/icon16.png
Ghostery	R://firefox-at-ghostery-dot-com/data/images/ghosty-16px.png
Kaspersky	R://urla-at-kaspersky-dot-com/data/icon-16.png
VideoDownloadHelper	R://b9db16a4-6edc-47ec-a1f4-b86292ed211d/data/images/icon-18.png
GTranslate	R://aff87fa2-a58e-4edd-b852-0a20203c1e17/icon.png
Privowny	C://privowny/content/icons/privowny_extension_logo.png

that are listed in Table 4, and the availability of the web accessible resources of 8 extensions that are listed in Table 5.

A.4.1 Detection of an ad blocker. We also infer the presence of an *advertisement blocker* by creating an invisible dummy advertisement, and by checking if it is removed or not. The dummy advertisement is a created division with the `id` property set to `ad_ads_pub_track`, the `class` property set to `ads .html?ad= /?view=ad text-ad textAd text_ad text_ads text-ads`, and the `style` property set to `width: 1px !important; height: 1px !important; position: absolute !important; left: -1000px !important; top: -1000px !important;`.

A.5 Size and color of web page elements

A.5.1 Bounding boxes. The attributes related to the *bounding boxes* concern a `div` element to which we append a `span` element. The `div` element has his `style` property set to the values displayed in Table 5. The `span` element contains a specifically crafted text that is provided below. The size of the bounding boxes (i.e., the width and the height of the rectangles of the `div` and the `span` elements) are then collected using the `getClientRects` function. The text of the `span` element is `\ua9c0 \u2603 \u20B9 \u2604 \u269b \u2624 \u23B7 \u262c \u2651 \u269d \u0601 \u0603 \uaac1 \u060e \u06dd \ud83c\udfe1 mmmmmmmmmmlil \u102a`. The characters that start with a `\u` are special Unicode characters (e.g., emoji, letter of a non-latin alphabet).

A.5.2 Width and position of a created div. The attribute named *width and position of a created* `div` is the properties of `width` and `transform-origin` of a newly created `div` element, obtained by

Table 6. Properties of the `div` element measured for the attributes related to the bounding boxes.

Property	Value
position	absolute
left	-9999px
textAlign	center
objectFit	scale-down
font	68px / 83px Helvetica, Arial, Sans-serif
zoom	66%
MozTransform	scale(0.66)
visibility	hidden

calling the `getComputedStyle` function. This created `div` element is afterward used to infer the color of layout components, as described below.

A.5.3 Colors of layout components. The attribute *colors of layout components* is obtained by applying the color of several layout components (e.g., the scroll bar) to the created `div` element, and by getting the color back from the property `W.getComputedStyle(new_div).color`. Each of the tested component gets its color extracted this way, and aggregated in this attribute. The color of each element is afterward extracted as a single attribute. They are displayed at the end of the Table 13.

A.6 WebGL properties

Our script collects several properties from the WebGL API. To obtain them, we create a `canvas` element and get its WebGL context by calling `getContext()` with any of the following parameters: `webgl`, `webgl2`, `moz-webgl`, `experimental-webgl`, or `experimental-webgl2`.

The property `MAX_TEXTURE_MAX_ANISOTROPY_EXT` is obtained from one of `[WEBKIT_EXT_, MOZ-_EXT_, EXT_]texture_filter_anisotropic`. To get the *UNMASKED* attributes, we first get an identifier named `id` from the unmasked property of the `getExtension('WEBGL_debug_renderer_info')` object, and then get the actual value by calling `getParameter(id)`. Finally, to get the `COMPRESSED_TEXTURE_FORMATS` property, we have to load the `[WEBKIT_]WEBGL_compressed-_texture_s3tc` extension first.

A.7 WebRTC fingerprinting

We include a WebRTC fingerprinting method similar to the method proposed by Takasu et al. [45]. The method consists into getting information about the Session Description Protocol of a generated WebRTC connection. Due to the variability of this information, we create two different connections and hold only the values that are identical between these two. As this method leaks internal IP addresses, we hash them directly on the client.

A.8 HTML5 canvases inspired by previous studies

We dedicate the Section 5.3 to a focus on dynamic attributes, and provide here examples of the two HTML5 canvases that are inspired by previous studies. Our script includes a canvas inspired by the AmIUnique study [29] in both PNG and JPEG formats (example given at Figure 17), and an enhanced version of it similar to the Morellian study [26] in PNG format (example given at Figure 18).

Fig. 17. HTML5 canvas inspired by the AmIUnique [29] study in PNG format.

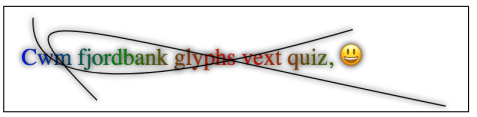

Fig. 18. HTML5 canvas similar to the Morellian [26] study in PNG format.

A.9 Audio fingerprinting

We dedicate the Section 5.3 to a focus on dynamic attributes, and provide here the concrete implementation together with the network of `AudioNode` objects used within each audio fingerprinting methods. These methods are inspired by the methods described by Englehardt et al. [14], and are designed to form complex networks of `AudioNode` objects to have more chances to induce fingerprintable behaviors.

A.9.1 Simple audio fingerprinting method. The *simple process* consists of three `OscillatorNode` objects that generate a periodic wave, connected to a single `DynamicsCompressorNode`, and finishing to a `AudioDestinationNode`. The architecture of the network of `AudioNode` objects for the simple process is depicted in Figure 19, together with the parameters set for each node. The `OscillatorNode` objects are started one after the other, and overlap at some time. The sequence of events is the following: (1) the triangle oscillator node is started at $t = 0$ seconds, (2) the square oscillator is started at $t = 0.10$ seconds, (3) the triangle oscillator is stopped $t = 0.20$ seconds, together with the start of the sine oscillator node, (4) the square oscillator is stopped at $t = 0.25$ seconds. When the rendering of the audio context is done, the `complete` event is triggered, and gives access to a `renderedBuffer` that contains the audio data encoded as a buffer of 32 bits floating-point numbers. The *audio fp simple* (AFS) attribute is an integer computed as the sum of the values of the `renderedBuffer`, that are first casted to absolute integers.

