Schéma d'Anonymisation de Dynamique de Frappe au Clavier

Denis Migdal, Christophe Rosenberger

Schéma d'Anonymisation de Dynamique de Frappe au Clavier

Denis Migdal, Christophe Rosenberger
Normandie Univ, UNICAEN, ENSICAEN, CNRS, GREYC, 14000 Caen, France
{denis.migdal,christophe.rosenberger}@ensicaen.fr

11 août 2019

Résumé

La dynamique de frappe au clavier permet l'authentification et l'identification d'utilisateurs en analysant leur façon de taper au clavier, e.g. lorsqu'ils naviguent sur Internet. La majorité des études de l'état de l'art visent à améliorer les performances des systèmes de dynamique de frappe au clavier. Dans cet article, nous adressons le problème d'empêcher la capture de la dynamique de frappe afin de protéger la vie privée des utilisateurs. L'authentification/identification, ainsi que le profilage, peuvent être vu comme des attaques que nous limitons dans cette contribution. Des résultats expérimentaux, obtenus sur des bases de données significatives, montrent l'intérêt de la des méthodes proposées.

1 Introduction

L'empreinte du navigateur (*Browser Fingerprinting*) vise à traquer les utilisateurs via leur navigateur grâce aux données discriminantes qu'un service peut collecter. Cela est usuellement fait pour "personnaliser des services" correspondant à des profils-type utilisateurs, e.g. suggérer des contenus en fonction de l'historique Internet de l'utilisateur. L'objectif de l'empreinte du navigateur n'est pas d'identifier les utilisateurs avec assurance, mais de le classer dans une catégorie, e.g. en identifiant un ensemble de sessions appartenant à un même utilisateur ou type d'utilisateur.

Les sites Panopticlick [4], IAmUnique [16], et UniqueMachine [3] permettent le calcul d'empreintes de navigateurs à partir de données collectées sur un site Internet, généralement grâce au réseau et une API JavaScript, afin de déterminé l'unicité de l'empreinte parmi l'ensemble des empreintes connues par le site. Plus l'empreinte du navigateur est unique, plus le service est capable de discriminer l'utilisateur. L'information utilisée pour l'empreinte du navigateur peut être lié, e.g. au matériel (e.g. GPU [3], écran), au système d'exploitation, au navigateur, à sa configuration, aux polices installées [4, 16], à l'historique Internet [27], ou aux domaines blacklistés [2]. De telles identifications ne sont souvent pas consenties par l'utilisateur, et posent une menace à sa vie privée, conduisant ainsi des chercheurs et développeurs à étudier cette problématique et de proposer des solutions pour protéger la vie privée des utilisateurs [16, 21, 20, 4, 1].

La capture de données biométriques peut aussi être ajoutée à l'empreinte du navigateur, e.g. en utilisant la souris [12, 24] et/ou la dynamique de frappe au clavier [22, 8, 15].

De telles modalités permettent aussi de déduire des informations personnelles à propos de l'utilisateur, e.g. son sexe ou de rapprocher des identités. Dans cet article, nous visons à protéger la vie privée des utilisateurs en anonimysant sa manière de taper au clavier, limitant ainsi les techniques de l'empreinte du navigateur, et empêchant la déduction d'information privée sur les utilisateurs, tout en permettant l'utilisation de cette modalité pour l'authentification d'utilisateurs consentent.

Comme tout système biométrique d'authentification, un système de dynamique de frappe au clavier (KDS) est composés de deux modules : l'enrôlement et la vérification. Chaque utilisateur doit s'enrôler dans le KDS afin de calculer ses templates de références à partir de plusieurs échantillons (*i.e.*, plusieurs saisies d'un mot de passe) acquis durant l'étape d'enrôlement. Pour chaque

saisie, une suite de temps est capturé (temps de pression et de relâchement de chaque touche) à partir desquelles des caractéristiques sont extraites (latences et durées) puis utilisées pour apprendre le modèle qui caractérise l'utilisateur. Lors d'une vérification, l'utilisateur tape son mot de passe. Le système extrait les caractéristiques et les compare au template de référence de l'utilisateur. Si la distance obtenue est en deçà d'un certain seuil, l'utilisateur est accepté, sinon il est rejeté.

Les premiers travaux sur la dynamique de frappe au clavier date des années 80 [6], bien que l'idée d'utiliser un clavier pour identifier des individus a été présenté en 1975 [25]. Dans les rapports préliminaires de Gaines *et al.* [6], sept secrétaires tapent plusieurs paragraphes, les chercheurs montrent alors qu'il est possible de différencier les utilisateurs avec leur façon de taper. Depuis, plusieurs études ont été conduites, permettant de réduire la quantité d'information requise pour construire une référence biométrique, tout en améliorant les performances [26, 19, 23, 17, 8].

Cependant, à la connaissance des auteurs, aucune étude n'a tenté de réduire les performances des KDS dans l'objectif de protéger la vie privée des utilisateurs contre des authentification, identification ou profilage non-désirés. En effet, de nombreux articles [10, 5] ont montré que certaines caractéristiques biométriques (émotion, sexe, âge ...) peuvent être extraites de la manière de taper au clavier. Les services Internet peuvent ainsi profiler leurs utilisateurs via un simple script JavaScript embarqué dans leurs pages web.

