Using hash visualization for real-time user-governed password validation

Julian Fietkau
julian.fietkau@unibw.de
Bundeswehr University Munich

Mandy Balthasar
mandy.balthasar@unibw.de
Bundeswehr University Munich

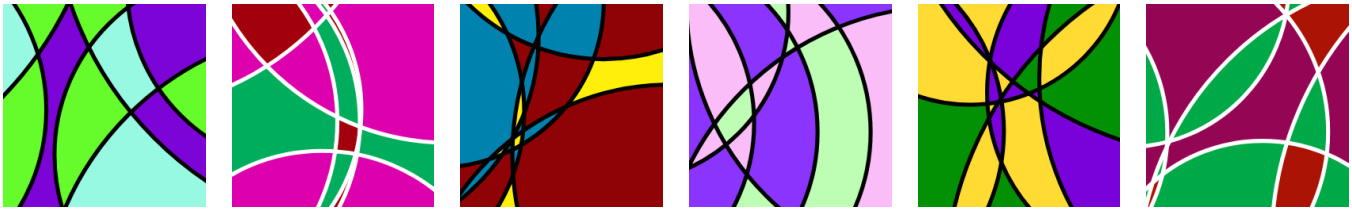

Figure 1: Six example images generated by the MosaicVisualHash algorithm based on random input data.

ABSTRACT

Building upon work by Perrig & Song [21], we propose a novel hash visualization algorithm and examine its usefulness for user-governed password validation in real time.

In contrast to network-based password authentication and the best practices for security which have been developed with that paradigm in mind, we are concerned with use cases that require user-governed password validation in non-networked untrusted contexts, i.e. to allow a user to verify that they have typed their password correctly without ever storing a record of the correct password between sessions (not even a hash). To that end, we showcase a newly designed hash visualization algorithm named MosaicVisualHash and describe how hash visualization algorithms can be used to perform user-governed password validation. We also provide a set of design recommendations for systems where hash visualization for password validation is performed in real time, i.e. as the user is in the process of typing their password.

CCS CONCEPTS

• **Security and privacy** → **Authentication**; • **Human-centered computing** → *Human computer interaction*;

KEYWORDS

hash visualization, image recognition, password masking, usable security, authentication, human-computer interaction

MuC'19 Workshops, Hamburg, Deutschland

https://doi.org/10.18420/muc2019-ws-302-04

1 INTRODUCTION

This work was created as a follow-up to an idea by Perrig & Song [21] in order to introduce the human cognitive ability to perceive and remember structured images into the password validation process.

The work of Perrig & Song [21] is based on the assumption that it is difficult for people to compare or memorize symbols that are strung together without an underlying meaning. This assumption is supported by the results of G. A. Miller [18] as well as Stuart K.Card et al. [4] on the possibilities and limits of man in the processing of information. The aim of Perrig & Song's work was to replace this human processing weakness with a strength. Thus, in selected scenarios, the validation of root keys in public key infrastructures and user authentication were to be facilitated, protecting system access from specific kinds of human errors. The human weakness with respect to character sequences was replaced by a human strength in the processing of structured images. Perrig & Song proposed a prototype based on a requirements analysis. A hash visualization was used for the root key validation. The generated images could be compared by humans more easily than the character strings that were commonly used. The fact that humans can recognize and compare colors, shapes and patterns excellently has already been proven several times before. Work by R. M. Boynton and D. E. Boss [3] as well as R. E. Reynolds et al. [22] or also L. G. Williams [28] at the end of the sixties to the beginning of the seventies underlines the human strength of image recognition. In the user authentication scenario, Perrig & Song replaced the

input of a number in the form of a password or PIN with the recognition of a known image. Perrig & Song pursued the idea of using a primitive hash visualization as a solution. For this, they analyzed the necessary requirements and proposed Andrej Bauer's Random Art algorithm [2] as a prototypical solution (see figure 2). The aim was to show how hash visualization can be used to increase security for root key validation and user authentication. In the Perrig & Song scenario, each user knows a small number of images from a previously defined portfolio. For authentication, the user is shown a selection of images that they can mark if they recognize them from the known portfolio.