A.9.2 Advanced audio fingerprinting method. The *advanced process* consists of four `OscillatorNode` objects, two `BiquadFilterNode` objects, two `PannerNode` objects, one `DynamicsCompressorNode`, one `AnalyserNode`, and one `AudioDestinationNode`. The architecture of the networks of `AudioNode` objects for the simple process is depicted in Figure 20, together with the parameters set for each node. The `OscillatorNode` objects are started one after the other, and overlap at some time. The sequence of events is the following: (1) the triangle oscillator node and the sine oscillator node with a frequency of 280 are started at $t = 0$ seconds, (2) the square oscillator is started at $t = 0.05$ seconds, (3) the triangle oscillator is stopped $t = 0.10$ seconds, (4) the sine oscillator with a frequency of 170 is stopped at $t = 0.15$ seconds, (5) the square oscillator is stopped at $t = 0.20$ seconds. When the rendering of the audio context is done, the `complete` event is triggered, and gives access to a `renderedBuffer` that contains the audio data encoded as a buffer of 32 bits floating-point numbers. The *audio fp advanced* (AFA) attribute is an integer computed as the sum of the values of the `renderedBuffer`, that are first casted to absolute integers. The *audio fp advanced frequency data* (AFA-FD) attribute is the sum of the frequency data obtained through the `getFloatFrequencyData` function of the `Analyser Node`.

B KEYWORDS

In this section, we provide the keywords used to infer the browser family, the operating system, and if the browser is a robot (i.e., if it is an automatized tool, hence is not a genuine visitor). We match these keywords on the UserAgent JavaScript property, that is first set to lower case. We manually compiled the keywords, by searching for meaningful keywords. Afterward, we classify

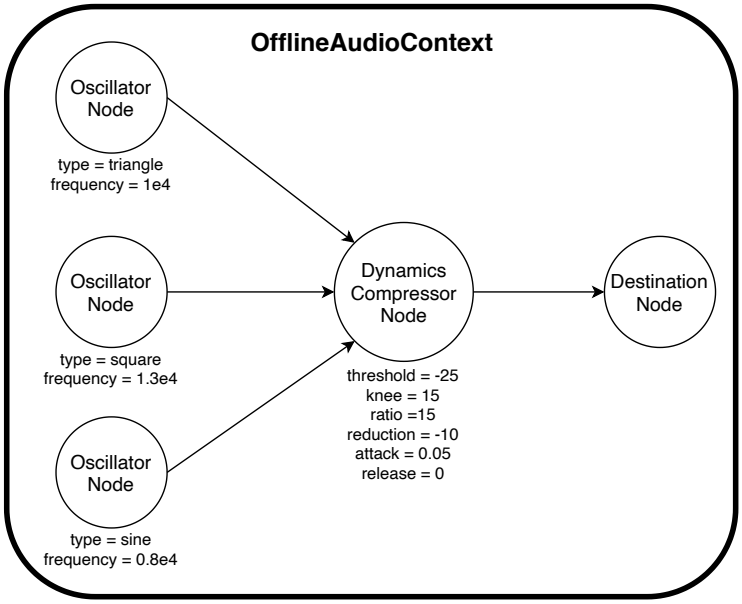

Fig. 19. Architecture of the network of `AudioNode` objects for the simple audio fingerprinting method.

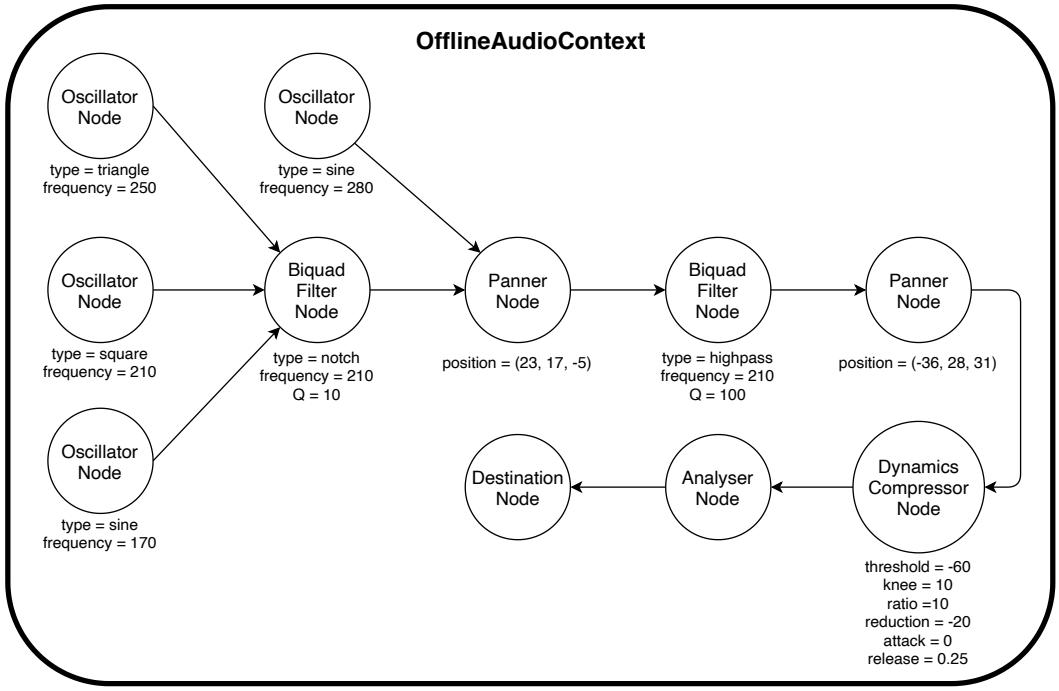

Fig. 20. Architecture of the network of `AudioNode` objects for the simple audio fingerprinting method.

Table 7. The keywords and the exact UserAgent values that we consider as indicating a robot browser. Long values are cut at a blank space and displayed with indentations.

Blacklisted keyword	Blacklisted value
googlebot	mozilla/4.0 (compatible; msie 7.0; windows nt 6.1; trident/7.0; slcc2; .net clr 2.0.50727; .net clr 3.5.30729; .net clr 3.0.30729; media center pc 6.0; .net4.0c; .net4.0e)
evaliant	
bot.html	
voilabot	mozilla/5.0 (x11; linux x86_64) applewebkit/537.36 (khtml, like gecko) chrome/52.0.2743.116 safari/537.36
google web preview	
spider	mozilla/5.0 (windows nt 6.3; rv:36.0) gecko/20100101 firefox/36.0
bingpreview	mozilla/5.0 (macintosh; intel mac os x 10.10; rv:38.0) gecko/20100101 firefox/38.0

the UserAgents given these keywords, and searched for additional keywords in the unclassified UserAgents that remain. This process was repeated until every UserAgent was classified. This process leads to a huge variety of finely-grained keywords, hence we only provide here the keywords that are meaningful for this study.

B.1 Robot keywords

We check that the `UserAgent` HTTP header does not contain the keywords, nor is set to the exact values, that are listed in Table 7.

