JAISCR, 2020, Vol. 10, No. 4, pp. 243 – 253
10.2478/jaiscr-2020-0016

BROWSER FINGERPRINT CODING METHODS INCREASING THE EFFECTIVENESS OF USER IDENTIFICATION IN THE WEB TRAFFIC

Marcin Gabryel[1,*], Konrad Grzanek[2], Yoichi Hayashi[3]

[1]*Department of Computer Engineering,*
Czestochowa University of Technology,
al. Armii Krajowej 36, 42-200 Częstochowa, Poland

[2]*Information Technology Institute,*
University of Social Sciences, 90 - 113 Lodz
Clark University, Worcester, MA 01610, USA

[3]*Department of Computer Science, Meiji University, Japan*

[*]*E-mail: marcin.gabryel@pcz.pl*

Submitted: 14th October 2019; Accepted: 29th April 2020

Abstract

Web-based browser fingerprint (or device fingerprint) is a tool used to identify and track user activity in web traffic. It is also used to identify computers that are abusing online advertising and also to prevent credit card fraud. A device fingerprint is created by extracting multiple parameter values from a browser API (e.g. operating system type or browser version). The acquired parameter values are then used to create a hash using the hash function. The disadvantage of using this method is too high susceptibility to small, normally occurring changes (e.g. when changing the browser version number or screen resolution). Minor changes in the input values generate a completely different fingerprint hash, making it impossible to find similar ones in the database. On the other hand, omitting these unstable values when creating a hash, significantly limits the ability of the fingerprint to distinguish between devices. This weak point is commonly exploited by fraudsters who knowingly evade this form of protection by deliberately changing the value of device parameters. The paper presents methods that significantly limit this type of activity. New algorithms for coding and comparing fingerprints are presented, in which the values of parameters with low stability and low entropy are especially taken into account. The fingerprint generation methods are based on popular Minhash, the LSH, and autoencoder methods. The effectiveness of coding and comparing each of the presented methods was also examined in comparison with the currently used hash generation method. Authentic data of the devices and browsers of users visiting 186 different websites were collected for the research.

Keywords: browser fingerprint, device fingerprint, LSH algorithm, autoencoder

1 Introduction

The process of identifying the users and their behavior on the Internet is rather commonplace. By collecting information about the user, the pages they visit, or their planned purchases, it is possible to develop a useful profile, for example, when communicating advertising content to them. The mechanism for user identification is that of storing a unique identifier in a cookie on their device [1]. This type of method is widely criticized for the possibility of privacy violation. This problem has been noticed by the European Parliament and for several years now, information about the use of cookies has had to be placed on websites [2]. However, despite the obvious need for privacy, unambiguous identification of the device or the browser is extremely helpful in ensuring security and preventing abusive practices on the Internet. Numerous fraudulent activities include credit card payment scams [3], artificial generation of Internet traffic, the so-called abusive traffic [4], generating click fraud [5, 6] or automating the collection of web site contents [7]. Based on various reports [8, 9], losses due to fraud in online advertising alone can range from 6.5 to 19 billion dollars. Fraud prevention is primarily about identifying the perpetrator and blocking their actions. Unfortunately, the HTTP protocol used on the Internet only allows the user to be identified by the imperfect cookie mechanism. It is very easy to get around it by removing or ignoring cookies. Other methods, such as trying to eliminate fraud by blocking IP numbers of devices, do not give the expected results. One public IP number can be used by multiple users simultaneously. An IP number can also be assigned dynamically. Connections via proxy servers or VPN are also common. Hence, the best solution is to identify the user with the so-called browser fingerprint [10].

A browser fingerprint is a set of information about a given browser, device, operating system and environmental and location settings of the user [11]. Due to a great variety of this information, fingerprint can be treated as a unique identifier. Unfortunately, the values of the parameters collected may change over time. This happens when updating the browser or operating system version, resizing the screen, installing a new plugin, etc. Once a fingerprint has been obtained, on average, it remains valid for several days [12]. That is why developing a way of comparing fingerprints taking into account the changes occurring in them poses a rather demanding challenge. Using hash functions to encode and search for a fingerprint is no longer relevant, as hash functions generate completely different hash values for minor input changes, which is particularly undesirable in the case of fingerprinting. For example, once the browser version is changed, the same user will be recognized as two different persons.

The aim of this paper is to propose new methods of encoding fingerprint browser features and a method of comparing them, taking into account changes in the values of these features. The paper presents a research project showing the dynamics of changes taking place in the fingerprint during subsequent visits of a given user to the website. The features will be divided into a group of stable parameters and a group of unstable ones. This knowledge will be used to develop two new methods of comparing fingerprints:

- a new method of hash generation using the MinHash technique compatible with the LSH algorithm,
- a dedicated neural network with an auto-encoder structure to encode the abovementioned two groups of fingerprint features.

The encoding effectiveness of both methods will be experimentally tested using the data obtained over a period of 3 months from 186 different websites.