Based on the preparatory work done by Perrig & Song [21], this paper focuses on use cases where a user is interested in validating a password that they have entered, even if the application is unable to provide secure automatic validation. One such real-world use case is the use of a password as a deterministic seed for a crypto key in a local software process. Therefore, this paper does not consider the implications of using a password for network-based authentication. If an incorrect password is provided to the application without any opportunity for error-correction, useless data is output without the software being able to notice. In the scenario under consideration, a user types in their password and at the same time needs to be able to ensure that the input is error-free without the typed password being visible on screen. The background of the scenario lies in an untrusted context, in which the storage of a password hash, for example in the browser, can present a security problem. The currently common method of authentication is by comparing the data provided by the user with the data stored by the computer. Instead of the widespread method of the comparison of a previously stored hash value by the machine, the comparison in this work is carried out by humans using a visualized hash value. The authentication as assertion of an identity remains the same as in conventional methods, but the authentication as verification of the assertion is taken over by the user and only the authorization, i.e. the actual granting or denial of rights, is performed by the machine. The insecure storage of a hash value is thus superfluous and the associated security risk can be avoided.

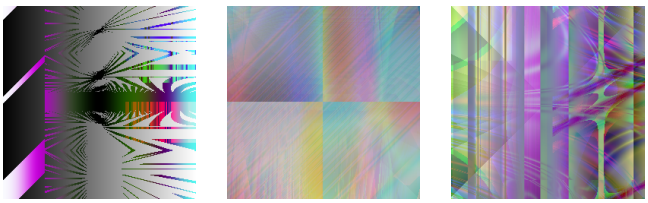

Figure 2: Example outputs of Andrej Bauer's Random Art algorithm [2], used by Perrig & Song [21] for hash visualization. The algorithm was not invented for this purpose, but fits the use case easily since it deterministically generates a structured image from a bit sequence.

In contemporary GUI systems, password input fields typically employ *password masking* as a security measure. When a character is typed into a password field, an asterisk or a circle (or another masking character) is shown instead of the typed character, which remains invisible on screen. The assumption is that other people who can see the screen ("over-the-shoulder attackers") should be prevented from reading the password. One downside to this approach is that the user cannot read the password they just typed either, so the ability to quickly see and correct typing mistakes is vastly reduced.

Even though it is near-ubiquitous, password masking has received criticism by user researchers, typically centering on the inability of users to check at a glance whether the password has been typed correctly [13, 19, 25, 30]. In real-world systems, there has been a shift away from strict password masking in recent years, e.g. with "show password" toggles appearing increasingly often and mobile devices showing the last typed password character in the password field for a brief amount of time. A few existing proposals for better password masking will be shown in section 2.

In addition to password-based authentication schemes, there are situations in which humans are asked to verify or compare hash values, such as when hash-based checksums are provided for file downloads, or when a user logs into a remote server and the client asks them to verify the server fingerprint (seen e.g. in the "Secure Shell" software). In those cases, the hash values are usually provided as a number sequence, most commonly in hexadecimal format, and require tedious and error-prone manual comparison.

We have developed a novel algorithm that visualizes a number sequence (e.g. a hash) as an image consisting of overlapping circle segments using a limited color palette, which provides a visual impression similar to abstract mosaics or stained glass windows. We have then implemented our algorithm into a demo application that uses it to visualize password hashes in real-time as the user types a password[1]. This article describes our algorithm step by step, gives context for our use case, and outlines further measures we have taken to make the hash visualization accessible to humans while safeguarding against various attacks.

The contributions that this article aims to provide consist of (a) the specification of user-governed password validation as a use case for password hash visualization, (b) the MosaicVisualHash algorithm, and (c) our design recommendations for secure and usable implementations of hash

[1] An interactive demo can be tested here:
https://jfietkau.github.io/Mosaic-Visual-Hash/demo-password.html

```
$ ssh example.com
Host key fingerprint is ...
+--[ECDSA 256]---+
|     ....o  E+*o*|
|      .o+...o=.=+|
|       oo+o..*.oo|
|        ooo * +*+|
|       .S. o +*++|
|         .   .. *o|
|               .oo|
|               . o|
|                . |
+---[SHA256]----+
```

Figure 3: Since release version 5.1, the *OpenSSH* software has offered the option of visualizing the host key of the server that the user is connecting to as a piece of ASCII art using a random walk algorithm. See Loss et al. [17] for details.

visualization for user-governed password validation. The focus of this work is thus on secure operability and the user experience of user-governed password validation, in order to provide a usable and secure process for the user in cases where automatic validation of the entered password is not viable. At the same time, the validation process is embedded in a positive user experience, as it makes use of image recognition, a strength of human cognition. In addition, the effects of color theory may stimulate the user positively during the validation process.