La contribution principale de cet article est de proposer plusieurs solutions simple afin que les internautes puissent décider si leur manière de taper au clavier doit être utilisé, ou non, pour un site Internet donné. L'utilisation de la manière de taper au clavier peut être utile afin d'améliorer la sécurité des authentifications tout en évitant les mots de passes trop complexes (e.g. pour un contrôle d'accès à un compte bancaire). Pour d'autres services, comme un réseau social, l'utilisateur est susceptible de désactiver la capture de ses données biométriques. Les méthodes proposées ont été implémentées dans une preuve de concept sous forme d'une WebExtension. Dans cette WebExtension, l'utilisateur peut facilement décider quels services peuvent utiliser ou non sa manière de taper au clavier (obligation RGPD).

L'article est organisé comme suit. La section 2 fournis quelques pré-requis sur la dynamique de frappe au clavier. Nous décrivons une attaque possible et les contremesures existantes dans la littérature. Nous proposons de nouveaux modèles de protection dans la section 3. Leur efficacité est illustré à travers des résultats expérimentaux sur des bases de données significatives. Une WebExtension implémentant ces protections est présenté dans la section 4 et est comparé avec la seule solution existante. La section 5 conclue et donne les perspective de cette étude.

2 Pré-requis

Nous présentons dans cette section quelques pré-requis sur la dynamique de frappe au clavier.

2.1 Système de dynamique de frappe au clavier

Comme le nombre d'échantillons collectés à l'étape d'enrôlement est généralement faible, certains systèmes de dynamique de frappe au clavier sont basés sur une distance. Dans le cadre de cete article, nous supposons que l'attaquant utilise la distance de Hocquet [11] : Nous souhaitons calculer un score entre deux templates K_A et K_B. Nous supposons que le template K_A est associé à μ and

Nom	Texte	# d'utilisateurs (45)	Résolution d'horloge	EER	Source
GREYC K	greyc laboratory	104	10.0144ms	14.75%	[9]
GREYC W1	laboratoire greyc	62	1ms	14.40%	[7]
GREYC W2	sésame	46	1ms	25.39%	[7]
CMU	.tie5Roanl	51	0.2ms	19.38%	[14]

TABLE 1 – Description des bases de données utilisées.

σ la moyenne et l'écart-type des échantillons (note : 0/0 assumé valoir 0).

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

Dans le cadre de cet article, les templates sont composés, pour chaque touche pressées, du temps entre deux pressions consécutives, et de la durée de pression d'une touche. Les 10 premiers échantillons de chaque utilisateurs sont utilisés pour le calcul du template de référence.

2.2 Bases de données de dynamique de frappe au clavier

Il existe plusieurs bases de données de dynamique de frappe au clavier [18]. Nous avons décidé dans ce travail de nous restreindre aux bases de données sur texte fixe (i.e. où l'utilisateur tape le même mot de passe). Les bases de données ont été nettoyées pour retirer les données incohérentes , e.g. les entrées dans lesquelles l'utilisateun n'a pas tapé le texte demandé. Cela correspond à 13% des entrées de GREYC W, et moins de 3 entrées pour les autres bases de données.

Afin d'obtenir des ensembles comparables, seuls les 45 premières entrées de chaque utilisateurs sont conservées. Les utilisateurs avec moins de 45 entrées et les bases de données avec moins de 45 utilisateurs sont retirés. À partir des bases de données sur texte fixe existantes, seules 3 correspondait à nos critères. De ces 3 bases de données, nous construisons 4 bases de données (l'une ayant deux textes fixes, 2 bases de données sont donc créées à partir de celle-là). La table 1 donne les bases de données utilisées dans ce travail.

2.3 Modèle d'attaque

L'attaquant est capable d'exécuter du code JavaScript arbitraire sur le navigateur de l'utilisateur afin de l'identifier, utilisant uniquement les événements claviers horodatés. Nous assumons dans cet article que le texte tapé est fixe, s.a. un login, e-mail, ou mot de passe. L'attaque est aussi possible sur du texte libre.

L'attaquant est capable de mesurer les temps des événements claviers qu'il reçoit grâche à la fonction JavaScript `Date.now()`. Ainsi, modifier l'horodatage est événements n'aura aucun effet sachant que l'attaquant pourra les mesurer lui-même. En revanche, les événements peuvent être retardés, i.e. en attendant avant d'envoyer l'évènement. Comme la boucle événementielle de JavaScript est mono-threadée, toute attente active est problématique et sera aisément détectée par l'attaquant utilisant `setInterval()`. Cela requiert ainsi de détruire l'évènement à retarder puis de le recréer après une attente passive avec `setTimeout()`.

L'attaquant a un *a priori* sur l'identité de l'utilisateur, est capable d'utiliser n'importe quel système de dynamique de frappe au clavier, et peut effectuer n'importe quel pré-traitement afin d'identifier ou de profiler l'utilisateur. La manière dont la dynamique de frappe est protégée, ainsi que les éventuels paramètres d'un tel schéma d'anonymisation est aussi assumé être connu par l'attaquant. Ainsi, les paramètres doivent être fixés pour chaque utilisateurs afin d'empêcher l'attaquant de les utiliser pour discriminer les utilisateurs via des techniques d'empreintes du navigateur [4].