This paper is organized as follows. Section 2 briefly discusses the related work highlighting the proposed methods: issues connected with the browser fingerprint, its practical applications and the possibilities offered by neural networks with the autoencoder structure for data encoding. Then, in Section 3, the definition of the browser fingerprint is given, the characteristics of the features selected for its creation are assessed and their changes over time are analyzed. Section 4 discusses the proposed algorithms for comparing browser fingerprints. The research tests showing their effectiveness are presented in Section 5. Section 6 concludes the paper and offers suggestions for future work.

2 Related works

In paper [13], it was noticed for the first time that there is a high likelihood of identifying the user through using various parameters extracted from the browser being used. However, the scale of the research at that time remained rather limited. Only in the next study [10], where the research was conducted on a larger number of users, was it shown that fingerprinting can become a unique identifier. Works on browser or device fingerprinting usually contain comprehensive information on specific features comprising the fingerprint [3, 10, 11, 12, 14]. The studies also focus on the analysis of stability of particular parameters [3, 12], examining different types of browsers [11, 12] or applied security features limiting the possibility of obtaining fingerprint features [12]. There are also analyses of creating unique fingerprints over the years [15]. Despite many recent changes in browsers, browser fingerprinting is still effective in user identification. The possibilities of creating cross-browser fingerprints when one user uses several browsers on one device are also investigated [16].

The subject of practical use of browser fingerprinting appears in the literature in several contexts including web tracking [10, 14], bot and fraud prevention [7] and augmented authentication [3]. Computer security companies commonly use this technique to detect bots and unusual activity on websites [17, 18]. In [7], it is shown that fingerprinting is a good method of detecting crawler robots, but at the same time it can be bypassed with little effort. The paper [19] presents the capabilities offered by browser fingerprinting that can be used in order to verify the software and hardware stack of a mobile or desktop client. The presented system can, for example, distinguish between traffic sent by an original smartphone running an original browser from an emulator or desktop client deceptively simulating the same configuration.

A number of publications address issues related to the abuse of online advertising and e-commerce. They mainly concern the problem of click frauds and credit card payments. Paper [4] proposes two novel inference techniques which can isolate click fraud attacks. One of them detects patterns of click reuse within an ad network clickstream and the second method, the bait-click defense, leverages the vantage point of an ad network to inject a pattern of bait clicks into a user's device. Further on, the authors in [20] deal with the problem of detecting Internet merchant fraud. Goods or services offered and sold at cheap rates, but never shipped is a simple example of this type of fraud. The authors suggest a framework to detect such fraudulent sellers with the help of the support vector machine approach.

Methods of detecting fraud on the Internet with the use of deep learning neural networks are also one of the subjects of many academic papers. In paper [21], the authors present a method of detecting credit card fraud. The main contribution of their work is the development of a fraud detection system that employs a deep learning architecture together with an advanced feature engineering process based on homogeneity-oriented behavior analysis. In [22], an ensemble neural network adapted as a hacking detection system to protect the computer system against cyber-attacks is presented. The presented ensemble neural network consists of an autoencoder, a deep belief neural network, a deep neural network and an extreme learning machine. The system's task is to monitor the activity within a network of connected computers so as to analyze the activity of intrusive patterns. In [23], an attempt was made to detect fraud in biometric systems. To detect this type of fraud, the authors propose a novel method for fingerprint spoofing detection using the Deep Boltzmann Machines (DBM) for the extraction of high-level features from images.

A special kind of neural networks are autoencoders. They make it possible to use their deepest layer to encode input data. An example here is the so-called semantic hashing published in [24], where this technique of efficient information retrieval is presented. A document is fed to the input of a neural network which generates a small binary vector. Two similar documents will have two identical or very similar hashes. By indexing a given document with this hash, it is possible to find other similar documents almost immediately – one only should calculate the hash of a given document and search for documents containing the same hash (or hashes that differ from each other by from one to two bits).

3 Generating and analyzing the browser fingerprint parameters

3.1 Definition

A browser or device fingerprint is a set of data related to the user device. It contains information about the hardware, operating system and browser, and its configuration [11]. The information is collected only directly from the browser of a user by Javascript and by a web server. The user remains unaware that they are being identified, as the use of browser fingerprinting leaves no trace.

For a quick search and comparison, the collected data set is given onto the input of a hash algorithm. The hash is an alphanumeric string of fixed length characters which becomes a unique identifier of a given browser [25]. This kind of fingerprint does not work in the case of mass-produced devices with limited configuration and upgrade possibilities, such as smartphones. In the case of one model of the device, the collected data is identical, generating the same fingerprints. Another problem is the instability of some features. For example, the software or operating system versions are regularly updated and then generate new fingerprints, too.

3.2 Parameters extracted from the browser

As mentioned in Subsection 3.1, the data necessary for the browser fingerprint are extracted directly from the browser. To cunduct the research under this paper the following features were selected:

- the features of the device (including device memory, color depth, logic cores, touch support, screen parameters, audio parameters)
- operating system parameters (i.e., its version, list of fonts, time zone)
- features of the browser (including its version, list of plug-ins, language list, User-agent header, adblock information, database information, Web Storage mechanisms, screen resolution available, window resolution available, Do Not Track header)
- graphics card information (canvas fingerprint, WebGL renderer)

To calculate the level of identifying information in each of the fingerprint features mentioned above, the measure of entropy is used. The higher the entropy is, the more unique and identifiable a fingerprint. Let H be the entropy, X - a discrete random variable with possible values $x_1,...,x_n$ and $P(X)$ – a probability mass function. The entropy follows this formula

$$H(X) = -\sum_i P(x_i)\log_b P(x_i). \qquad (1)$$

For $b = 2$ it is the Shannon entropy and the result is in bits. The fingerprint's features, along with the calculated entropy, are presented in Table 1. The data came from 131,326 users who made 365,209 visits to different websites over a period of three months. The table does not include parameters with the entropy below 0.1. Some parameters, such as `screen_id` and `User-agent`, were broken down into individual elements. For `screen_id` it is `width`, `height`, `available_width` and `available_height`. In the case of `User-agent` the whole sequence was divided into elements starting with prefix `ua_`.

