HAL open science

Keystroke Dynamics Anonymization System

Denis Migdal, Christophe Rosenberger

To cite this version:

Denis Migdal, Christophe Rosenberger. Keystroke Dynamics Anonymization System. SeCrypt International Conference on Security and Cryptography, Jul 2019, Prague, Czech Republic. hal-02126985

HAL Id: hal-02126985

https://hal.science/hal-02126985v1

Submitted on 11 Aug 2019

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Keystroke Dynamics Anonymization System

Denis Migdal[1] , Christophe Rosenberger[1]
[1] Normandie Univ, UNICAEN, ENSICAEN, CNRS, GREYC, 14000 Caen, France

{denis.migdal, christophe.rosenberger}@ensicaen.fr

August 11, 2019

Abstract

Keystroke Dynamics enables the authentication or identification of users by analyzing their way of typing, e.g. when browsing the Internet. Most studies in the state of the art focus on increasing Keystroke Dynamics Systems performances. In this paper, we address the issue of avoiding the biometric capture of keystroke dynamics in order to protect users' privacy. Authentication/identification, profiling can be considered as attacks we limit in this contribution. Experimental results obtained on significant datasets show the benefits of the proposed approaches.
Keywords: Keystroke Dynamics Anonymization System, Keystroke Dynamics, Anonymization, WebExtension, JavaScript, Browser, Browser Fingerprinting.

1 Introduction

Browser Fingerprinting aims at tracking users through their browser thanks to discriminant data a service can collect. This is usually proposed to "personalize services" corresponding to users profile-type, s.a. suggesting contents depending on the user's assumed Internet history. Browser Fingerprinting goal is not to identify users with assurance, but to classify the user into a category, e.g. by identifying a set of browsing sessions belonging to the same user, or type of users.

Panopticlick (Eckersley, 2010), IAmUnique (Laperdrix et al., 2016), and UniqueMachine (Cao and Wijmans, 2017) websites enable the computation of browser fingerprints from data collected by the website, generally through the network and a JavaScript API, to determine the fingerprint uniqueness among the previously computed. The more the browser fingerprint is unique, the more the service is able to discriminate the user. Information used for browser fingerprinting might be linked, e.g. to the hardware (e.g. GPU (Cao and Wijmans, 2017), screen), to the operating system, to the browser, its configuration, installed fonts (Eckersley, 2010; Laperdrix et al., 2016), browser history (Weinberg et al., 2011), or blacklisted domains (Boda et al., 2012). Such identification and tracking are often not consented by the user, and poses a threat to users' online privacy, thus, leading researcher and developers to study this issue and propose solutions in order to protect users' privacy (Laperdrix et al., 2016; Nikiforakis et al., 2015; Moore and Thorsheim, 2016; Eckersley, 2010; Acar et al., 2013).

Biometric capture can also be added to browser fingerprinting, e.g. using the mouse (Jorgensen and Yu, 2011; Shen et al., 2013) or/and Keystroke (Revett et al., 2007a; Giot et al., 2011; Kim et al., 2018). Such modality also enables to deduce private information about the user, s.a. his/her gender or to link some identities. In this study, we aim at protecting users' privacy by anonymizing keystroke, thus limiting browser fingerprinting and preventing deduction of private information about users, while still allowing use of this modality in authentication for consenting users.

As any biometric authentication solution, a keystroke dynamic system (KDS) is composed of two main modules: the enrollment and the verification modules. Each user must enroll himself/herself in the KDS in order to

compute its biometric reference template given multiple samples (*i.e.*, several inputs of the password) acquired during the enrollment step. For each input, a sequence of timing information is captured (*i.e.*, time when each key is pressed or released) from which some features are extracted (*i.e.*, latencies and durations) and used to learn the model which characterizes each user. During a verification request, the claimant types his/her password. The system extracts the features and compares them to the biometric reference template of the claimant. If the obtained distance is below a certain threshold, the user is accepted, otherwise he/she is rejected.

First works on KD have been done in the eighties (Gaines et al., 1980), although the idea of using a keyboard to automatically identify individuals has first been presented in 1975 (Spillane, 1975). In the preliminary report of Gaines *et al.* (Gaines et al., 1980), seven secretaries typed several paragraphs of text and researchers showed that it is possible to differentiate users with their typing patterns. Since then, several studies have been done, allowing to decrease the quantity of information needed to build the biometric reference, while improving the performances (Umphress and Williams, 1985; Monrose and Rubin, 2000; Revett et al., 2007b; Lee and Cho, 2007; Giot et al., 2011).