2.4 Contre-mesures dans la littérature

Afin d'éviter l'attaque décrite précédemment, l'internaute peut désactiver l'exécution de JavaScript dans son navigateur Internet. Cela impacte lourdement sa navigation.

La principale idée de protection de l'utilisateur contre l'identification/profilage via les données de dynamique de frappe au clavier est de déformer les données envoyées. Très peu d'études ont été réalisées dans l'état de l'art pour éviter la capture de la dynamique de frappe au clavier sur Internet. À notre connaissance, il n'existe qu'un seul travail évitant la capture de la dynamique de frappe au clavier : KeyboardPrivacy[20], une extension Google Chrome. L'horodatage de chaque évènement est calculé de la sorte :

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

Où b est une valeur aléatoire suivant une loi Uniforme entre 0 et a (cette valeur est définie par l'utilisateur).

2.5 Performance de l'attaque

La capacité d'un attaquant à authentifier un utilisateur sera quantifié comme l'estimation maximale de l'*Equal*

Error Rate (EER) qui correspond à la configuration du système biométrique où le FAR égale le FRR. Le *False Acceptance Rate* (FAR) décris le ratio de données imposteurs acceptés, le *False Rejection Rate* (FRR) décris le ratio de données légitimes rejetées.

La performance d'un schéma d'anonymisation de dynamique de frappe au clavier (KDAS) sera ainsi quantifié par le minimum de l'estimation maximale de l'EER pour chaque KDS et pré-traitements possibles. Pour un KDS et pré-traitement possible, et si le KDAS n'est pas déterministe, le KDAS est testé 20 fois, et la moyenne de l'estimation maximale de l'EER pour chaque test est utilisé. Si la base de données n'est pas indiquée, le nombre donné est la moyenne des 4 bases de données utilisées dans cette étude.

2.6 Pré-traitement de l'attaquant

L'horodatage d'un événement donné dépend de la résolution et du *jitter* de l'horloge utilisée pour le mesurer. La *résolution* est le temps moyen entre deux tics d'horloges théoriques. Le *jitter* est la différence entre le temps du tic d'horloge théorique et le temps réel. Cela signifie qu'un événement survenant à un temps t aura un horodatage de $\lfloor t/r \rfloor * r + j$, où r est la résolution d'horloge, et j un bruit aléatoire (le jitter). Les études existantes ont montrés que les résolutions d'horloges influences les performances des KDS [13], et que la discrétisation peut en améliorer les performances [8].

Dans la suite, nous illustrons l'impact de la discrétization. Les valeurs d'horodatage ont été discrétisées avec 1 001 résolutions différentes (de 0 à 1 par pas de 1/1,000). Comme montré dans la Figure 3, l'attaquant peut, de la sorte, espérer un léger ($J \simeq 0.02$ pour GREYC W1) ou négligeable ($J < 0.005$). La Figure 1 montre que la discrétisation peut à la fois améliorer et empirer l'EER en fonction de la résolution. La Figure 2 zoom sur la zone où l'EER décroît.

Le jitter peut être retiré en utilisant la formule suivante : $t' = \lceil t/r \rfloor * r$. Comme le montre la Figure 3, le jitter a une influence négligeable sur l'EER (diff < 0.004, et noJ $\simeq$ J), et ne nécessite ainsi pas d'être retiré.

Dans la section suivante, nous proposons de nouvelles solutions pour protéger les internautes contre leur identification/profilage via leur manière de taper au clavier.

3 Schémas de protection proposés

Nous proposons différentes solutions pour anonymiser la dynamique de frappe au clavier des utilisateurs. Leur objectif est de permettre aux services internet d'utiliser la dynamique de frappe au clavier lorsque l'utilisateur consent (pour des applications de sécurité), mais de fournir des données altérées sinon (pour protéger la vie privée).

3.1 Protection sans coûts

Les évènements de relâchement clavier peuvent être automatiquement générés à une durée constante après l'évènement de pression e.g. 2ms (A). Comme montré dans la Figure 4, une telle strategie augmente de manière significative l'EER ($0.044 \leq A \leq 0.117$).

L'écran de l'utilisateur dessine typiquement une frame toutes les 1/60 secondes. Ainsi, dans le cadre d'une utilisation ordinaire, le temps d'apparition d'un évènement entre deux frames consécutives ne fait aucune différence pour un utilisateur, i.e. tout retardement d'un évènement pour coller au temps de la prochaine frame est *de facto* impossible à percevoir pour un utilisateur, et est ainsi considéré sans coûts. Une telle opération peut être trivialement effectuée grâce à `Window.requestAnimationFrame()`.

Comme montré dans la Figure 4, la génération automatique des évènements de relâchement après retardement de l'évènement de pression à la prochaine frame (DA), augmente aussi l'EER. Une telle stratégie est aussi intéressante car elle supprime des informations qui pourrait être exploités par d'autres KDS.

Dans la suite, pour les KDAS avec coûts, les événements de pressions seront au préalables retardés à la prochaine frame, et les événements de relâchements automatiquement générés ensuite.