B.2 Device type

To infer the device type of a browser, we match keywords sequentially with the UserAgent of the browser. The set of keywords can overlap between two device types (e.g., the UserAgent of tablet browsers often contain keywords of mobile browsers, like `mobile` for example). Due to this overlapping problem, we verify that the UserAgent of the browser contains the keyword of the device type, and does not contain the keyword of some other device types. Table 8 lists the keywords that we leverage to infer each device type.

The *mobile devices* are smartphones, and do not include tablets. We assert that their UserAgent contain a `mobile` keyword, and no `tablet` nor `miscellaneous` keyword. To infer that a device is a *tablet*, we assert that their UserAgent contain a `tablet` keyword, and no `miscellaneous` keyword. The *miscellaneous devices* are game consoles and smart TVs. We assert that their UserAgent contain a `miscellaneous` keyword. Finally, to infer that a device is a *desktop* computer, we assert that their UserAgent does not contain any of the `mobile`, `tablet`, or `miscellaneous` keyword. In the table, we omit a `miscellaneous` keyword due to its size, which is: opera/9.80 (linux i686; u; fr) presto/2.10.287 version/12.00 ; sc/ihd92 stb.

B.3 Browser and operating system families

Table 9 lists the keywords that we leverage to infer the family of a browser, and Table 10 lists the keywords that we leverage to infer the operating system family of a browser. As a keyword can be in the UserAgent of two different families, we verify the keywords sequentially in the order presented in the tables, and classify a device in the first family having a keyword that matches.

C BROWSER FINGERPRINTING-BASED AUTHENTICATION MECHANISM

Browser fingerprinting can enhance an authentication mechanism by providing an additional barrier at a low usability and deployability cost. In this section, we provide an example of how it

Table 8. The keywords that we consider as indicating each device type.

Mobile	Tablet	Miscellaneous
phone	ipad	wii
mobile	tablet	playstation
android	terra pad	smart-tv
iphone	tab	smarttv
blackberry		googletv
wpdesktop		opera tv
		appletv
		nintendo
		xbox

Table 9. The keywords that we consider as indicating each browser family.

Browser Family	Keywords
Firefox	Firefox
Internet Explorer	MSIE, Trident/7.0
Chrome	Chrome, Chromium

Table 10. The keywords that we consider as indicating each operating system family.

Operating System Family	Keywords
Windows 10	Windows NT 10.0
Windows 7	Windows NT 6.1
Other Windows	Windows NT, Windows 7, Windows 98, Windows 95, Windows CE
Mac OS	Mac OS X (but not iPad, nor iPhone)
Linux-based	Linux, CrOS, NetBSD, FreeBSD, OpenBSD, Fedora, Ubuntu, Mint
Android	Android
Windows Phone	Windows Phone
iOS	iPad, iPhone (but not Mac OS X)

can concretely be implemented. The verifier controls a web platform on which users have a registered account. Each registered account is associated to an identifier, and to a set of authentication factors. The authentication factors include the fingerprints of each registered browser for a user. We emphasize that dynamic attributes can be integrated to the browser fingerprints to enforce a challenge-response mechanism that mitigates replay attacks [26, 38].

Enrollment. The enrollment consists for a user to create his account and link the several authentication factors that he uses. During this step, the user and the verifier agrees on the account identifier (e.g., username, email address, phone number) and on the authentication factors (e.g., password, email address, phone number) that are assigned to the user. The fingerprint of the browser in use by the user during the enrollment is collected and stored as the first browser fingerprint of the user. To register an additional browser, the user is required to authenticate using other strong factors (e.g., a physical token, one time passwords), before getting the fingerprint of this new browser stored.

Authentication. During each authentication, the user claims an account by providing the identifier and by presenting the used authentication factors. The verifier compares the presented authentication factors with the ones stored for this user, and if they match the user is deemed legitimate

and is given access to the account. Otherwise, the user is denied access, and the verifier can take preventive actions [18] according to her policy (e.g., blocking the account). The verifier notably compares the collected browser fingerprint with the fingerprints of the browsers registered to the account. If the other factors match, and the fingerprint of one of the registered browser matches the collected fingerprint, the fingerprint stored for this browser is updated to the newly collected one.

Account Recovery. It happens that legitimate users are not able to provide the authentication factors (e.g., a password is forgotten, a physical token is lost). When it occurs, users are given access to an account recovery mechanism [32] that leverages other authentication factors than the usual ones (e.g., face-to-face verification, email verification). Users cannot mistake their browser fingerprint, but it can become hard to recognize if too many changes are brought to the web environment into which the browser is running. In this case, the user is asked to undergo the account recovery step, and to select the registered browser for which to update its fingerprint.

D ADVANCED VERIFICATION MECHANISM

Section 3.3.4 describes a simple verification mechanism that simply checks that the number of identical attributes between the presented fingerprint and the one stored is below a threshold. In this section, we present the results obtained using an *advanced verification mechanism* that incorporates matching functions that authorize limited changes between the attribute values of the two fingerprints. The methodology to obtain the datasets is the same as described in Section 4.3.3.

D.1 Attributes matching

The *advanced verification mechanism* leverages matching functions for the comparison between attributes. It counts the attributes that match between the two compared fingerprints, given the matching functions, and considers the evolution legitimate if this number is above a threshold. More formally, we seek to compare the stored fingerprint f to the presented fingerprint g. To do so, we compare the values $f[a]$ and $g[a]$ of the attribute a for the fingerprints f and g, using the matching function $\approx^a$. The *matching function* $\approx^a$ of the attribute a verifies that the distance between $f[a]$ and $g[a]$ are below a threshold θ^a. Finally, we deem that g is a legitimate evolution of f, if the total number of matching attributes between f and g is above a threshold Θ.

Similarly to previous studies [13, 22, 48], we consider a distance measure that depends on the type of the attribute. The Damerau-Levenshtein distance [21] is used for the textual attributes, the Jaccard distance [51] is used for the set attributes, the absolute difference is used for the numerical attributes, and the identity function is used for the categorical attributes. The *distance thresholds* of each attribute is obtained by training a Support Vector Machines [20] model on the two classes of each month sample, and extract the threshold from the resulting hyperplane. At the exception of the dynamic attributes, that are required to be identical (i.e., the distance threshold is null) as they would contribute to a challenge-response mechanism [26, 38].