However, to the knowledge of the authors, no study has yet tried to decrease their performances in order to protect users' privacy against unwanted authentication/identification or profiling. Indeed, many papers (Giot and Rosenberger, 2012; Epp, 2010) have shown that some soft biometric characteristics (emotion, gender, age ...) can be extracted from keystroke dynamics features, internet services could profile users given a simple JavaScript code embedded in the web page.

The main contribution of this paper is to propose multiple simple solutions for internet users to decide whether its keystroke dynamics features could be used or not on a specific website. Using keystroke dynamics could be useful to enhance the security of authentication avoiding complex password (logical access control to a bank account as for example). For other services, such as social networks, an user might choose to disable the keystroke dynamics capture. The proposed methods have been implemented as a WebExtension as an operational proof of concept. With this WebExtension, any user can easily decide for which service, its keystroke dynamics features could be used or not (GDPR requirement).

The article is organized as follows. Section 2 provides some background information on the keystroke dynamics biometric modality. We describe the possible attack and the existing countermeasure in the literature. We propose new protection schemes in section 3. Their efficiency is illustrated through experimental results on significant datasets. An implemented WebExtension is presented in Section 4 and compared with the only existing solution. Section 5 concludes and gives some perspectives of this study.

2 Background

We present in this section some background information on keystroke dynamics.

2.1 Keystroke dynamics system

As the number of collected samples during the enrollment step is usually low, many Keystroke Dynamics Systems are based on a distance. In the scope of this article, we suppose that the attacker uses the Hocquet distance function (Hocquet et al., 2007): We aim at computing a score between two templates K_A and K_B. We suppose that the template K_A is associated by μ and σ the average value of biometric samples and the standard deviation (note that

Name	Text	# of users (45)	Clock resolution	EER	Source
GREYC K	greyc laboratory	104	10.0144ms	14.75%	(Giot et al., 2009)
GREYC W1	laboratoire greyc	62	1ms	14.40%	(Giot et al., 2012)
GREYC W2	sésame	46	1ms	25.39%	(Giot et al., 2012)
CMU	.tie5Roanl	51	0.2ms	19.38%	(Killourhy and Maxion, 2009)

Table 1: Description of used datasets.

0/0 is assumed to be 0).

$$Score = 1 - \frac{1}{n}\sum_{i=1}^{n} e^{-\frac{|K_B(i)-\mu_i|}{\sigma_i}} \quad (1)$$

In the scope of this paper, the templates are composed of the gap and dwell durations for each typed key, i.e. the duration between two consecutive key press, and the time a key is pressed. The 10 first templates of each user are used for the reference template computation.

2.2 Keystrole dynamics datasets

There exist many keystroke dynamics datasets (Monaco, 2018). We decided in this work to focus on fixed text datasets (i.e. where users typed the same passphrase). Datasets have been cleaned to remove incoherent data, e.g. entries in which the user did not type the asked text. This corresponds to 13% of entries in GREYC W, and less than 3 entries for other datasets.

In order to get comparable sets, only the first 45 entries per users is kept, users with less than 45 entries, and datasets with less than 45 users, are discarded. From the existing fixed-text datasets, only 3 matched our criteria. From these 3 datasets, we build 4 datasets composed of a fixed text Keystrokes for each user (one having 2 fixed text, 2 datasets are thus created). Table 1 gives the datasets used in this work.

2.3 Attacker model

The attacker is able to execute arbitrary JavaScript code on the users' browser in order to identify them, using only the keyboard events' timestamps. We assumed, in this paper, the typed text to be fixed, s.a. a login, e-mail, or password. The attack is also possible on free text.

The attacker is able to measure the timestamps of keyboard events she/he receives with the JavaScript function `Date.now()`. Thus, modifying the events' timestamps will have no effect, as the attacker can measure them himself. However, events can be delayed, i.e. waiting some time before sending the keyboard event. As JavaScript events loop is mono-threaded, any active wait is troublesome and will be easily detected by the attacker using `setInterval()`, thus requiring the delayed event to be destroyed, and recreated after a passive wait with `setTimeout()`.

The attacker has an *a priori* on the user's identity, and will be able to use any Keystroke Dynamics Systems, and to perform any pre-processing, in order to identify or profile him. The way the Keystroke Dynamics is protected, and the eventual parameters of such anonymization scheme is also assumed to be known by the attacker. Thus, such parameters should be fixed for all users in order to prevent the attacker from using them to discriminate users through browser fingerprinting techniques (Eckersley, 2010).

2.4 Countermeasures in the literature

In order to avoid the attack described previously, the internet user can disable the JavaScript capability of his/her internet browser. It has lots of usage consequences for him/her.