3.2 Protections non-bloquantes

Afin d'augmenter encore l'EER, certains événements doivent être retardés au delà de la prochaine frame. Ce délai peut ainsi être perçu par les utilisateurs et constitue donc un coûts en terme d'utilisabilité du KDAS. Ce coût, que nous appelons latence, est calculé comme le nombre

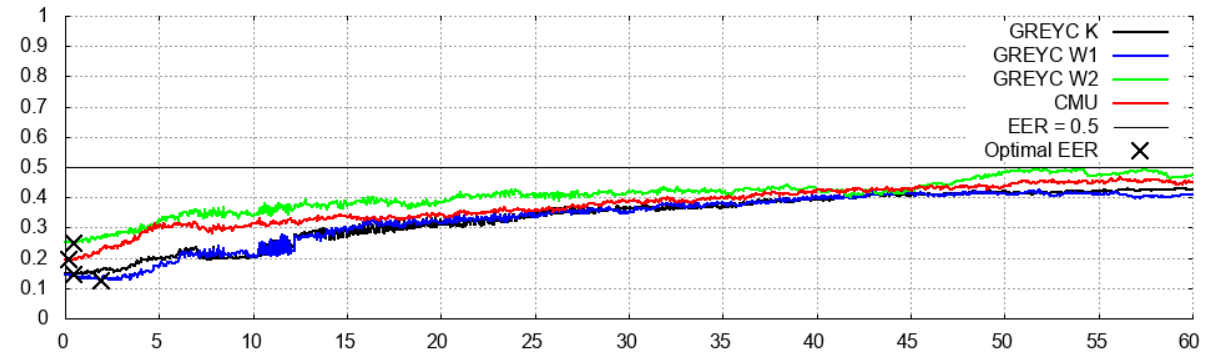

FIGURE 1 – Impact de la discrétisation sur les performances du KDS.

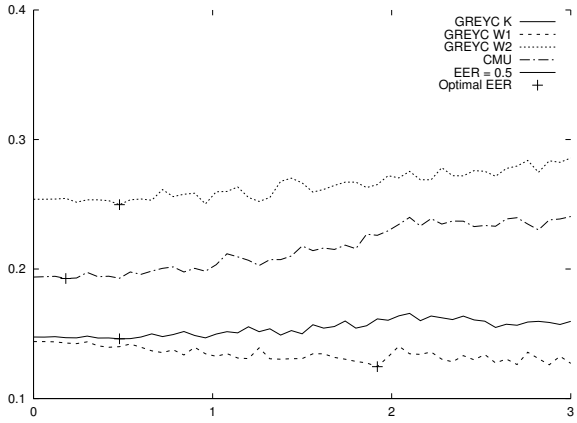

FIGURE 2 – Impact de la discrétisation sur les performances du KDS (zoomé).

maximal de frames sautées durant l'écriture d'un texte donné.

Les KDAS non-bloquant retardent les événements indépendamment des précédents, avec la seule contrainte de conserver l'ordre des événements de pression. Leur paramètre N est le nombre de frame qui peuvent être sautées, et ainsi *de facto* leur latence.

Deux KDAS non-bloqunat sont étudiés. Dans le premier, les événements sont discrétisés avec une résolution de $(N+1)/60$ (delay), et dans la seconde, les événements sont retardés de n frames avec n un bruit discret uniforme $n \sim U(0,N)$ (rdelay). Ces deux KDAS ont été testés avec 15 configurations, $N \in [\![0,14]\!]$ pour delay, et $N \in [\![1,15]\!]$ pour rdelay. Comme montré dans la Figure 5, les deux donnent une protection significative comparativement aux KDAS sans coûts. Cependant, pour la même latence, rdelay semble toujours meilleurs que delay.

De surcroît, delay améliore l'EER for N=1, comparativement au KDAS sans coûts. Cependant, cela de diminue pas la sécurité comme tout pré-traitement effectué avec des données publiques ne peut diminuer la sécurité. En effet, même si un pré-traitement donné diminue l'EER, l'at-

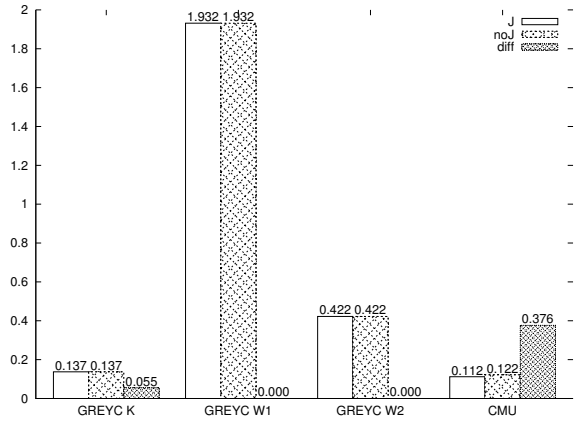

FIGURE 3 – *Gains* absolu maximal d'EER en utilisant 1 001 résolutions différentes avec (J) ou sans jitter (noJ), et différence maximale entre l'EER avec et sans jitter (diff). Les valeurs d'EER sont exprimées en %.

taquant est susceptible d'effectuer un tel pré-traitement s'il n'a pas été effectué par l'utilisateur.