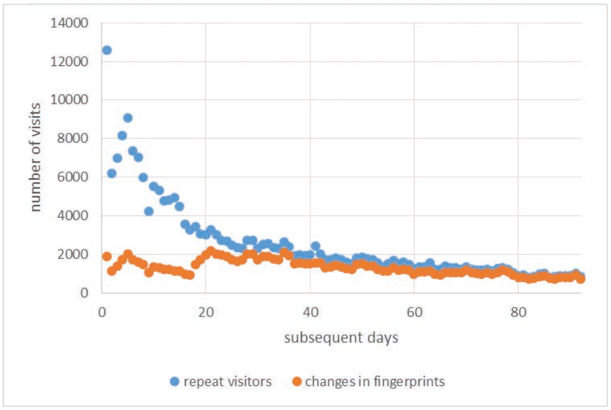

Figure 1. Changes in the browser fingerprints recorded daily in repeat visitors.

The same dataset was used to investigate the changes in the values of browser fingerprints. Figure 3.2 presents a graph showing the number of times that the website was accessed again over the following days. The graph also shows the number of users with a change in the value of at least one fingerprint feature. The graph shows that on the second day about 12,000 users returned to the website, and among them as many as 2,000 showed a change in at least one feature. The last day of the research recorded a return of approximately 1,000 users who had accessed the website on the first day of the project. All the repeat visitors displayed changes in at least one fingerprint feature. Figure 3.2 shows the percentage of visitors returning to the website over the following days. It also shows the percentage of users visiting the website on that day when each of them had a change in at least one feature. It can be seen on day 20 that 56% of the repeat visitors had already had a change of at least one feature. After 40 days, the changes had occurred in 80% of the investigated users. Table 1 shows for each feature the percentage of the users who had shown the changes. This means that the data can be divided into a stable data group and an unstable data group.

Table 1. Obtainable device fingerprint features

Feature	No. of bits of entropy	No. of changes in %	Group
device_memory	1.66	0.0	1
do_not_track_val_id	0.21	0.1	1
fonts	1.36	0.4	1
audio_params_id	0.50	1.2	2
webgl_vendor_id	1.66	0.0	1
webgl_renderer_id	5.35	0.1	1
logic_cores	1.18	0.0	1
platform	1.48	0.0	1
timezone	0.15	0.0	1
app_version	10.97	64.6	2
touch_enabled	0.68	0.0	1
max_touch_points	0.84	0.0	1
screen_id	4.78	1.7	-
width	2.60	1.2	2
height	4.32	1.1	2
av_width	2.63	1.4	2
av_height	4.58	1.4	2
adblock_enabled	0.26	0.6	1
canvas_2d_fingerprint	5.62	12.9	2
browser_plugins_hash	0.88	0.0	1
user-agent	10.98	64.7	-
br_version	4.26	62.9	2
os_version	2.8	4.9	2
app_version	10.97	64.4	2
platform	1.48	0.0	1
ua_device_brand_name	2.56	0.0	1
ua_device_model	6.51	0.0	1
ua_client_name	1.87	0.0	1
ua_client_version	4.76	63.3	2
ua_client_type	0.39	0.0	1
ua_device_type	1.08	0.0	1
ua_device_brand	2.56	0.0	1
ua_device_code	6.64	0.0	1
ua_os_name	0.78	0.0	1
ua_os_version	2.90	5.1	2
ua_preferred_client_name	2.32	0.2	1
ua_preferred_client_version	5.05	59.3	2
ua_preferred_client_type	0.31	0.2	1

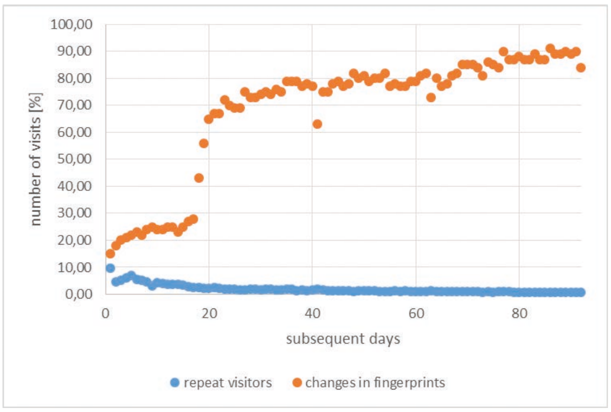

Figure 2. Percentage of changes in the browser fingerprints recorded daily in repeat visitors.