Otherwise, the main idea of protecting the user from identification/profiling given the keystroke dynamics data is to disturb the collected information. Very few works have been done in the state of the art to avoid the correct capture of keystroke dynamics on Internet. To our knowledge, there exists a single work implemented as a browser extension. KeyboardPrivacy(Moore and Thorsheim, 2016) is a Google Chrome extension that implements such a protection. Timestamp of each event is computed as follows:

$$t'_i = max(t'_{i-1}, t_i) + \begin{cases} b & \text{1 time out of 2} \\ 0 & \text{1 time out of 2} \end{cases}$$

Where b is a random value following an Uniform distribution between 0 and a (this value is user-defined).

2.5 Attack performance

The capacity of an attacker to authenticate an user will be quantified with the maximal estimation of the Equal Error Rate (EER), which corresponds to configuration of the biometric system when FAR equals FRR. The False Acceptance Rate (FAR) describes the ratio of accepted impostor data, the False Rejection Rate (FRR) describes the ratio of falsely rejected legitimate users.

The performance of a KD Anonymization Scheme (KDAS) will thus be quantified as the minimum of the maximal estimation of the EER for each possible KDS and pre-processing. For a given KDS and pre-preprocessing, if the KDAS is not deterministic, the

KDAS is tested 20 times, and the mean of the maximal estimation of the EER for each test is used. If the dataset is not indicated, the number given is the average number for each of the 4 datasets used in this study.

2.6 Attacker pre-processing

The timestamp of a given event depends of the resolution and jitter of the clock used to measure it. The *resolution* is the mean time between two clocks tics, and the *jitter*, the difference between the theoretical clock tic timestamp and its real timestamp. This mean that an event occurring at a time t will have a timestamp of $\lfloor t/r \rfloor * r + j$, where r is the clock resolution, and j a random noise (the jitter). Existing studies have found that the clock resolution influence KDS performances (Killourhy and Maxion, 2008), and discretization might improve KDS performances (Giot et al., 2011).

In the following, we illustrate the impact of discretization. Timestamps values have been discretized using 1,001 different resolutions (from 0 to 1 by step of 1/1,000). As shown in Figure 3, attacker might expect slight (J $\simeq$ 0.02 for GREYC W1) or negligible (J $<$ 0.005) EER improvement by doing so. Figure 1 shows the discretization can both increase or decrease the EER depending on the resolution. Figure 2 zooms on the area where the EER decreases.

Jitter can be removed with the following formula : $t' = \lceil t/r \rfloor * r$. As shown in Figure 3, it has negligible influence on the EER (diff $<$ 0.004, and noJ $\simeq$ J), and thus does not need to be removed.

In the next section, we propose new solutions to protect internet users against their identification/profiling through their keystroke dynamics.

3 Proposed protection schemes

We propose different solutions to anonymize keytroke dynamics of users. Their objective is to be able to use keystroke dynamics features for internet services when the user consents (for security applications), and to provide altered data otherwise (for privacy protection).

3.1 Costless protection

Release keyboard events can be automatically generated at a constant time after the pressure event e.g. 2ms (A). As shown in figure 4, such strategy increases significantly the EER ($0.044 \leq$ A ≤ 0.117).

Users' screen typically draw a frame every 1/60 seconds. Thus, in an ordinary use, the time an event occurred between two consecutive frames makes no difference to the users, i.e. any delay of an event to match the time of the next frame is *de facto* impossible to perceive for an user, and thus assumed costless. Such operation can be trivially done thanks to `Window.requestAnimationFrame()`.

As shown in Figure 4, automatically generating release events after delaying pressure events to the next frame (DA), gives slight increase of the EER. However, such strategy is interesting as it would suppress information that could be exploited by other KDS.

In the following, non-costless KDAS, pressure events will be delayed beforehand, and release events will be automatically generated afterward.

3.2 Non-blocking protection

In order to further increase the EER values, some events have to be delayed beyond the next frame. Such delay might be perceivable by the users and thus constitute a cost in terms of usability of the KDAS. This cost, we call latency, is computed as the maximal number of frames skipped during a typing of a given text.

Non-blocking KDAS delays pressure events independently from the previous, with the only constraint to preserve the events' order. Their parameters N are the number of frames that can be skipped, and *de facto* their latency.

Two non-blocking KDAS are studied. In the first, events are discretized with a resolution of $(N+1)/60$ (delay), and in the second, events are delayed by n frames with n an uniform discrete noise $n \sim U(0,N)$ (rdelay). These two KDAS were tested with 15 configurations, $N \in [\![0,14]\!]$ for delay, and $N \in [\![1,15]\!]$ for rdelay. As shown in Figure 5, both provide significant protection compared to the costless KDAS. However, for the same latency, rdelay seems always better than delay.