D.2 Distribution of matching attributes

Figure 21 displays the distribution of the matching attributes between the same-browser comparisons and the different-browsers comparisons, starting from 51 matching attributes as there are no observed value below. Figure 22 presents a focus that starts from 214 matching attributes, below which there are less than 0.001 of the same-browser comparisons. We can observe that the two classes are well separated, as 99% of the same-browser comparisons have at least 235 identical attributes, and 99% of the different-browsers comparisons are below. The different-browsers

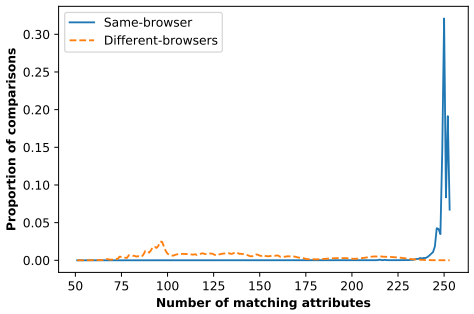

Fig. 21. The number of matching attributes between the same-browser comparisons and the different-browsers comparisons.

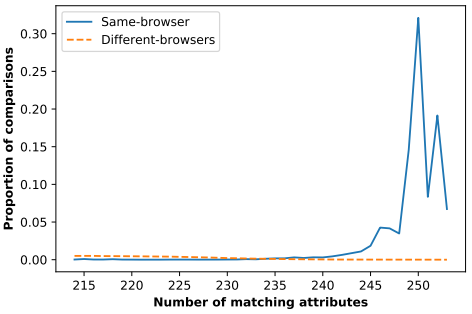

Fig. 22. The number of matching attributes between the same-browser comparisons and the different-browsers comparisons, starting from 214 matching attributes.

comparisons have generally fewer and more spread number of matching attributes than the same-browser comparisons. The different-browsers comparisons have between 51 and 253 matching attributes, with an average of 134.59 attributes, and a standard deviation of 43.25 attributes. The same-browser comparisons have between 81 and 253 matching attributes, with an average of 249.45 attributes, and a standard deviation of 3.69 attributes.

D.3 Distribution of match rates

Figure 23 displays the false match rate (FMR), which is the proportion of the same-browser comparisons that are classified as different-browsers comparisons, and the false non-match rate (FNMR), which is the inverse. The displayed results are the average for each number of matching attributes among the six samples. As there are no same-browser comparisons that have less than 235 matching attributes, the FNMR is null until this value. However, after exceeding this threshold, the FNMR increases as same-browser comparisons begin to be classified as different-browsers comparisons. The equal error rate, which is the rate where both the FMR and the FNMR are equal, is of 0.66% and is achieved for 234 matching attributes.

D.4 Comparison with identical matching

The matching functions of the advanced verification mechanism leads to more attributes that match than attributes that are identical between two fingerprints. The higher number of matching attributes happens for the same-browser comparisons, but also for the different-browsers comparisons. This reduces the False Non-Match Rate (FNMR), but also increases the False-Match Rate (FMR). Due to the FMR being higher, the equal error rate is slightly higher for the advanced verification mechanism that leverages matching functions.

Table 11 compares the results of the simple verification mechanism that leverages the identical attributes, and the advanced verification mechanism that leverages the matching attributes. Considering the matching functions only increase the average number of matching attributes for the same-browser comparisons by 0.81, whereas this increase is greater for the different-browsers comparisons at 7.18. The matching functions seem to contribute more to falsely linking different-browsers comparisons than same-browser comparisons. Due to this, the equal error rate is slightly higher for the advanced verification mechanism than for the simple one, respectively at 0.67% against 0.61%. Finally, we remark that the range of the identical attributes for the same-browser comparisons only goes up to 252 attributes. This is explained by our deduplication step during the

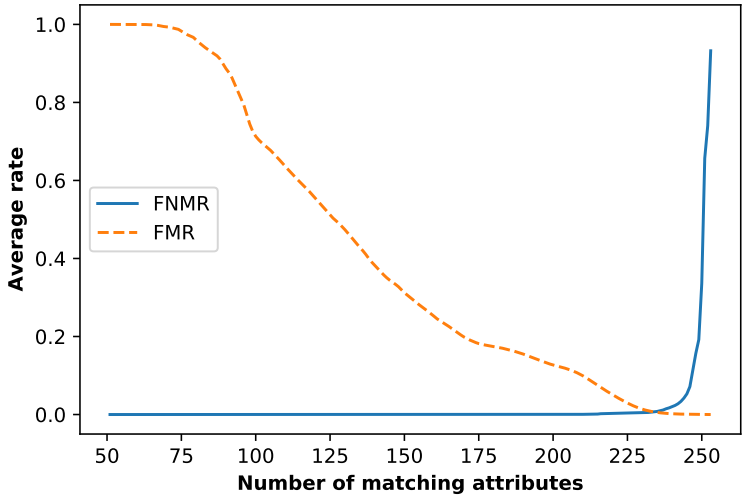

Fig. 23. False match rate (FMR) and false non-match rate (FNMR) given the required number of matching attributes, averaged among the six samples.

Table 11. Comparison between the simple verification mechanism using identical attributes and the advanced verification mechanism using matching attributes.

Result	Simple	Advanced
Same-browser: range of identical or matching attributes	[72; 252]	[81; 253]
Same-browser: average identical or matching attributes	248.64	249.45
Same-browser: standard deviation	3.91	3.69
Different-browsers: range of identical or matching attributes	[34; 253]	[51; 253]
Different-browsers: average identical or matching attributes	127.41	134.59
Different-browsers: standard deviation	44.06	43.25
Equal error rate (EER)	0.62%	0.67%
Threshold of identical or matching attributes	231	233

preprocessing of the dataset (see Section 2.4.3) that removes the consecutive fingerprints that are identical, hence they are forcibly different. As for the different-browsers comparisons, it always goes up to 253 attributes, as coincidence can make fingerprints of different browsers match.

E ATTRIBUTES LIST AND PROPERTY

In this section, we provide the complete list of our fingerprinting attributes, together with their property.

E.1 Properties distribution

In this section, we discuss the distribution of the properties of our 262 attributes that are displayed in Table 13. We focus here on the distribution of the number of distinct values, and refer the reader to Section 5 for the distribution of the other properties of the attributes, namely: the normalized entropy, the minimum normalized conditional entropy, the sameness rate, the median collection

Table 12. The minimum, the average, the maximum, and the standard deviation (Std. dev.) of the distinct values, of the normalized entropy, of the minimum normalized conditional entropy, of the median collection time in seconds, of the median size in bytes, and of the sameness rate of the attributes.

Property	Minimum	Average	Maximum	Std. dev.
Distinct values	1	7,633	671,254	51,298
Normalized entropy	0.000	0.090	0.420	0.094
Minimum normalized conditional entropy	0.000	0.008	0.181	0.021
Sameness rate	0.470	0.982	1.000	0.069
Median collection time (seconds)	0.008	0.811	2.184	0.932
Median size (bytes)	1	23.51	502	60.84

time, and the median size. Table 12 provides the minimum, the average, the maximum, and the standard deviation of these properties among the attributes.