4 Browser fingerprint encoding methods

4.1 Hashing

Hashing refers to the process of generating a fixed-size output from an input of a variable size. This is done through the use of mathematical formulas known as hash functions (implemented as hashing algorithms). A conventional hash function should have collision resistance. This is a feature that prevents the same hash value from being obtained from two different input sets. These two properties of the hash function, a fixed hash length and collision resistance, enable a quick search of large data sets. Instead of comparing dozens of parameters for identical values, only hash values are compared with each other. The introduction outlines the disadvantages of using this solution for searching and comparing browser fingerprints. In the following Subsections, new methods for obtaining hashes and their use in searching for similar browser fingerprints are proposed.

4.2 The LSH algorithm

The main task of the Locality-Sensitive Hashing (LSH) algorithm is to quickly compare documents in terms of their contents. The documents do not need to be identical, as it is in the case of the hash function, because the proposed method is not resistant to minor changes in the document. The LSH algorithm consists of three steps:

– transforming the document into a set of characters of length *k* (the shingling method, also known as *k*-shingles or *k*-grams method),

– compressing the shingles set using the "Min-Hashing" method, so that the similarity of the base sets of documents in their compressed versions can still be checked,

– the LSH algorithm, which allows us to find the most similar pairs of documents or all pairs that are above some lower bound in similarity.

Shingling is an effective method of representing a document as a set. To generate the set, we need to select short phrases or sentences from the document – the so-called shingles. This causes documents to have many common elements in their sets even if the sentences appear in documents in a different order.

The next step consists in creating the so-called characteristic matrix, where the columns contain sets of shingles of individual documents, and the consecutive lines correspond to individual shingles. In the matrix cells at the intersection of row i and column j, there is value 1 in the case of the i-th shingle in the j-th document.

In the MinHash algorithm a so-called *SIG* signature matrix is created with the dimensions $m \times n$, where each of the m documents corresponds to n signatures. The matrix is calculated by performing random and independent n permutations of m rows of the characteristic matrix. The MinHash value for the column of the j-th document is the number of the first row (in the order resulting from the permutations), for which this column has value 1. These calculations are time-consuming, therefore instead of selecting random n row permutations, random n hash functions $h_1, h_2, \dots h_n$ are selected. The signature matrix is built taking into account each row in the given order. Let $SIG_{k,j}$ be an element of the signature matrix for the k-th hash function and column j of document d_j. The next steps in generating the signatures matrix are shown in Algorithm 1.

Algorithm 1 Algorithm for generating the signatures matrix.

```
1. Initially, set $SIG_{k,j}$ to $\infty$ for all values of $k$ and $j$.
2. For each $i$-th row of the characteristic matrix repeat points 2 and 3.
3. Calculate $h_1(j), h_2(j), \dots, h_n(j)$.
4. For each column $j$, check if there is 1 in row $i$.
   If yes, then for each $k = 1, 2, \dots, n$, set $SIG_{k,j} = \min(SIG_{k,j}, h_k(j))$.
```

The idea of the LSH algorithm allows checking the similarity of two elements. As a result of its operation, information is returned whether the pair forms a so-called "candidate pair", i.e. whether their similarity is greater than a specified threshold t (similarity threshold). Any pair that hashed to the same bucket is considered as a “candidate pair”. Dissimilar pairs that do hash to the same bucket are false positives. On the other hand, those pairs, which despite being similar, do not hash to the same bucket under at least one of the hash functions, are false negatives. A detailed description of particular parts of the LSH algorithm can be found in many works including [26] and [27].

In the proposed algorithm the i-th fingerprint parameters f_i need to be divided into the stable fs_i and unstable ones fn_i (see Section 3.2) and $f_i = \{fs_i, fn_i\}$. A MinHash algorithm starts operating for each set of parameters, which will generate two signature matrixes $SIGs_{k,j}$ and $SIGn_{k,j}$. The algorithm starts the search process twice, i.e. for the stable and unstable parameters. The following steps of the algorithm are presented in Algorithm 2. Its operation results in returning a set of fingerprints f_{qn} similar to fingerprint f_q.

Algorithm 2 The LSH algorithm for searching for similar browser fingerprints in relation to parameter stability.

```
Initial procedure:

1. Prepare two groups of parameters: stable ones $f_s = \{f_{s1}, \dots, f_{sm}\}$ and unstable ones
   $f_n = \{f_{n1}, \dots, f_{nm}\}$, where $m$ – number of fingerprints.
2. According to Algorithm 1 determine two signature matrices $SIGs_{k,j_s}$ and $SIGn_{k,j_n}$
   (for stable and unstable parameters, respectively) for two sets $d_s$ and $d_n$, $k = 1, \dots, m$,
   $j_s = 1, \dots, n_s$, $j_n = 1, \dots, n_n$, $n_s$, $n_n$ – number of signatures for stable and
   unstable parameters.
3. Determine the similarity thresholds $t_s$ and $t_n$ for stable and unstable parameters, respectively.

To find fingerprints $f_{qn}$ similar to fingerprint $f_q$ the following steps need to be carried out:

1. Start the LSH algorithm using the stable parameters $f_s$ to find similar candidate pairs $f_{qs}$
       $f_{qs} = \mathrm{LSH}(SIGs(f_q), SIGs(f_s), t_s)$,
   where: $SIGs(f_q)$, $SIGs(f_s)$ – signature matrix values for stable parameters obtained for the
   parameters of fingerprints $f_q$ and $f_s$.
2. Having found fingerprints $f_{qs}$ do one more search for a similar fingerprint, but this time
   using unstable parameters $f_n$
       $f_{qn} = \mathrm{LSH}(SIGn(f_q), SIGn(f_{qs}), t_n)$,
```

where: $SIGn(f_q)$, $SIGn(f_{qs})$ – signature matrix values for unstable parameters obtained for the parameters of fingerprints f_q and f_n.