Moreover, delay improves the EER for N=1, compared

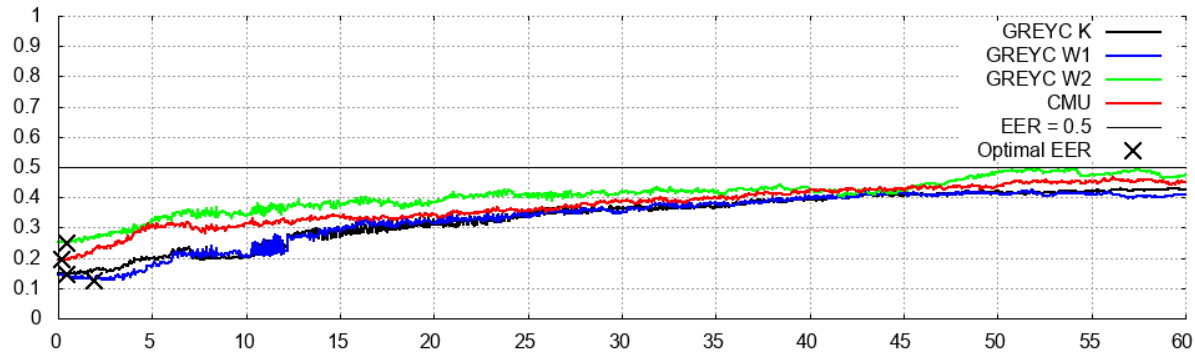

Figure 1: Impact of discretization on the performance of KDS.

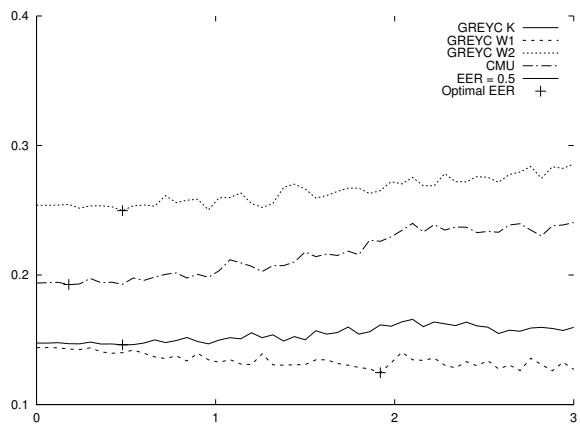

Figure 2: Impact of the discretization on the performance of KDS (zoomed).

to the costless KDAS. However, this does not decrease the security as any pre-processing done with public information cannot decrease the privacy. Indeed, even if a given pre-processing decreases the EER, the attacker is likely to perform such pre-processing if it has not be done by the user.

3.3 Blocking protection

In order to continue to increase the EER value, events can be delayed depending on the previous event. The first blocking KDAS ensures that there is at least N frames between each pressure events (block_delay), the second (block_rdelay) delays them such as the ith pressure event's delayed timestamp (t'_i) is computed from the original timestamp t_i as follows: $t'_i = max(t'_{i-1}, t_i) + U(0,N)$..

As shown in Figure 5, both blocking KDAS increase the EER faster than non-blocking KDAS. However, as shown in Figure 6, their latency also quickly explodes. Thus, in order to compare fairly the KDAS between them, Figures 7 and 8 gives the EER in function of the mean and maximal latency.

As shown in Figure 7, block_rdelay is in mean

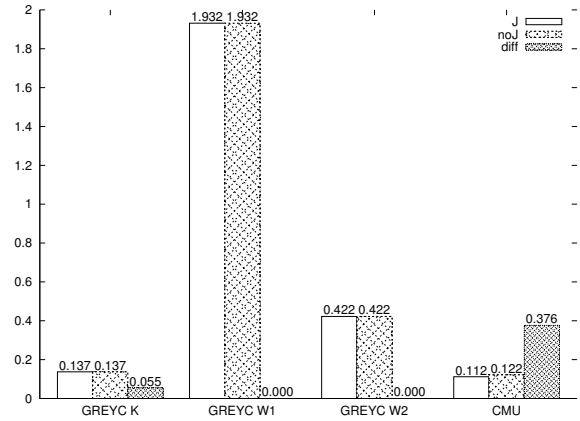

Figure 3: Maximal absolute *gains* on the EER using 1,001 different discretizations with (J) or without jitter (noJ), and maximal difference between EER with and without jitter (diff). EER values are expressed in %.

slightly better than its non-blocking equivalent rdelay. block_delay is by far better than delay, but still worst than rdelay. As for delay, block_rdelay might improve the EER compared to the costless KDAS for $N \leq 5$.