E.1.1 Distribution of distinct values. Figure 24 depicts the cumulative distribution of the distinct values among the attribute. The distinct values presents high variations, as can be seen by the standard deviation of $51,298$. We have 42% of the attributes that have 10 distinct values or less, 63% that have 100 distinct values or less, and 79% that have $1,000$ distinct values or less. Only 5 attributes have more than $100,000$ distinct values, which are the WebRTC fingerprinting method ($671,254$ values), the list of plugins ($314,518$ values), the custom canvas in the PNG format ($269,874$ values) and in the JPEG format ($205,005$ values), together with the list of mime types ($174,876$ values).

The values of the attributes are of different nature, impacting the distinct values that can be observed among different population or time ranges. Some attributes have a *fixed number of values*, like the Boolean attributes or the categorical attributes that have a fixed set of possibilities. Other attributes are composed of *elements having a fixed set* of possibilities, but their combination provides a high number of values. It is the case for the attributes that are related to languages that typically are a combination of language identifiers (e.g., fr), that can be valued (e.g., $q = 0.80$). These attributes also comprise the list of fonts, that is composed of Boolean values that indicates the presence of a given font. Other attributes consists into an *integer* or a *real number* represented as a floating-point number, resulting in a large set of possible values. This category comprises any size related attribute (e.g., the screen width and height), or our audio fingerprinting method which results are floating-point numbers with a high precision of more than 8 digits. Finally, some attributes are *textual information* that can include version number, which also results in a high number of distinct values. Moreover, as new values appear through time (e.g., new versions, new software component), the set of the observed values is expected to grow over the observations. Examples are the `UserAgent` or the list of plugins, as these attributes are typically composed of the name and the version of several hardware and software components.

E.2 Attributes list

Table 13 lists our attributes, together with their number of distinct values seen during the experiment (*Values*), their normalized entropy (*N. Ent.*), their sameness rate (*% Same*), their median size (*Size*), and their median collection time (*Time*).

E.2.1 Attributes nomenclature. To stay concise, we replace the name of common JavaScript objects or of API calls by abbreviations. We denote D the JavaScript `document` object, M the `Math` object, N the `navigator` object, S the `screen` object, and W the `window` object. Additionally, we denote A an initialized *Audio Context*, AA an initialized *Audio Analyser*, and AD the `A.destination`

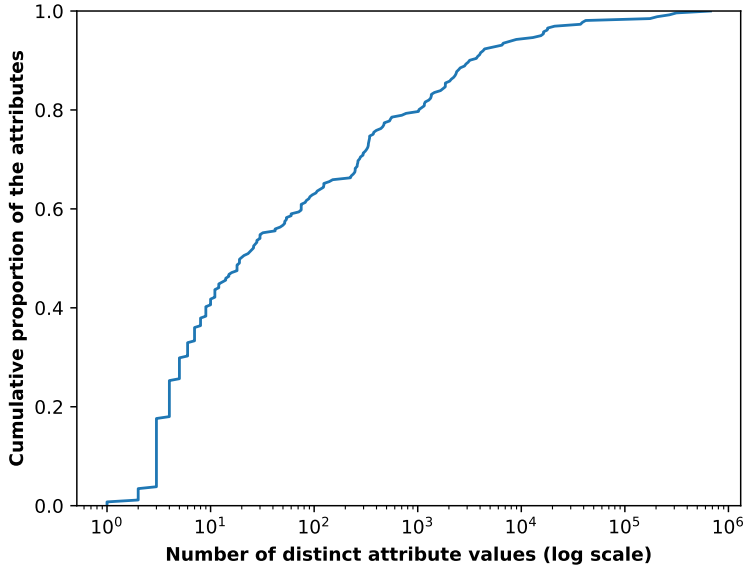

Fig. 24. Cumulative distribution of the number of distinct values of the attributes, in logarithmic scale.

property. Finally, we denote *WG* an initialized *WebGL Context*, *WM* the `WG.MAX_` prefix, and *WI* the `WG.IMPLEMENTATION_` prefix.

Due to the diversity of JavaScript engines, some properties are accessible through different names, regularly prefixed by `moz` for Firefox or `ms` for Internet Explorer. We use square brackets to easily denote these cases, and consider that `A.[B, C]` means that the property is accessed through `A.B` or `A.C`. If there is only one element inside these brackets, this one is optional. We denote `[...]` a part that is omitted but described in the corresponding attribute family description.

Table 13. Browser fingerprinting attributes, together with their distinct values, their normalized entropy, their minimum normalized conditional entropy (MNCE), their stability, their median size, and their median collection time. The attributes from which we derive the extracted ones are ignored for the MNCE.

Attribute	Values	N. Ent.	MNCE	% Same	Size	Time
N.userAgent	38,863	0.394	0.046	0.978	115	0.000
Listing of N	1,660	0.207	0.009	0.989	502	0.001
Listing of screen	82	0.129	0.000	0.999	209	0.000
N.language	228	0.066	0.006	0.999	2	0.000
N.languages	1,448	0.094	0.010	0.998	17	0.000
N.userLanguage	124	0.036	0.001	1.000	1	0.000
N.systemLanguage	115	0.037	0.001	1.000	1	0.000
N.browserLanguage	52	0.036	0.000	1.000	1	0.000
N.platform	32	0.068	0.000	1.000	5	0.000
N.appName	5	0.003	0.000	1.000	8	0.000
N.appVersion	37,310	0.342	0.000	0.984	107	0.000
N.appMinorVersion	10	0.035	0.000	1.000	1	0.000
N.product	2	0.003	0.000	1.000	5	0.000
N.productSub	10	0.067	0.000	1.000	8	0.000
N.vendor	21	0.064	0.000	1.000	1	0.000
N.vendorSub	2	0.035	0.000	1.000	1	0.000
N.cookieEnabled	1	0.000	0.000	1.000	4	0.000
N.cpuClass	6	0.039	0.000	1.000	1	0.000
N.oscpu	60	0.071	0.000	1.000	1	0.000
N.hardwareConcurrency	28	0.086	0.025	0.999	1	0.000
N.buildID	1,351	0.076	0.010	0.989	1	0.000
[N.security, D.security[Policy]]	30	0.038	0.001	1.000	7	0.000
N.permissions	3	0.045	0.000	1.000	1	0.000
W.Notification.permission	5	0.043	0.001	0.999	7	0.000
W.Notification.maxActions	3	0.041	0.000	1.000	1	0.000
N.[msM, m]axTouchPoints	42	0.098	0.004	0.999	3	0.000
D.createEvent("TouchEvent") support	3	0.032	0.000	1.000	1	0.000
W.ontouchstart support	3	0.032	0.000	1.000	1	0.000
N.javaEnabled()	4	0.045	0.008	0.997	1	0.000
N.taintEnabled()	3	0.045	0.000	1.000	1	0.000
[[N, W].doNotTrack, N.msDoNotTrack]	11	0.085	0.012	1.000	6	0.000
N.connection support	3	0.022	0.000	1.000	1	0.000
N.connection.type	12	0.028	0.003	0.992	1	0.000
N.connection.downlink	91	0.032	0.003	0.995	1	0.000
N.[mozC, c]onnection.bandwidth	6	0.023	0.000	1.000	3	0.000
N.mediaDevices support	3	0.044	0.000	0.999	1	0.000
N.mediaDevices.getSupportedConstraints()	12	0.090	0.000	0.997	144	0.000
W.Intl.Collator().resolvedOptions()	311	0.096	0.000	0.999	115	0.005
W.Intl.DateTimeFormat().resolvedOptions()	1,849	0.154	0.011	0.996	111	0.003
W.Intl.NumberFormat().resolvedOptions()	260	0.070	0.000	0.999	138	0.001
W.Intl.v8BreakIterator().resolvedOptions()	75	0.046	0.000	0.999	1	0.000
N.getGamepads()	18	0.090	0.000	0.998	1	0.001