3. Return obtained similar fingerprints f_qn.

4.3 Deep learning methods – autoencoders

An autoencoder is a neural network with at least one hidden layer. The input and output have the same size. The autoencoder is trained in such a way that the values given onto its input are to be copied onto its output. The encoder aims to compress data to a low-dimensional representation, while the decoder aims to reconstruct the input data from the low-dimensional representation generated by the encoder [28].

The learning set is $X = \{x_1, x_2, \ldots, x_N\} \in R^m$ where x_i is the m-dimensional feature vector, N – the number of samples. The encoder maps input vector x_i onto the hidden representation $h_i \in R^n$ using function f_θ as in

$$h_i = f_\theta(x_i) = s(Wx_i + b), \tag{2}$$

where $W \in R^{m\times n}$ is a set of weights, n is the number of units in the hidden layer h, $b \in R^n$ is a bias vector, θ is set $\{W, b\}$, and $s(\cdot)$ is the adopted activation function (sigmoid function) determined by the following formula

$$s(t) = \frac{1}{1+\exp^{-t}}. \tag{3}$$

The decoder maps back values h_i obtained in the hidden layer onto the output vector $y_i \in R^m$ according to the following formula

$$y_i = g_{\dot\theta}(x_i) = s(\dot W h_i + \dot b), \tag{4}$$

where $\dot W \in R^{n\times m}$ are weights, $\dot b \in R^m$ is a bias vector and $\dot\theta = \{\dot W, \dot b\}$.

The learning of the autoencoder consists in minimizing the difference between input x_i and output y_i. To this end a loss function is calculated as shown in the following formula

$$L(x_i, y_i) = \|x_i - y_i\|^2 = \|x_i - s(\dot W s(Wx_i + b) + \dot b)\|^2. \tag{5}$$

The aim of the learning is thus finding optimum values of parameters θ and $\dot\theta$ facilitating the minimizing of the error between the input and output for the whole training set

$$\theta, \dot\theta = \arg\max_{\theta,\dot\theta} L(x, y). \tag{6}$$

The autoencoder presented above is discussed in relation to continuous data x. In the case of categorical data, one-hot encoded data are given into the input. In the case of one variable x of the categorical type, input dimension m is equal to the number of categories. Each category is numbered with consecutive natural numbers. Single vector x_i is filled with zeros and contains a single value 1 in place j, where j is the category number. For this type of data function $s(\cdot)$ in formula (3) will take the value of softmax [29], where for each of the outputs k

$$s(t)_k = \frac{e^{t_k}}{\sum_{j=1}^{m} e^{t_j}}, \tag{7}$$

where $t \in R^m$, and t_j is j-th element of vector t. For the softmax function, the loss function $L(x_i, y_i)$ is calculated by using the categorical cross-entropy

$$L(x_i, y_i) = -\sum_{j=1}^{m} x_{ij} \log(y_{ij}), \tag{8}$$

where m is the number of outputs (number of categories).

For encoding stable and unstable fingerprint features, there are two hashes required. The autoencoder will, therefore, consist of two pairs of encoder-decoders for stable and unstable parameters, respectively. The autoencoder will thus have two hidden layers, h_1 and h_2, whose outputs will return the values of the fingerprint hashes given onto the network input. The diagram of such an autoencoder is shown in Figure 3.

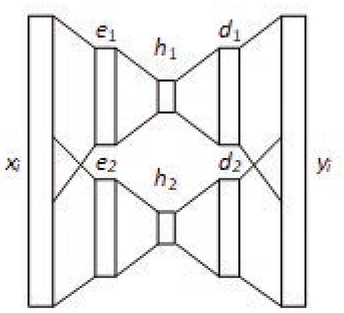

Figure 3. Dedicated structure of autoencoders.