However, as shown in Figure 8, when considering the maximal latency, non-blocking KDAS out-perform by far blocking KDAS. block_rdelay only starts to be better than delay when the maximal latency exceed near 24 frames (0,4 seconds), that is an high latency, and never come close to rdelay. block_delay only becomes better than the costless KDAS when the maximal latency exceed near 20 frames (0,333 seconds).

Moreover, when users type too fast (or N too high), blocking KDAS latency adds up at each key pressed. When this happens, t'_i will only be computed from t'_{i-1}, i.e. every users will have the same way of typing, but at the cost of a non-ergonomic and unacceptable latency. Adapting N to match the user typing speed would enable browser fingerprinting attacks, as it would enable the attacker to discriminate users in function of their configuration, i.e. the N parameter. This suggests that blocking KDAS should be avoided in favor of non-blocking KDAS.

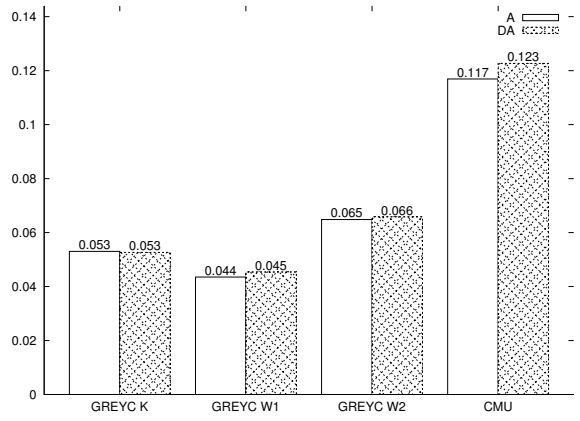

Figure 4: Minimal absolute *loss* on the EER using 1,001 different discretizations with automatic release (A) and delay then automatic release (DA).

4 Proof of concept implementation

We developed *Keystroke Anonymization*, a Firefox WebExtension, that implements the previously cited KDAS. The WebExtension was used during the writing of this paper on Overleaf (method: rdelay, N: 15). Users can enable/disable the protection using the Ctrl+K shortcut, and can enable/disable generation of events using the Ctrl+G shortcut.

A demonstration is also integrated to the WebExtension enabling users to test usability and the protection of different configurations (see Figure 9).

4.1 Implementation issues

The manifest is a JSON configuration file used by WebExtensions. In order to make active the WebExtension on all pages, `content_script`'s `matches` field is set to <all_url>.

The WebExtension listens on each Keyboard events in order to delay them. One important point is that the WebExtension listeners must be called *before* any other, or else the attacker will be able to block call to the WebExtension listeners, i.e. to prevent events from being delayed by the WebExtension.

For that, `content_script`'s `run_at` field must be set to `document_start`, in order to the WebExtension script to be executed before the page scripts, thus allowing it to register listeners before any else. Indeed, listeners are called in the order of their registration.

Moreover, listeners must be added on `document`, with the third parameter of `addEventListener()`, `capture`, set to true. Indeed, event propagation has two phases in JavaScript, capture and bubble. In the capture phase, events are propagated from the root element, `document`, to the target element, e.g. an input. Then during the bubble phase, events are propagated from the target element to the root element. Thus, in order to be the first to capture the event, the WebExtension must capture it during the capture phase, on the root element. The page must be reloaded upon WebExtension installation or activation, in order to ensure to be the first to register listener on already opened pages.

Only the keydown and keyup events are listened to. If the event has be been delayed, its immediate propagation is stopped. If the event is a keydown, the

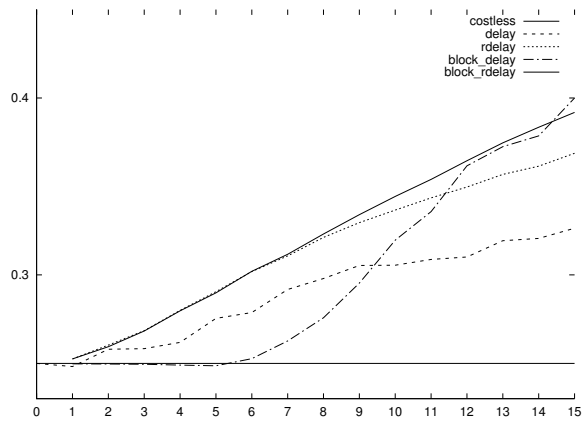

Figure 5: Minimal EER with 5 KDAS in function of their parameter N.