Attribute	Values	N. Ent.	MNCE	% Same	Size	Time
W.InstallTrigger.enabled()	4	0.037	0.000	1.000	1	0.000
W.InstallTrigger.updateEnabled()	4	0.037	0.000	1.000	1	0.000
N.msManipulationViewsEnabled	5	0.052	0.000	1.000	3	0.000
N.[msP, p]ointerEnabled	9	0.051	0.000	1.000	3	0.000
D.msCapsLockWarningOff	3	0.039	0.000	1.000	1	0.000
D.msCSSOMElementFloatMetrics	4	0.039	0.000	1.000	1	0.000
N.[msW, w]ebdriver	6	0.048	0.001	1.000	3	0.000
W.Debug.debuggerEnabled	5	0.042	0.000	0.989	1	0.000
W.Debug.setNonUserCodeExceptions	4	0.042	0.000	0.989	1	0.000
new Date(2016, 1, 1).getTimezoneOffset()	60	0.008	0.001	0.999	2	0.000
Different Timezone at 01/01 and 06/01	3	0.005	0.002	0.999	1	0.000
S.width	1,280	0.192	0.005	0.987	4	0.000
S.height	1,016	0.188	0.015	0.987	3	0.000
W.screenX	3,071	0.125	0.047	0.925	1	0.000
W.screenY	1,181	0.126	0.049	0.925	1	0.000
S.availWidth	1,746	0.202	0.016	0.985	4	0.000
S.availHeight	1,353	0.268	0.058	0.984	3	0.000
S.availTop	460	0.060	0.003	1.000	1	0.000
S.availLeft	372	0.048	0.005	0.999	1	0.000
S.(pixelDepth, colorDepth)	14	0.031	0.001	1.000	5	0.000
S.deviceXDPI	249	0.073	0.000	0.993	1	0.000
S.deviceYDPI	249	0.073	0.000	0.993	1	0.000
S.systemXDPI	75	0.053	0.000	0.999	1	0.000
S.systemYDPI	75	0.053	0.000	0.999	1	0.000
S.logicalXDPI	6	0.039	0.000	1.000	1	0.000
S.logicalYDPI	6	0.039	0.000	1.000	1	0.000
W.innerWidth	2,572	0.263	0.023	0.957	4	0.000
W.innerHeight	2,297	0.388	0.181	0.906	3	0.000
W.outerWidth	2,481	0.293	0.074	0.909	4	0.000
W.outerHeight	4,046	0.327	0.117	0.872	3	0.000
W.devicePixelRatio	2,035	0.103	0.026	0.992	1	0.000
W.mozInnerScreenX	3,682	0.065	0.011	0.991	1	0.000
W.mozInnerScreenY	3,170	0.102	0.020	0.991	1	0.000
W.offscreenBuffering	4	0.067	0.000	1.000	4	0.000
S.orientation	3	0.044	0.000	1.000	1	0.000
S.[orientation.type, [moz, ms]Orientation]	26	0.107	0.003	0.994	21	0.000
S.orientation.angle	7	0.050	0.002	0.996	1	0.000
W.localStorage support	4	0.001	0.001	1.000	1	0.001
W.sessionStorage support	4	0.000	0.000	1.000	1	0.000
W.indexedDB support	3	0.002	0.000	1.000	1	0.000
W.openDatabase support	3	0.045	0.000	1.000	1	0.000
W.caches support	3	0.045	0.000	1.000	1	0.000
M.tan(-1e300)	15	0.087	0.002	1.000	19	0.000
M.tan(3.14159265359 * 0.3333 * 1e300)	12	0.085	0.000	0.999	18	0.000
M.acos(0.000000000000001)	4	0.058	0.000	1.000	18	0.000
M.acosh(1.000000000001)	7	0.071	0.000	1.000	24	0.000