2 RELATED WORK

This article builds directly on the work presented by Perrig & Song [21], who introduce a definition and a number of quality requirements for Hash Visualization Algorithms (HVA). They also examine Random Art [2] as a possible implementation and provide a method for statistical analysis of the likelihood of random image collisions. Dhamija [11] expands upon the "image portfolio" idea of using hash visualizations for passwordless user authentication, which is distantly related to the approach we pursue here.

Since Perrig & Song's original publication [21], a small number of software projects have put their idea of hash visualization into practice, perhaps the widest-known one being OpenSSH's visual fingerprint [17]. Their use case is the verification of a server key fingerprint, which has traditionally been displayed as a series of hexadecimal numbers. The visual fingerprint adds a simple image based on a random walk algorithm to the key verification form (see figure 3). Cipriani [6] extends this implementation to showcase visual improvements that would be possible if compatibility with legacy systems were not a priority for OpenSSH. The *IBM*

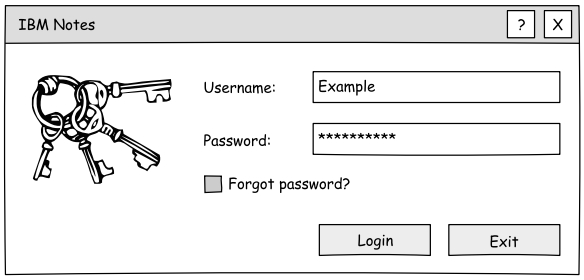

Figure 4: Schematic recreation of an IBM Notes 9 user authentication dialog. As the user types their password, the keys shown in the image on the left change position, size and color. Before submitting their password, the user can check if the image is the same as always – if not, the password has been entered incorrectly.

Notes software suite offers a dynamic visualization to allow users to verify their own password as it is being typed (see figure 4), which is in essence the same approach we pursue in this article, although the algorithm in question has not been published and we do not know if it makes any use of hashing. From seeing its output, we get the impression that it is much simpler and more prone to image collisions than the other HVAs mentioned.

There are also several standalone HVAs that have been developed and published over the years, such as Identicon [20], Vash [7], Robohash [10] and vizHash [23].

The aforementioned HVAs all produce relatively large and complex images. In contrast, there have been a number of HVA implementations for real-time password validation that restrict their visualizations to the password field itself. Examples of this approach are Chroma-Hash [27], HashMask [9] and Paul Sawaya's work on visual hashing [24]. By aiming to be integrated into existing password GUIs without any redesign, these implementations are constrained to a very small space, hence the image complexity that they can offer is reduced compared to HVAs intended for larger, separate images such as the one presented in this paper. Additionally, the password masking characters can potentially encroach

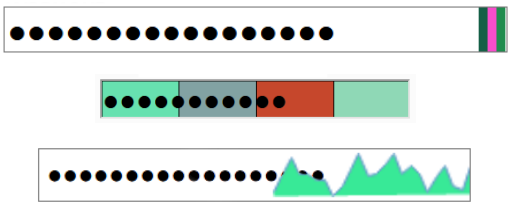

Figure 5: Three separate implementations of password hash visualization within the password input field. From top to bottom: Chroma-Hash [27], Paul Sawaya's visual hashing [24] and HashMask [9].

on the visualization, when given a password of sufficient length. See figure 5 for example visualizations of these algorithms.

Outside of hash visualization, other approaches for real-time user-governed password validation have been proposed. Chris Dary's HalfMask [8] shows password input characters as plain text, but visually overlays several nonsense characters onto each real character, effectively rendering the password unreadable but retaining some visual recognizability for the user. Khamis et al. [15] showcase the similarly functioning *Passquerade*, which renders the textual password through an image filter that distorts the characters and makes them mostly illegible. The idea behind the approach is that the user, who has knowledge of the correct password, is able to see any typing mistakes even in the distorted characters, whereas an over-the-shoulder attacker who does not know the password is unable to read it. Khamis et al. demonstrate some empirical success. Gruschka and Lo Iacono [12] propose *TransparentMask*, a system in which different shapes and colors are used for masking characters, either per character or after character groups of specific lengths. Their work also includes a stochastic risk assessment.