There is a training set $x_i \in x_{1j}, x_{2k} \in R^m$, where $x_{1j} \in R^{m_1}$, $x_{2k} \in R^{m_2}$ – stable and unstable input data, m_1 – the number of stable data, m_2 – the number of unstable data, $m = m_1 + m_2$. The values of the hidden layers will then be

$$h_{1j} = f_{\theta_1}(x_{1j}) = s(W_1 x_{sj} + b_1), \tag{9}$$

and

$$h_{2j} = f_{\theta_2}(x_{1j}) = s(W_2 x_{nk} + b_2), \tag{10}$$

where: $W_1 \in R^{m_1 \times n_1}$, $W_2 \in R^{m_2 \times n_2}$ – weights of the hidden layers, n_1, n_2 is the number of units in the hidden layer h_{1j} and h_{2k}, $b_1 \in R^{n_1}$, $b_2 \in R^{n_2}$ – bias vectors, $\theta_1 = \{W_1, b_1\}$ and $\theta_2 = \{W_2, b_2\}$. The output value $y_i \in \{y_{1j}, y_{2k}\} \in R^m$ of the autoencoder will require certain calculations

$$y_{1j} = g_{\dot{\theta}_1}(x_{1j}) = s(\dot{W}_1 h_{1j} + \dot{b}_1) \tag{11}$$

and

$$y_{2k} = g_{\dot{\theta}_2}(x_{2k}) = s(\dot{W}_2 h_{2j} + \dot{b}_2). \tag{12}$$

where: $\dot{W}_1 \in R^{n_1 \times m_1}$, $\dot{W}_2 \in R^{n_2 \times m_2}$ are the weights, $\dot{b}_1 \in R^{m_1}$, $\dot{b}_2 \in R^{m_2}$ are the bias vectors and $\dot{\theta}_1 = \{\dot{W}_1, \dot{b}_1\}$ and $\dot{\theta}_2 = \{\dot{W}_2, \dot{b}_2\}$. For this autoencoder the loss function will be expressed by the following formula

$$\begin{aligned} L(x_i, y_i) &= \frac{L(x_{1i}, y_{1i}) + L(x_{2i}, y_{2i})}{2} \\ &= \frac{\|x_{1i} - y_{1i}\|^2 + \|x_{2i} - y_{2i}\|^2}{2} \\ &= \frac{\|x_{1i} - s(\dot{W}_1 h_{1j} + \dot{b}_1)\|^2 + \|x_{2i} - s(\dot{W}_2 h_{2j} + \dot{b}_2)\|^2}{2}. \end{aligned} \tag{13}$$

After the learning process has been completed, only encoders are used for further research. They are treated as hash functions. The inputs of the encoders are given fingerprint feature values (as one-hot encode data). The output values obtained by the two layers h_1 and h_2 are rounded to integers.

5 Experiments

For this research on browser fingerprints coding methods, authentic data were collected during visits to 186 different types of websites including online shops, companies offering various services and financial institutions, including banks. For 3 months, the total number of registered clicks totaled ca. 45 million. For the sake of the research, the data for which the fingerprints parameters changed5 to 9 times in one user within the three months were selected. The number of the data filtered in this way amounted to ca. 213,000. About 33,000 unique users were identified. The data containing fingerprints were then divided in the 80/20 ratio into training and test groups. Most of the data were used to initiate the MinHash algorithm or to learn the neural network. The fingerprints from the test group were used to check the effectiveness of the search.

Precision and *recall* were used to evaluate the effectiveness of the tests [30]. *Precision* is the ratio of the number of correctly classified data to the total number of irrelevant and relevant data classified

$$precision = \frac{tp}{tp + fp}, \tag{14}$$

and *recall* is the ratio between the number of data that are correctly classified to the total number of positive data

$$recall = \frac{tp}{tp + fn}, \tag{15}$$

where tp – true positive, fp – false positive, fn – false negative and they can be derived from a confusion matrix [30]. The parameter which combines the above two parameters is $F1$ score that it is the harmonic mean of *precision* and *recall*

$$F1 \text{ score} = \frac{2 \cdot precision \cdot recall}{precision + recall}. \tag{16}$$

The first of the conducted experiments consists in testing the effectiveness of Algorithm 2. The initiating part of this algorithm requires that the data that constitute the browser fingerprint are divided into two groups, i.e. stable and unstable (according to the results obtained in Section 3.2). The initial values of the algorithm parameters must then be determined: the number of signatures for encoding n_s, n_n and threshold values t_s, t_n. The selection of these values requires a number of tests with different combinations of these parameters. The following values have been adopted for each parameter of the algorithm: n_s, $n_n \in \{2, 4, 8, 16, 24, 32, 64, 96, 128\}$, $t_s = 1$, $t_n \in \{1, 0.9, 0.8, 0.7, 0.6, 0.5\}$. The best values were obtained for $n_s = 128$. Table 2 presents a summary of the obtained results. The following rows show the results for different t_n values, while the columns show the results for different n_n values. The best was a set of parameters $n_s = 128$, $n_n = 4$, $t_s = 1$ and $t_n = 0.5$ lub $t_n = 0.6$. Here the value

$F1 = 0.35$ for $precision = 0.34$ and $recall = 0.36$ was obtained.