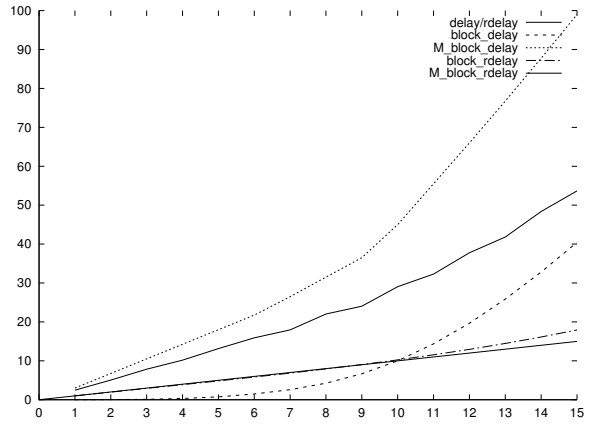

Figure 6: Mean, and maximum (prefixed with M_), of the expected latency for 5 KDAS in function of their parameter N.

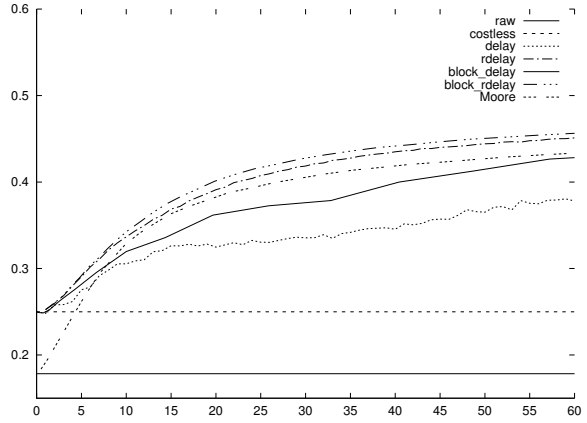

Figure 7: EER in function of mean latency.

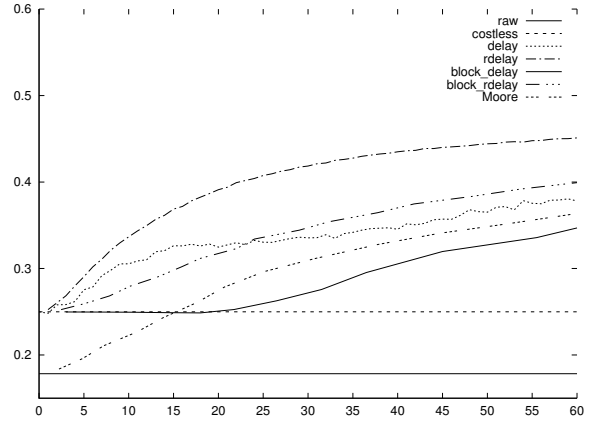

Figure 8: EER in function of maximal latency.

event is captured, i.e. added to an array. As previously stated, delaying event must be done without active wait. Thus, requiring to stop the event propagation with `event.stopImmediatePropagation()`, and to latter re-inject it with `event.target.dispatchEvent(event)`.

The function `window.requestAnimationFrame()` is used to call an handler in order to process captured events before each frame. The frame in which each event will have to be re-injected in then computed depending on the KDAS method, and the parameter N.

However, re-injected event will loose their *trusted* status as it no longer originate from user action. This means that the event will trigger listeners but will not trigger the target default behavior, e.g. add a character on an input. This default behavior has thus to be simulated. Keyboard events that are not a character (`event.key.length != 1`), or when the ctrl key is pressed (`event.ctrlKey`) will not be delayed.

For inputs and text area, this requires to delete the current selection (between

`elem.selectionStart`, `elem.selectionEnd`), insert the character between, set the cursor position (`elem.setSelectionRange(start+1, start+1)`), generate an input event, and add a listener to trigger a change event when the element loses focus. As `elem.selectionStart` and `elem.selectionEnd` are not defined for all types of inputs (e.g. email), the type of the input (`elem.type`), has to be changed to `text` while accessing and modifying theses properties.

div elements can also be used to type text thanks to the `contentEditable=true` attribute. This is used, e.g. by the webmail GMail to write e-mail. For contentEditable elements, current selection must be deleted `window.getSelection().deleteFromDocument()`. The element and position in which insert the character is givent by `selection.focusNode` and `selection.focusOffset`. If the element is a div, its content must be cleared (`div.removeChild(div.fistChild)`), and a new div containing a TextNode must be appended to the fist div. If the element is a TextNode, or once the TextNode created, its content is modified through `textNode.textContent`.