Chatterjee et al. [5] aim to attack the problem of typing mistakes in user passwords from the other direction by providing typo-tolerant password authentication. In their scenario, a network-based password authentication system would still require the password to be mostly correct, but would accept common typos in the correct password instead of requiring an exact match.

Tan et al. [26] present a detailed study of public key fingerprint comparison methods, including several hash visualizations in addition to numerous textual and numerical representations. They conclude that visual representations can significantly improve the user experience of the comparison process, and that the visualization algorithm should be carefully chosen with robustness in mind for scenarios where man-in-the-middle attacks are a concern.

To summarize, previous work has been published on hash visualization in general and in relation to user authentication, including work on password hash visualization within password input fields, as well as ideas for user-governed password validation via novel kinds of password masking or other ideas. As far as we can tell there has as yet been no scientific inquiry into hash visualization for user-governed password validation as a dedicated GUI element showing images of higher complexity, which is the combination of ideas presented in this article.

3 ALGORITHM

MosaicVisualHash is an algorithm that turns a bit sequence of arbitrary length into an image. The conversion process does not take the requirements of cryptographic hash functions into account. If MosaicVisualHash is used for password visualizations, a suitable hash function (such as SHA-256) should be used as an intermediate step before the input is passed to the algorithm.

The visualization has two configurable parameters that influence the number of input bits the algorithm can use: the number of curves and the number of colors in the image. The default values are 6 and 3 respectively, but the system designer can adjust these values. The upper bound to the number of input bits is then calculated as $\textit{number-of-curves} \times 24 + \textit{number-of-colors} \times 8 + 16$, or $6 \times 24 + 6 \times 8 + 128 = 320$ for the default parameter values. The visualization process needs exactly that number of bits, so depending on the length of the input bit sequence, the preparation portion of the algorithm can perform either a collapse or an extension of the input bits. If more than the required number of input bits is provided, the bits at indexes larger than the maximum are combined with the previous input bits by performing a modulo operation on the index and an XOR operation on the new input bit and the corresponding earlier bit. This is to ensure that every input bit has an influence on the final image. If the input bit sequence is too short to serve as the visualization basis, it is simply repeated.

MosaicVisualHash then splits the resulting bit sequence into a number of segments. The first segment of length $\textit{number-of-curves} \times 16$ describes the individual parameters of the curves that appear in the image. The algorithm uses 16 bits for each curve. Three bits are used for the curvature (or in other words, the radius of the circle segment), five for the rotation angle, and four each for the X and Y displacement. The second segment is the basis for the color palette. Each color uses 16 input bits from the sequence. The first color uses 8 bits for the hue and another 8 bits for the luminance. After that, the details of how the remaining bits are converted into colors depends on the number of colors requested. If just one color is requested, nothing else happens. For two or three colors, MosaicVisualHash attempts to pick complementary colors in order to ensure a visually pleasing image. The individual areas of the image are colored based on the remainder of the number of overlaid circles at a specific point divided by the number of available colors, ensuring that neighboring areas receive distinct colors where possible.

The input bit sequence has no direct influence on the color of the curve contours. The system designer can set this color to a fixed value, or if none is provided, MosaicVisualHash will attempt to pick either black or white as a contrast depending on the average luminance of the other colors in the image.

Figure 1 shows several example images produced by MosaicVisualHash based on random input data.

MosaicVisualHash was developed with the consideration that the resulting images should be relatively pleasing to the average user's visual preferences. We have not attempted to verify empirically whether we have been successful in that regard.

It is worth noting that the images created by MosaicVisualHash, when used in a security-related context, rely in part on the user's ability to recall and differentiate the colors used in the image. This process in turn depends on the algorithm making sensible color selections as well as the user's color sensing capability.