Table 2. The $F1$ score obtained for $n_s = 128$ and $t_s = 1$.

	n_n									
t_n	2	4	8	16	24	32	48	64	96	128
1	0.298	0.297	0.281	0.286	0.287	0.289	0.290	0.290	0.290	0.290
0.9	0.298	0.297	0.281	0.285	0.279	0.283	0.282	0.284	0.283	0.283
0.8	0.298	0.297	0.282	0.278	0.276	0.276	0.275	0.275	0.274	0.274
0.7	0.298	0.297	0.289	0.284	0.286	0.285	0.277	0.273	0.272	0.270
0.6	0.298	**0.350**	0.317	0.293	0.300	0.285	0.285	0.281	0.279	0.273
0.5	0.321	**0.350**	0.345	0.327	0.334	0.305	0.305	0.306	0.285	0.287

The next experiment concerned the use of a neural network with a dedicated structure consisting of two autoencoders (see Section 4.3). Several possible combinations of neural network hyperparameters were tested. Each time a different number of encoder and decoder layers, units in each layer and the size of the deepest coding layers h_1 and h_2. were selected. The optimization parameters were assumed as follows: categorical cross-entropy as the loss function, stochastic gradient descent optimizer, number of epochs – 250 and batch size 32. The $F1$ score values for different network structures are presented in Table 3. The subsequent rows show the examined structures of neural networks, specified number of units for encoding layers h_1 and h_2, and the $F1$ score value obtained for the test data. The best results were achieved by network No. 4 considering the obtained $F1$ score and network size.

Table 3. The $F1$ score results obtained for different structures of autoencoders.

No.	Type of the parameter group	Structure	No. of units of layer h	$F1$ score
1	1	512-256-64-256-512	64	0.327
	2	512-256-4-256-512	4	
2	1	512-256-32-256-512	64	0.303
	2	512-256-8-256-512	4	
3	1	640-368-128-368-640	128	0.361
	2	256-128-8-128-256	8	
4	1	640-368-96-368-640	96	0.377
	2	256-128-8-128-256	8	
5	1	256-128-32-128-256	32	0.298
	2	256-128-8-128-256	8	

The best results of the two coding and fingerprint benchmarking methods presented in this paper are presented in Table 4. The results are compared with the commonly used hashing method (see Section 4.1). Two experiments using hash function SHA1 were conducted. In the first case, stable and unstable features were given onto the hash function input. In the second case only stable features were given. The new methods proposed in the paper give much better results than commonly used hashing methods.

Table 4. Comparison of the best results obtained.

Method	The best $F1$ score
hashing stable and unstable parameters (see 4.1)	0.276
hashing stable parameters	0.202
MinHash algorithms with LSH (see 4.2)	0.350
autoencoders (see 4.3)	0.377

6 Conclusion

The paper presents two new methods of encoding and comparing browser fingerprints: an algorithm using MinHash encoding (cooperating) with the LSH and a dedicated structure of a neural network consisting of two encoders. Both methods use the results of a previously conducted analysis of changes in fingerprint characteristics over time. This allowed selecting two groups of features: stable and unstable ones. The presented experimental results showed that the effectiveness of searching for similar fingerprints is much greater if the difference between these two groups is taken into account when encoding.

A great advantage which the proposed methods offer is the possibility to easily save the results of encoding in the database. This is possible both with the hashes obtained by MinHash algorithms and with the hashes obtained by the h_1 and h_2 autoencoder layers. This gives the possibility to put both algorithms into practice. Both methods can also be used when creating device fingerprints for different browsers used by one user [16]. In this case, it will be necessary to perform an appropriate analysis of the changes occurring in the fingerprint feature values beforehand and create appropriate groups of stable and unstable features.

Browser fingerprinting is a tool that helps to reduce the number of fraudulent activities on the Internet. The new algorithms proposed in this paper may increase the level of detecting online fraudulent activities being carried out by some users by identifying them more accurately and efficiently.

References

[1] Kristol D.M., HTTP cookies: Standards, privacy, and politics, ACM Trans. Internet Techn. 1 (2) (2001) 151–198.

[2] Low C., Cookie law explained, 2016. on-line https://www.cookielaw.org/the-cookie-law/ (retrieved:03/2020).

[3] Alaca, F., Van Oorschot, P. C. (2016, December). Device fingerprinting for augmenting web authentication: classification and analysis of methods. In Proceedings of the 32nd Annual Conference on Computer Security Applications (pp. 289-301).

[4] Nagaraja, S., Shah, R. (2019, May). Clicktok: click fraud detection using traffic analysis. In Proceedings of the 12th Conference on Security and Privacy in Wireless and Mobile Networks (pp. 105-116).

[5] Mouawi, R., Elhajj, I.H., Chehab, A. et al. Crowdsourcing for click fraud detection. EURASIP J. on Info. Security 2019, 11 (2019)

[6] Dave, V., Guha, S., Zhang, Y. (2012, August). Measuring and fingerprinting click-spam in ad networks. In Proceedings of the ACM SIGCOMM 2012 conference on Applications, technologies, architectures, and protocols for computer communication (pp. 175-186).

[7] Vastel, A., Rudametkin, W., Rouvoy, R., Blanc, X. (2020, February). FP-Crawlers: Studying the Resilience of Browser Fingerprinting to Block Crawlers. In NDSS Workshop on Measurements, Attacks, and Defenses for the Web (MADWeb'20).

[8] 2019. https://www.emarketer.com/content/digital-ad-fraud-2019

[9] Barker S. ,"Future Digital Advertising, Artificial Intelligence & Advertising Fraud 2019-2023", Juniper Research, 2019

[10] Eckersley P., How unique is your web browser? in: Privacy Enhancing Technologies, 10th International Symposium, PETS 2010, Berlin, Germany, July 21-23, 2010. Proceedings, 2010, pp. 1–18

[11] Laperdrix, P., Bielova, N., Baudry, B., Avoine, G. (2019). Browser Fingerprinting: A survey. arXiv preprint arXiv:1905.01051.