3.3 Protection bloquante

Afin de continuer d'augmenter l'EER, les événements peuvent être retardés en fonction des événements précédents. Le premier KDAS bloquant s'assure qu'il y ai au moins N frames entre chaque évènements de pression (block_delay), le second (block_rdelay) les retarde de sorte à ce que le i^{ieme} l'horodatage de l'événement de pression pressure (t'_i) est calculé à partir de l'horodatage original t_i comme suit : $t'_i = max(t'_{i-1}, t_i) + U(0,N)$..

Comme montré dans la Figure 5, les deux KDAS bloquant augmente l'EER plus rapidement que les KDAS non-bloquant. Cependant, comme montré par la Figure 6, leur latence explose rapidement. Ainsi, à fin de comparer équitablement les KDAS, les Figures 7 and 8 donnent l'EER en fonction de la latence moyenne et maximale.

Comme montré dans la Figure 7, block_rdelay est en moyenne légèrement meilleurs que sont équivalant non-bloquant. block_delay est de loin meilleurs que delay, mais est encore pire que rdelay. Tout comme delay, block_rdelay risque d'améliorer l'EER comparativement au KDAS sans coûts pour $N \leq 5$.

Cependant, comme montré dans la Figure 8, lorsque la latence maximale est étudiée, les KDAS non-bloquant

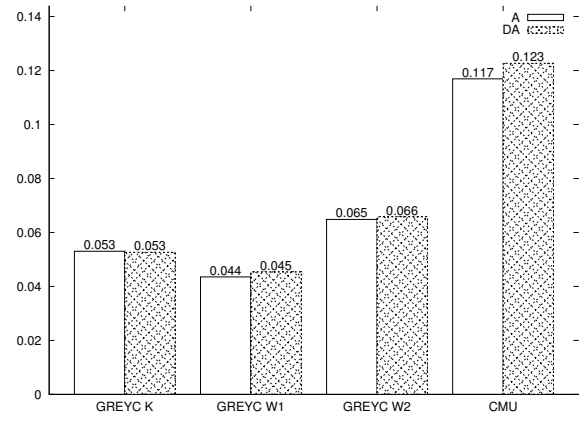

FIGURE 4 – *Perte* absolue minimale d'EER en utilisant 1 001 résolutions différentes avec relâchement automatique (A) et délai puis relâchement automatique (DA).

surclassent de loin les KDAS bloquant. block_rdelay commence à dépasser delay uniquement quand la latence maximale dépasse les 24 frames (0,4 seconds), ce qui est une latence élevée, et ne se rapproche jamais de rdelay. block_delay commence à dépasser le KDAS sans coûts costless KDAS uniquement quand la latence maximale dépasse 20 frames (0,333 seconds).

De plus quand les utilisateurs tapent trop rapidement (ou N trop élevé), la latence des KDAS bloquant se cumule à chaque pression d'une touche. Quand cela arrive, t'_i va seulement dépendre de t'_{i-1}, i.e. chaque utilisateurs aura la même manière de taper au clavier, mais au coût d'une latence non-ergonomique et inacceptable. Adapter le paramètre N pour correspondre à la vitesse de frappe de l'utilisateur permettrait des attaques par des techinques d'empreintes de navigateur, comme cela permettrait à l'attaquant de discriminer les utilisateurs en fonction de leur configuration, i.e. le paramètre N. Cela suggère que les KDAS bloquants devraient être évités en faveur des KDAS non-bloquants.

4 Implémentation d'une preuve de concept

Nous avons développé *Keystroke Anonymization*, une WebExtension Firefox, qui implémente les KDAS précédemment évoqués. La WebExtension a été utilisée durant l'écriture de cet article sur Overleaf (méthode : rdelay, N : 15). L'utilisateur peut (dés)activer la protection en utilisant le raccourcit clavier Ctrl+K et (dés)activer la génération d'événements avec le raccourcit clavier Ctrl+G.

Une démonstration est aussi intégrée à la WebExtension permettant aux utilisateurs de tester l'utilisabilité te la protection de différentes configurations (voir Figure 9).

4.1 Problèmes d'implémentation

Le manifeste est un fichier de configuration JSON configuration utilisé dans les WebExtensions. Afin de rendre active la WebExtension sur toutes les pages, le champ `matches` de `content_script` est mis à <all_url>.

La WebExtension écoute chaque évènements clavier afin de les retarder. Un point important est que l'écouteur

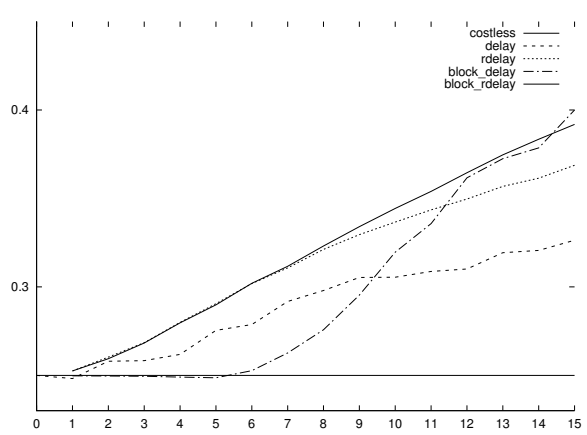

FIGURE 5 – EER minimal de 5 KDAS en fonction de leur paramètre *N*.