Attribute	Values	N. Ent.	MNCE	% Same	Size	Time
M.asinh(0.00001)	6	0.071	0.000	1.000	23	0.000
M.asinh(1e300)	6	0.058	0.000	1.000	17	0.000
M.atan(2)	3	0.018	0.000	1.000	18	0.000
M.atan2(0.01, 1000)	3	0.045	0.000	1.000	23	0.000
M.atanh(0.0001)	5	0.070	0.000	1.000	22	0.000
M.cosh(15)	8	0.058	0.000	1.000	18	0.000
M.exp(-1e2)	8	0.028	0.000	1.000	21	0.000
M.exp(1e2)	9	0.028	0.000	1.000	22	0.000
M.LOG2E	3	0.039	0.000	1.000	18	0.000
M.LOG10E	3	0.000	0.000	1.000	18	0.000
M.E	2	0.000	0.000	1.000	17	0.000
M.LN10	3	0.000	0.000	1.000	17	0.000
D.defaultCharset	71	0.075	0.001	0.999	1	0.000
Width and height of fallback font text	2,347	0.199	nan	0.998	11	0.100
W.[performance, console].jsHeapSizeLimit	24	0.083	0.002	0.991	3	0.000
W.menubar.visible	5	0.035	0.000	1.000	4	0.000
W.isSecureContext	4	0.045	0.000	0.999	5	0.000
S.fontSmoothingEnabled	4	0.042	0.001	1.000	1	0.000
new Date(0)	1,846	0.118	0.004	0.998	82	0.004
new Date(”0001-1-1”)	2,107	0.150	0.010	0.999	60	0.002
new Date(0) then setFullYear(0)	2,376	0.136	0.013	0.998	61	0.001
Detection of an adblocker	19	0.002	0.001	0.999	1	2.157
Firebug resource detection	3	0.037	0.000	1.000	1	0.055
YahooToolbar resource detection	3	0.037	0.000	1.000	1	0.056
EasyScreenshot resource detection	3	0.037	0.000	1.000	1	0.056
Ghostery resource detection	3	0.037	0.000	1.000	1	0.057
Kaspersky resource detection	3	0.037	0.000	1.000	1	0.057
VideoDownloadHelper resource detection	3	0.038	0.001	0.998	1	0.057
GTranslate resource detection	3	0.037	0.000	1.000	1	0.059
Privowny resource detection	2	0.037	0.000	1.000	1	0.059
Privowny page content change	3	0.000	0.000	1.000	3	2.170
UBlock page content change	4	0.000	0.000	1.000	1	2.183
Pinterest page content change	10	0.001	0.001	0.999	1	2.153
Grammarly page content change	3	0.000	0.000	1.000	1	2.184
Adguard page content change	3	0.000	0.000	1.000	1	2.165
Evernote page content change	3	0.000	0.000	1.000	1	2.181
TOTL page content change	3	0.000	0.000	1.000	1	2.181
IE Tab page content change	11	0.000	0.000	1.000	1	2.156
WebRTC fingerprinting	671,254	0.294	0.144	0.765	1	0.764
WG.SHADING_LANGUAGE_VERSION	23	0.103	0.000	0.996	18	0.001
WG.VERSION	247	0.123	0.008	0.995	10	0.000
WG.VENDOR	11	0.080	0.000	0.997	7	0.000
WG.RENDERER	14	0.089	0.000	0.996	12	0.000
WG.ALIASED_POINT_SIZE_RANGE	42	0.129	0.006	0.996	5	0.000
WG.ALIASED_LINE_WIDTH_RANGE	30	0.077	0.003	0.996	3	0.000
WM.VIEWPORT_DIMS	13	0.107	0.009	0.995	11	0.000

Attribute	Values	N. Ent.	MNCE	% Same	Size	Time
WG.SUBPIXEL_BITS	9	0.039	0.001	0.997	1	0.000
WG.SAMPLE_BUFFERS	5	0.039	0.000	0.996	1	0.000
WG.SAMPLES	9	0.075	0.001	0.992	1	0.000
WG.COMPRESSED_TEXTURE_FORMATS	3	0.034	0.000	0.998	23	0.000
WM.VERTEX_UNIFORM_ATTRIBUTES	18	0.118	0.006	0.996	3	0.000
WM.COMBINED_TEXTURE_IMAGE_UNITS	19	0.079	0.003	0.996	2	0.000
WM.FRAGMENT_UNIFORM_ATTRIBUTES	18	0.109	0.004	0.996	3	0.000
WM.CUBE_MAP_TEXTURE_SIZE	11	0.084	0.008	0.995	5	0.000
WG.STENCIL_VALUE_MASK	8	0.050	0.000	0.996	10	0.000
WG.STENCIL_WRITEMASK	7	0.050	0.000	0.996	10	0.000
WG.STENCIL_BACK_VALUE_MASK	8	0.050	0.000	0.996	10	0.000
WG.STENCIL_BACK_WRITEMASK	7	0.050	0.000	0.996	10	0.000
WM.TEXTURE_SIZE	10	0.081	0.009	0.995	5	0.000
WG.DEPTH_BITS	7	0.047	0.000	0.996	2	0.000
WM.VARYING_ATTRIBUTES	19	0.121	0.009	0.996	2	0.000
WI.COLOR_READ_FORMAT	7	0.073	0.003	0.994	4	0.000
WM.RENDERBUFFER_SIZE	11	0.080	0.003	0.995	5	0.000
WG.STENCIL_BITS	5	0.016	0.000	0.997	1	0.000
WM.TEXTURE_IMAGE_UNITS	7	0.033	0.000	0.997	2	0.000
WM.VERTEX_ATTRIBS	8	0.017	0.000	0.997	2	0.000
WM.VERTEX_TEXTURE_IMAGE_UNITS	9	0.057	0.001	0.996	2	0.000
WI.COLOR_READ_TYPE	6	0.041	0.000	0.996	4	0.000
WM.TEXTURE_MAX_ANISOTROPY_EXT	9	0.029	0.000	0.997	2	0.001
WG.getContextAttributes()	54	0.114	0.009	0.995	138	0.000
WG.getSupportedExtensions()	535	0.209	0.027	0.990	401	0.008
WG.[...].UNMASKED_VENDOR_WEBGL	27	0.115	0.000	0.995	9	0.000
WG.[...].UNMASKED_RENDERER_WEBGL	3,786	0.268	0.073	0.991	20	0.000
WebGL precision format	25	0.071	0.001	0.996	114	0.001
Our designed WebGL canvas	1,158	0.263	0.023	0.990	64	0.041
Width and position of a created `div`	17,832	0.324	nan	0.940	18	0.086
Colors of layout components	7,707	0.153	nan	0.986	492	0.090
Size of bounding boxes of a created `div`	16,396	0.369	nan	0.470	31	0.195
Presence of fonts	17,960	0.305	0.110	0.996	198	0.456
Support of video codecs	84	0.114	0.001	0.999	78	0.002
Support of audio codecs	52	0.128	0.002	0.999	61	0.001
Support of streaming codecs	50	0.132	0.010	0.999	133	0.002
Support of recording codecs	7	0.069	0.000	0.999	140	0.001
W.speechSynthesis.getVoices()	3,967	0.204	0.034	0.945	250	0.550
N.plugins	314,518	0.394	0.100	0.950	134	0.001
N.mimeTypes	174,876	0.311	0.017	0.982	112	0.000
A.state	5	0.082	0.000	0.999	7	0.000
A.sampleRate	16	0.070	0.019	0.997	5	0.000
AD.channelCount	5	0.036	0.000	1.000	1	0.000
AD.channelCountMode	4	0.036	0.000	1.000	8	0.000
AD.channelInterpretation	4	0.036	0.000	1.000	8	0.000
AD.maxChannelCount	20	0.058	0.003	1.000	1	0.000