[12] Kobusinska, A., Pawluczuk, K., Brzezinski, J. (2018). Big Data fingerprinting information analytics for sustainability. Future Generation Computer Systems, 86, 1321-1337.

[13] Mayer J R. 2009. Any person... a pamphleteer": Internet Anonymity in the Age of Web 2.0. Undergraduate Senior Thesis, Princeton University (2009).

[14] Steven E. and Arvind N. 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.

[15] Gómez-Boix, A., Laperdrix, P., Baudry, B. (2018, April). Hiding in the crowd: an analysis of the effectiveness of browser fingerprinting at large scale. In Proceedings of the 2018 world wide web conference (pp. 309-318).

[16] Cao, Y., Li, S., Wijmans, E. (2017, March). (Cross-) Browser Fingerprinting via OS and Hardware Level Features. In NDSS.

[17] 2020. The Evolution of Hi-Def Fingerprinting in Bot Mitigation - Distil Networks. https://resources.distilnetworks.com/all-blog-posts/device-fingerprinting-solution-bot-mitigation

[18] 2020. Device Tracking Add-on for minFraud Services - MaxMind. https://dev.maxmind.com/minfraud/device/

[19] Bursztein, E., Malyshev, A., Pietraszek, T., Thomas, K. (2016, October). Picasso: Lightweight device class fingerprinting for web clients. In Proceedings of the 6th Workshop on Security and Privacy in Smartphones and Mobile Devices (pp. 93-102).

[20] Renjith, S. (2018). Detection of Fraudulent Sellers in Online Marketplaces using Support Vector Machine Approach. arXiv preprint arXiv:1805.00464.

[21] Zhang, X., Han, Y., Xu, W., Wang, Q. (2019). HOBA: A novel feature engineering methodology for credit card fraud detection with a deep learning architecture. Information Sciences.

[22] Ludwig, S. A. (2019). Applying a neural network ensemble to intrusion detection. Journal of Artificial Intelligence and Soft Computing Research, 9(3), 177-188.

[23] de Souza, G. B., da Silva Santos, D. F., Pires, R. G., Marana, A. N., Papa, J. P. (2019). Deep features extraction for robust fingerprint spoofing attack detection. Journal of Artificial Intelligence and Soft Computing Research, 9(1), 41-49.

[24] Salakhutdinov, R., Hinton, G. (2009). Semantic hashing. International Journal of Approximate Reasoning, 50(7), 969-978.

[25] 2020. FingerprintJS. Fraud detection API. https://fingerprintjs.com/

[26] Leskovec J., Rajaraman A., Ullman J.D.: Mining of Massive Datasets, Cambridge University Press, 2014

[27] Azgomi, H., Mahjur, A. (2013). A Solution for Calculating the False Positive and False Negative in LSH Method to Find Similar Documents. Journal of Basic and Applied Research, 3, 466-472.

[28] Goodfellow, Ian; Bengio, Yoshua; Courville, Aaron (2016). Deep Learning. MIT Press

[29] Bengio Y., Learning deep architectures for ai Found. Trends Mach. Learn., vol. 2, no. 1, pp. 1–127, Jan. 2009.

[30] Olson, D.L., Delen, D.: Advanced Data Mining Techniques, 1st edn. Springer, Heidelberg (2008).

Marcin Gabryel earned his Ph.D. degree in computer science at Czestochowa University of Technology, Poland, in 2007. He is an assistant professor in the Department of Computer Engineering at Częstochowa University of Technology. His research focuses on developing new methods in computational intelligence and data mining. He has published over 50 research papers. His present research interests include deep learning architectures and their applications in databases and security.

Konrad Grzanek, scientist, programmer and lecturer. Graduate of the Technical University of Łódź (FTIMS). Assistant professor at the Social Academy of Sciences. He holds a Ph.D. from Częstochowa University of Technology (CUT). His research interests focus on programming languages, software quality, software development processes, and artificial intelligence, in particular on combining machine learning methods with static software analysis. As a programmer, he is an advocate and promoter of the functional programming style. Author of over 30 publications related to various problems of computer science and software engineering.

Prof. **Yoichi Hayashi** received the Dr. Eng. degree in systems engineering from Tokyo University of Science, Tokyo in 1984. In 1986, he joined the Computer Science Department of Ibaraki University, Japan, as an Assistant Professor. Since 1996, he has been a Full Professor at Computer Science Department, Meiji University, Tokyo. He was a visiting professor at the University of Alabama at Birmingham and University of Canterbury (New Zealand). He authored over 230 published computer science papers. His current research interests include explainable AI, deep learning, rule extraction, high-performance classifiers, and medical informatics and medical imaging. He has been the Action Editor of Neural Networks and the Associate Editor of IEEE Trans. Fuzzy Systems. He has served as Editor-in-Chief and Associate Editor, Editorial Board Member, Review Board Member, Guest Editor and Reviewer in 60 academic journals, and was involved in the work for the European Research Council Executive Agency, National Sciences and Engineering Research Council of Canada (NSERC). He has been a senior member of the IEEE since 2000.