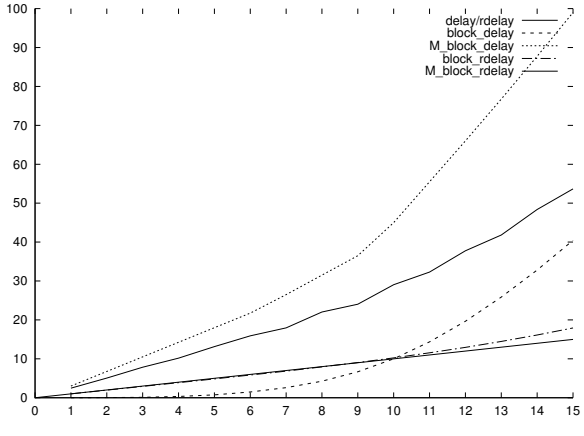

FIGURE 6 – Moyenne et maximum (préfixé par M_), de la latence attendue pour 5 KDAS en fonction de leur paramètre *N*.

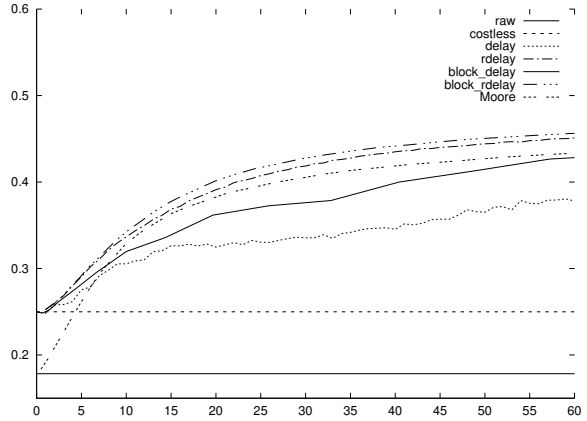

FIGURE 7 – EER en fonction de la latence moyenne.

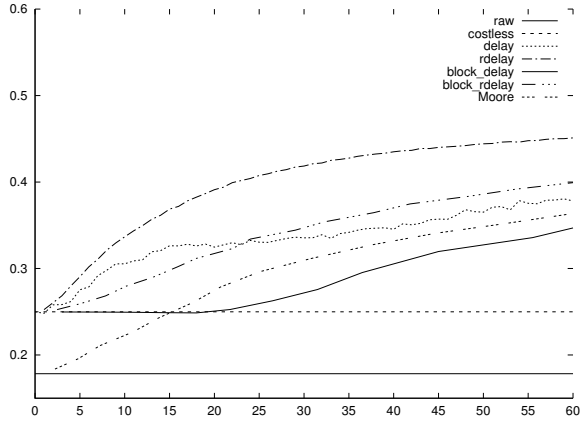

FIGURE 8 – EER em fonction de la latence maximale.

de la WebExtension doit être appelé *avant* tout autre, ou sinon un attaquant sera capable de bloquer l'écouteur de la WebExtentions, i.e. d'empêcher les événements d'être retardés par la WebExtension.

Pour cela, le champ `run_at` de `content_script` doit être mis à `document_start`, pour que le script de la WebExtension soit exécuté avant les scripts de la page, permettant ainsi d'enregistrer l'écouteur avant tout autre. En effet, les écouteurs sont appelés dans l'ordre de leur enregistrement.

De surcroît, l'écouteur doit être ajouté sur `document`, avec le troisième paramètre de `addEventListener()`, `capture`, à *true*. En effet, la propagation d'évènement en JavaScript a deux phases, *capture* et *bubble*. Dans la phase de capture, les événements sont propagés de l'élément racine, `document`, à l'élément cible, e.g. une zone de texte. Puis durant la phase de bubble, les évènements sont propagés de la cible vers l'élément racine. Ainsi afin d'être le premier à capturer l'évènement, la WebExtension doit le capturer durant la phase de capture, sur l'élément ra-

cine. Les pages doivent être rechargée lors de l'installation ou activation de la WebExtension installation or activation, afin de s'assurer d'être le premier à enregistrer un écouteur sur les pages déjà ouvertes.

Seuls les événements *keydown* et *keyup* sont écoutés. Si l'évènement a été retardé, sa propagation immédiate est stoppée. Si l'évènement est une pression, l'événement est capturé, i.e. ajouté à une liste. Comme précédemment déclaré, le retardement d'évènements doit être fait sans attente actives. Nécessitant ainsi d'arrêter la propagation immédiate avec `event.stopImmediatePropagation()`, et d'ensuite ré-injecter l'évènement avec `event.target.dispatchEvent(event)`.

La fonction `window.requestAnimationFrame()` est utilisée pour appeler le gestionnaire afin de traiter les évènements capturés avant chaque frames. La frame dans laquelle chaque évènement devra être ré-injecté est ensuite calculé en fonction de la méthode KDAS et du paramètre N.

Cependant, l'évènement réinjecté perd son status *trusted* comme il n'origine plus d'une action utilisateur.