Attribute	Values	N. Ent.	MNCE	% Same	Size	Time
AD.numberOfInputs	3	0.035	0.000	1.000	1	0.000
AD.numberOfOutputs	3	0.035	0.000	1.000	1	0.000
AA.channelCount	5	0.067	0.000	1.000	1	0.001
AA.channelCountMode	5	0.037	0.000	1.000	3	0.000
AA.channelInterpretation	4	0.036	0.000	1.000	8	0.000
AA.numberOfInputs	4	0.036	0.000	1.000	1	0.000
AA.numberOfOutputs	4	0.036	0.000	1.000	1	0.000
AA.fftSize	3	0.035	0.000	1.000	4	0.000
AA.frequencyBinCount	3	0.035	0.000	1.000	4	0.000
AA.maxDecibels	3	0.035	0.000	1.000	3	0.000
AA.minDecibels	3	0.035	0.000	1.000	4	0.000
AA.smoothingTimeConstant	4	0.046	0.000	0.998	3	0.000
Audio fp simple	337	0.153	0.004	0.958	18	1.403
Audio fp advanced	561	0.147	0.001	0.953	17	1.636
Audio fp advanced frequency data	546	0.161	0.011	0.950	17	1.639
Our designed HTML5 canvas (PNG)	269,874	0.420	0.021	0.922	64	0.260
Our designed HTML5 canvas (JPEG)	205,005	0.399	0.001	0.936	64	0.265
HTML5 canvas inspired by AmIUnique (PNG)	8,948	0.353	0.002	0.986	64	0.031
HTML5 canvas inspired by AmIUnique (JPEG)	6,514	0.312	0.001	0.989	64	0.039
HTML5 canvas similar to Morellian (PNG)	41,845	0.385	0.034	0.947	64	0.037
Accept HTTP header	26	0.028	0.000	0.997	3	0.000
Accept-Encoding HTTP header	30	0.019	0.002	1.000	13	0.000
Accept-Language HTTP header	2,833	0.124	0.022	0.999	35	0.000
User-Agent HTTP header	20,961	0.350	0.002	0.978	108	0.000
Accept-Charset HTTP header	18	0.002	0.000	1.000	1	0.000
Cache-Control HTTP header	47	0.055	0.023	0.706	1	0.000
Connection HTTP header	2	0.000	0.000	1.000	5	0.000
TE HTTP header	2	0.000	0.000	1.000	1	0.000
Upgrade-Insecure-Requests HTTP header	2	0.000	0.000	1.000	1	0.000
X-WAP-Profile HTTP header	4	0.000	0.000	1.000	1	0.000
X-Requested-With HTTP header	151	0.004	0.000	1.000	1	0.000
X-ATT-DeviceId HTTP header	1	0.000	0.000	1.000	1	0.000
X-UIDH HTTP header	1	0.000	0.000	1.000	1	0.000
X-Network-Info HTTP header	4	0.000	0.000	1.000	1	0.000
Via HTTP header	4,272	0.007	0.003	0.999	1	0.000
Any conditional HTTP headers	5,394	0.095	0.042	0.899	192	0.000
Number of bounding boxes	15	0.062	0.008	0.998	1	0.195
Number of plugins	54	0.147	0.000	0.984	1	0.001
Number of WebGL extensions	28	0.176	0.000	0.991	2	0.008
Width and height of first bounding box	12,937	0.350	nan	0.486	30	0.195
Width and height of second bounding box	1,332	0.103	nan	0.941	1	0.195
Width and height of third bounding box	772	0.076	nan	0.965	1	0.195
List of widths of bounding boxes	6,690	0.299	nan	0.986	16	0.195
List of heights of bounding boxes	2,222	0.264	nan	0.474	14	0.195
Width of first bounding box	4,418	0.281	0.038	0.987	14	0.195
Height of first bounding box	1,848	0.246	0.070	0.490	14	0.195

Attribute	Values	N. Ent.	MNCE	% Same	Size	Time
Width of second bounding box	471	0.085	0.002	0.998	1	0.195
Height of second bounding box	398	0.088	0.007	0.941	1	0.195
Width of third bounding box	224	0.060	0.004	0.999	1	0.195
Height of third bounding box	343	0.064	0.000	0.966	1	0.195
Width of a created `div`	15,473	0.316	0.007	0.940	6	0.086
Origin of a created `div`	16,375	0.316	0.008	0.942	11	0.086
Width of fallback font text	1,029	0.184	0.024	0.998	5	0.100
Height of fallback font text	1,159	0.164	0.010	0.998	5	0.100
Color of ActiveBorder element	702	0.078	0.005	1.000	18	0.090
Color of ActiveCaption element	475	0.073	0.002	1.000	18	0.090
Color of AppWorkspace element	321	0.067	0.001	1.000	18	0.090
Color of Background element	2,917	0.074	0.017	1.000	16	0.090
Color of ButtonFace element	297	0.079	0.004	1.000	18	0.090
Color of ButtonHighlight element	264	0.058	0.000	1.000	18	0.090
Color of ButtonShadow element	343	0.076	0.001	1.000	18	0.090
Color of ButtonText element	104	0.004	0.000	1.000	12	0.090
Color of CaptionText element	123	0.014	0.000	1.000	12	0.090
Color of GrayText element	333	0.071	0.004	1.000	18	0.090
Color of Highlight element	1,088	0.097	0.016	0.987	17	0.090
Color of HighlightText element	89	0.049	0.001	1.000	18	0.090
Color of InactiveBorder element	334	0.060	0.000	1.000	18	0.090
Color of InactiveCaption element	441	0.062	0.001	1.000	18	0.090
Color of InactiveCaptionText element	265	0.088	0.006	0.999	15	0.090
Color of InfoBackground element	239	0.057	0.000	1.000	18	0.090
Color of InfoText element	96	0.003	0.000	1.000	12	0.090
Color of Menu element	376	0.087	0.004	1.000	18	0.090
Color of MenuText element	124	0.020	0.001	1.000	12	0.090
Color of Scrollbar element	275	0.072	0.000	1.000	18	0.090
Color of ThreeDDarkShadow element	75	0.071	0.001	1.000	18	0.090
Color of ThreeDFace element	297	0.062	0.000	1.000	18	0.090
Color of ThreeDHighlight element	261	0.048	0.000	1.000	18	0.090
Color of ThreeDLightShadow element	280	0.074	0.001	1.000	18	0.090
Color of ThreeDShadow element	339	0.075	0.000	1.000	18	0.090
Color of Window element	329	0.019	0.001	1.000	18	0.090
Color of WindowFrame element	140	0.069	0.000	1.000	18	0.090
Color of WindowText element	107	0.004	0.000	1.000	12	0.090