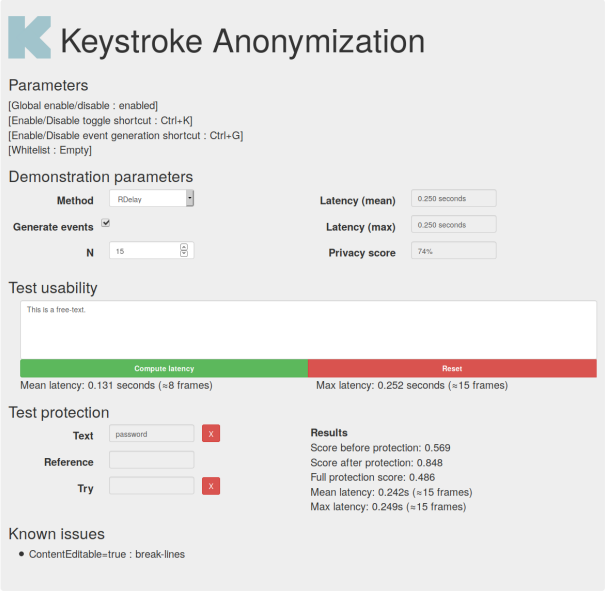

Figure 9: Screenshot of the WebExtension (debug mode).

Before creating an input event, the cursor has to be updated in the following way:

```
let range = document.createRange();
range.setStart(textNode, start+1);
range.setEnd(textNode, start+1);
range.collapse(false);
selection.addRange(range).
```

Unfortunately, the creation of new lines ignore the position of the cursor if the mouse or the arrows key has not been used since the last delayed event. Events s.a. keypress, input, change, could also not be generated when simulating the default behavior on events, to increase the privacy protection by making it more difficult to an attacker to deduce the event timestamp, however, this might impact the functionality of some websites.

4.2 Comparison with KeyboardPrivacy

As shown in Figure 7, KeyboardPrivacy is, in average, slightly less efficient than rdelay when the latency exceed near 7 frames (117ms), and is worst than any other non-costless KDAS before when the latency is near under 7 frames. It is even less efficient than a costless KDAS when the latency is near under 4 frames (67ms).

As shown in Figure 8, KeyboardPrivacy is, when considering the maximal latency, worst than any other KDAS at the exception of block_delay. It is even less efficient than the costless KDAS when the maximal latency is near under 15 frames (0,25 seconds). The construction of this KDAS extension seems to be ad hoc, and could be improved using the conclusion of this study:

- use passive waits instead of active waits ;
- automatically generate release events ;
- delays pressure events to the next frame ;
- use non-blocking KDAS (rdelay) to limit the latency ;
- use fixed parameters for all users to prevent fingerprinting attacks.

It also suffers from several security vulnerabilities. Indeed, the events are captured during the bubble phase, instead of the capture phase. Moreover, the script is, by default, executed after the page has been loaded. This WebExtension also does not support ContentEditable fields.

5 Conclusion and perspectives

This work constitutes a preliminary study on the Keystroke Dynamics Anonymization Scheme. Performances of presented KDAS has been demonstrated using 3 state of the art fixed-text keystroke dynamics datasets. However performances and latency may vary depending on the written text, and the user. KDAS introduce a trade-off between performances (security) and latency (usability). The latency has been evaluated in term of duration, and should be evaluated in terms of usability.

Other KDS could be tested, for authentication, but also, e.g. for soft-biometrics. Attacker model could also be modified in order to include the knowledge of non-protected users references. Other KDAS are also possible, e.g. using non-regular discretization, using non-uniform random laws, or by merging KDAS (e.g. merging delay and rdelay). An hardware implementation of such KDAS, could be also imagined, e.g. in the form of a programmable USB to USB device between the keyboard and the computer. Presented KDAS techniques could be applied to mouse events.

REFERENCES

Acar, G., Juarez, M., Nikiforakis, N., Diaz, C., Gürses, S., Piessens, F., and Preneel, B. (2013). Fpdetective: dusting the web for fingerprinters. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 1129–1140. ACM.

Boda, K., Földes, Á., Gulyás, G., and Imre, S. (2012). User tracking on the web via cross-browser fingerprinting. *Information Security Technology for Applications*, pages 31–46.

Cao, S. Y. and Wijmans, E. (2017). Browser fingerprinting via os and hardware level features. *Network & Distributed System Security Symposium, NDSS*, 17.