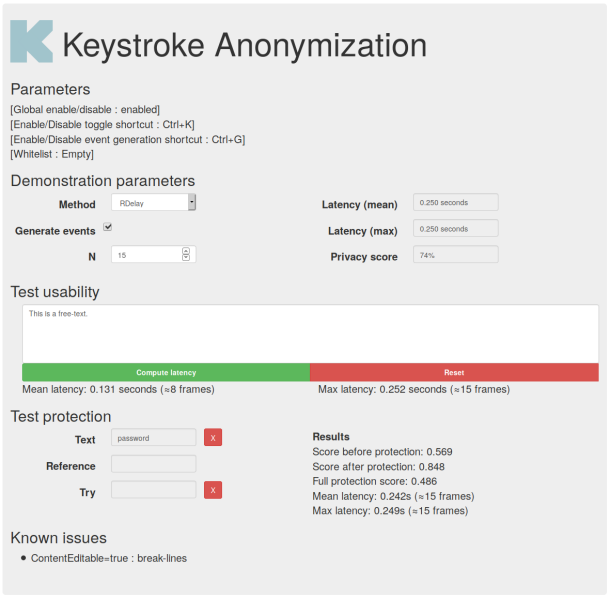

FIGURE 9 – Capture d'écran de la WebExtension (mode debug).

Cela signifie que l'évènement déclenchera les écouteurs, mais n'enclenchera plus le comportement par défaut de la cible, e.g. ajouter un caractère dans une zone de texte. Ce comportement par défaut doit donc être simulé. Les évènements clavier qui ne sont pas un caractère (`event.key.length != 1`), ou quand la touche ctrl est enfoncée (`event.ctrlKey`) ne seront pas retardés.

Pour les zones de textes, cela requiert de supprimer la sélection courante (entre `elem.selectionStart` et `elem.selectionEnd`), d'ajouter le caractère entre, mettre à jour la position du curseur (`elem.setSelectionRange(start+1, start+1)`), générer un évènement *input*, et ajouter un écouter pour déclencher un évènement *change* lorsque l'élément perd son focus. Comme `elem.selectionStart` et `elem.selectionEnd` ne sont pas définis pour tous les types de zones de textes (e.g. email), le type de la zone de texte (`elem.type`), doit être changé à `text` lorsque ces propriétés sont accédées et modifiées.

Les éléments div peuvent aussi être utilisés pour entrer du texte grâce à l'attribut `contentEditable=true`. Cela est utilisé, e.g. par GMail pour écrire des e-mails. Pour les éléments contentEditable, la sélection courrante doit être supprimée `window.getSelection().deleteFromDocument()`. L'élément et la position dans laquelle insérer le caractère est donné par `selection.focusNode` et `selection.focusOffset`. Si l'élément est un div, sont contenu doit être supprimé (`div.removeChild(div.fistChild)`), et une nouvelle div contenant un TextNode doit être ajouté à la première div. Si l'élément est un TextNode, ou nue fois le TextNode créé, son contenu est modifié via `textNode.textContent`.

Avant de créer un évènement input, le curseur doit être mis à jour de la manière suivante :

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

Malheureusement, la création de nouvelles lignes ignore la position du curseur si la souris ou les touches de directions n'ont pas été utilisés depuis le dernier retardement d'évènements. les évènements s.a. keypress, input, change, peuvent aussi ne pas être générés lorsque le com-

portement par défaut est simulé afin d'augmenter la protection, rendant plus difficile pour un attaquant de déduire l'horodatage des évènements. Cependant, cela peut impacter les fonctionnalités de certains sites Internet.

4.2 Comparaison avec KeyboardPrivacy

Comme montré dans la Figure 7, KeyboardPrivacy est, en moyenne, légèrement moins performant que rdelay lorsque la latence excède 7 frames (117ms), et est pire que tout autre KDAS coûteux lorsque la latence est en deçà de 7 frames. Il est même moins efficace qu'un KDAS non-coûteux lorsque la latence est en deçà de 4 frames (67ms).

Comme montré dans la Figure 8, KeyboardPrivacy est, losque considérant la latence maximale, pire que tout autre KDAs à l'exception de block_delay. Il est même pire que le KDAS sans coûts lorsque la latence maximale est en deçà de 15 frames (0,25 seconds). La construction de cette extension KDAS semble être ad hoc, et pourrait être améliorée en utilisant les conclusion de cette étude :

— utiliser une attente passive au lieu d'une attente ;
— générer automatiquement les évènement de relâchement ;
— retarder les évènement de pression à la prochaine framee ;
— utiliser un KDAS non-bloquant (rdelay) pour limiter la latence ;
— utiliser des paramètres fixes pour tous les utilisateurs afin d'empêcher des attaques par empreinte de navigateur.

L'extension souffre aussi de nombreuses vulnérabilités. En effet, les évènements sont captures lors de la phase de bubble au lieu de la phase de capture. De plus, le script, est par défaut, exécuté après que la page ai été chargée. L'extension ne supporte pas aussi les champs ContentEditable.

5 Conclusion et perspectives

Ce travail constitue une étude préliminaire sur les schémas d'anonymisation de la dynamique de frappe. Les performances des KDAS présentés ont été démontrées en utilisant 3 bases de données de dynamique de frappe au clavier sur texte fixe de l'état de l'art. Cependant les performances et latences peuvent varier en fonction du texte écrit et de l'utilisateur. Les KDAS introduisent un compromis entre performances (sécurité) et latence (utilisabilité). La latence a été évaluée en terme de durée, et devrait être évaluée en terme d'utilisabilité.