Eckersley, P. (2010). How unique is your web browser? In *International Symposium on Privacy Enhancing Technologies Symposium*, pages 1–18. Springer.

Epp, C. (2010). Identifying emotional states through keystroke dynamics. Master's thesis, University of Saskatchewan, Saskatoon, CANADA.

Gaines, R., Lisowski, W., Press, S., and Shapiro, N. (1980). Authentication by keystroke timing: some preliminary results. Technical Report R-2567-NSF, Rand Corporation.

Giot, R., Abed, M. E., and Rosenberger, C. (2012). Web-based benchmark for keystroke dynamics biometric systems: a statistical analysis. In *Intelligent Information Hiding and Multimedia Signal Processing (IIH-MSP), 2012 Eighth International Conference on*, pages 11–15. IEEE.

Giot, R., El-Abed, M., Hemery, B., and Rosenberger, C. (2011). Unconstrained keystroke dynamics authentication with shared secret. *Computers & Security*, 30(6-7):427–445.

Giot, R., El-Abed, M., and Rosenberger, C. (2009). Greyc keystroke: a benchmark for keystroke dynamics biometric systems. In *IEEE International Conference on Biometrics: Theory, Applications and Systems (BTAS 2009)*, pages 1–6.

Giot, R. and Rosenberger, C. (2012). A new soft biometric approach for keystroke dynamics based on gender recognition. *International Journal of Information Technology and Management (IJITM). Special Issue on : "Advances and Trends in Biometrics by Dr Lidong Wang*, 11(1/2):35–49.

Hocquet, S., Ramel, J.-Y., and Cardot, H. (2007). User classification for keystroke dynamics authentication. In *The Sixth International Conference on Biometrics (ICB2007)*, pages 531–539.

Jorgensen, Z. and Yu, T. (2011). On mouse dynamics as a behavioral biometric for authentication. In *Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security*, pages 476–482. ACM.

Killourhy, K. and Maxion, R. (2008). The effect of clock resolution on keystroke dynamics. In *Proceedings of the 11th international symposium on Recent Advances in Intrusion Detection*, pages 331–350. Springer.

Killourhy, K. S. and Maxion, R. A. (2009). Comparing anomaly detectors for keystroke dynamics. In *Proc. of the 39th Ann. Int. Conf. on Dependable Systems and Networks*, pages 125–134.

Kim, J., Kim, H., and Kang, P. (2018). Keystroke dynamics-based user authentication using freely typed text based on user-adaptive feature extraction and novelty detection. *Applied Soft Computing*, 62:1077–1087.

Laperdrix, P., Rudametkin, W., and Baudry, B. (2016). Beauty and the beast: Diverting modern web browsers to build unique browser fingerprints. *Security and Privacy (SP)*, pages 878–894.

Lee, H. and Cho, S. (2007). Retraining a keystroke dynamics-based authenticator with impostor patterns. *Computers & Security*, 26(4):300–310.

Monaco, V. (2018). Public keystroke dynamics datasets.

Monrose, F. and Rubin, A. (2000). Keystroke dynamics as a biometric for authentication. *Future Generation Computer Syststems*, 16(4):351–359.

Moore, P. and Thorsheim, P. (2016). Keyboard privacy plugin.

Nikiforakis, N., Joosen, W., and Livshits, B. (2015). Privaricator: Deceiving fingerprinters with little white lies. *Proceedings of the 24th International Conference on World Wide Web*, pages 820–830.

Revett, K., de Magalhaes, S., and Santos, H. (2007a). On the use of rough sets for user authentication via keystroke dynamics. In *EPIA Workshops*, pages 145–159.

Revett, K., Gorunescu, F., Gorunescu, M., Ene, M., Tenreiro, S. d. M., and Santos, H. M. D. (2007b). A machine learning approach to keystroke dynamics based user authentication. *International Journal of Electronic Security and Digital Forensics*, 1:55–70.

Shen, C., Cai, Z., Guan, X., Du, Y., and Maxion, R. A. (2013). User authentication through mouse dynamics. *IEEE Transactions on Information Forensics and Security*, 8(1):16–30.

Spillane, R. (1975). Keyboard apparatus for personal identification. IBM Technical Disclosure Bulletin.

Umphress, D. and Williams, G. (1985). Identity verification through keyboard characteristics. *Internat. J. Man Machine Studies*, 23:263–273.

Weinberg, Z., Chen, E. Y., Jayaraman, P. R., and Jackson, C. (2011). I still know what you visited last summer: Leaking browsing history via user interaction and side channel attacks. *Security and Privacy (SP)*.