D'autres KDS pourraient être testés, pour l'authentification, mais aussi, e.g. pour de la biométrie douce. Le modèle de l'attaquant pourrait aussi être modifié pour inclure la connaissance de références utilisateurs non-protégés. D'autres KDAS sont aussi possibles, e.g. en utilisant une discrétization non-régulière, des lois aléatoires non-uniformes, ou en fusionnant des KDAS (e.g. fusionner delay et rdelay). Une implémentation matérielle de tels KDAS pourrait aussi être imaginé, e.g. sous la forme d'un composant programmable USB vers USB se plaçant entre le clavier et l'ordinateur. Les techniques de KDAS présentées pourrait aussi être appliquée aux évènements souris.

Références

[1] Gunes Acar, Marc Juarez, Nick Nikiforakis, Claudia Diaz, Seda Gürses, Frank Piessens, and Bart Preneel. Fpdetective : dusting the web for fingerprinters. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 1129–1140. ACM, 2013.

[2] Károly Boda, Ádám Földes, Gábor Gulyás, and Sándor Imre. User tracking on the web via cross-browser fingerprinting. *Information Security Technology for Applications*, pages 31–46, 2012.

[3] SL Yinzhi Cao and E Wijmans. Browser fingerprinting via os and hardware level features. *Network & Distributed System Security Symposium, NDSS*, 17, 2017.

[4] Peter Eckersley. How unique is your web browser ? In *International Symposium on Privacy Enhancing Technologies Symposium*, pages 1–18. Springer, 2010.

[5] Clayton Epp. Identifying emotional states through keystroke dynamics. Master's thesis, University of Saskatchewan, Saskatoon, CANADA, 2010.

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

[7] R. Giot, M. El Abed, and C. Rosenberger. 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, 2012.

[8] Romain Giot, Mohamad El-Abed, Baptiste Hemery, and Christophe Rosenberger. Unconstrained keystroke dynamics authentication with shared secret. *Computers & Security*, 30(6-7) :427–445, September 2011.

[9] Romain Giot, Mohamad El-Abed, and Christophe Rosenberger. Greyc keystroke : a benchmark for keystroke dynamics biometric systems. In *IEEE International Conference on Biometrics : Theory, Applications and Systems (BTAS 2009)*, pages 1–6, 2009.

[10] Romain Giot and Christophe Rosenberger. 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, 2012.

[11] Sylvain Hocquet, Jean-Yves Ramel, and Hubert Cardot. User classification for keystroke dynamics authentication. In *The Sixth International Conference on Biometrics (ICB2007)*, pages 531–539, 2007.

[12] Zach Jorgensen and Ting Yu. 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, 2011.

[13] K. Killourhy and R. Maxion. 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, 2008.

[14] Kevin S Killourhy and Roy A Maxion. Comparing anomaly detectors for keystroke dynamics. In *Proc. of the 39th Ann. Int. Conf. on Dependable Systems and Networks*, pages 125–134, 2009.

[15] Junhong Kim, Haedong Kim, and Pilsung Kang. Keystroke dynamics-based user authentication using freely typed text based on user-adaptive feature extraction and novelty detection. *Applied Soft Computing*, 62 :1077–1087, 2018.

[16] Pierre Laperdrix, Walter Rudametkin, and Benoit Baudry. Beauty and the beast : Diverting modern web browsers to build unique browser fingerprints. *Security and Privacy (SP)*, pages 878–894, 2016.

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

[18] Vinnie Monaco. Public keystroke dynamics datasets, 2018.

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

[20] Paul Moore and Per Thorsheim. Keyboard privacy plugin, 2016.

[21] Nick Nikiforakis, Wouter Joosen, and Benjamin Livshits. Privaricator : Deceiving fingerprinters with little white lies. *Proceedings of the 24th International Conference on World Wide Web*, pages 820–830, 2015.

[22] K. Revett, S.T. de Magalhaes, and H.M.D. Santos. On the use of rough sets for user authentication via keystroke dynamics. In *EPIA Workshops*, pages 145–159, 2007.

[23] Kenneth Revett, Florin Gorunescu, Marina Gorunescu, Marius Ene, Sergio de Magalhees Tenreiro, and Henrique M. Dinis Santos. A machine learning approach to keystroke dynamics based user authentication. *International Journal of Electronic Security and Digital Forensics*, 1 :55–70, 2007.

[24] Chao Shen, Zhongmin Cai, Xiaohong Guan, Youtian Du, and Roy A Maxion. User authentication through mouse dynamics. *IEEE Transactions on Information Forensics and Security*, 8(1) :16–30, 2013.

[25] RJ Spillane. Keyboard apparatus for personal identification. IBM Technical Disclosure Bulletin, April 1975.

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

[27] Zachary Weinberg, Eric Y Chen, Pavithra Ramesh Jayaraman, and Collin Jackson. I still know what you visited last summer : Leaking browsing history via user interaction and side channel attacks. *Security and Privacy (SP)*, 2011.