For the color selection, the algorithm takes a number of steps to ensure that the colors are high in contrast and saturation. First, the algorithm is restricted to a slice of the *hue, saturation, lightness* color space [16] that always maximizes the saturation and that removes the darkest 20% as well as the lightest 10% of the remaining colors from the possible palette. This happens in order to ensure that picking several colors of different hues will certainly lead to sufficiently different colors, even if many or all of them happen to be very high or low in lightness. See figure 6 for a visualization of the color space. For the first color to be chosen, the algorithm always maps its input bits onto this color space for a pseudo-random result. If a palette of two or three colors is requested, the remaining colors are chosen by moving a specific range of distances in the hue dimension (again influenced by the input bits) to make it highly likely for contrasting colors to be picked. If four or more colors are requested, MosaicVisualHash picks them via consecutive jumps by the golden angle along the hue circle, a process that maximizes the distance between colors for an indeterminate palette size [1].

Regarding the user's ability to differentiate the colors presented, we make no specific claims. Color sensitivity varies across populations, with non-negligible portions of all humans suffering from some kind of color vision deficiency such as red-green blindness [29]. Design recommendations for systems that are accessible to users with vision deficiencies generally focus on ensuring that colors are picked in such a way that they are sufficiently differentiable for all users [14]. In the case of password hash visualization, it is not a major concern whether all users perceive the provided images the same way, or whether the perceived color contrast is comparable across the user population. The central goal is to ensure that a specific user will perceive the same image the same way every time, which is not something that requires any particular color choice considerations. Thus, established best practices for "designing for colorblindness" are not necessarily transferable to password hash visualization. Further research may bring more clarity in regards to what measures could be helpful.

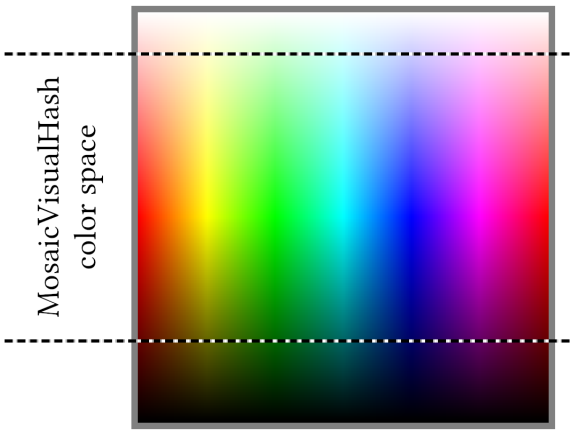

Figure 6: As the base palette for its images, MosaicVisualHash uses a 2D slice of the HSL color space [16] where the saturation value is maximized, i.e. no grays appear, as shown here. Further, the lightness value is clamped to a range excluding the darkest 20% and the lightest 10% of the color space, in order to avoid HSL color specifications where the hue would have too low an influence on the final color. The area within the dashed lines constitutes the color space from which the algorithm can select its colors.

MosaicVisualHash has been implemented in the JavaScript programming language and is available to the public as open source software[2].

4 DESIGN RECOMMENDATIONS

A successful implementation of MosaicVisualHash requires some additional considerations beyond the process of the visualization algorithm itself. In this section, we will describe one existing practical use case for real-time password hash visualization and explain several measures we took to strengthen practical security and privacy.

This use case employs MosaicVisualHash as a tool for user-governed password validation. The user enters a password into an interactive password text field via a keyboard, one character at a time. The password field hides the characters behind asterisks or another masking character, so the password is invisible to an over-the-shoulder attacker and screen capturing methods. After the user has finished typing the password, they can look at the visualization. If the visualization looks the same as it did for previous times when the user entered the same password, they can be reasonably certain that the password matches (i.e. the password itself was correctly recalled and the transfer from user memory into computer memory via the keyboard did not introduce any errors). This way, the user can verify the presence of the

[2]GitHub: https://github.com/jfietkau/Mosaic-Visual-Hash

correct password without any software-based validation of the specific password being used.

Ideally, the user would be able to glance at the visualization immediately after they have finished typing their password. However, as system designers we do not know the length of the password in advance. From this point we have two options: we could delay the visualization until the user triggers it via some deliberate action (e.g. pressing the "Enter" key or clicking a button), or we could display the visualization continuously, as the password is being typed.

Delaying the visualization until a separate user action is performed has the advantage that a visualization is only visible when it is needed, after the complete password is present. The main disadvantage is that it introduces friction into the user experience: Ideally, we expect that the user would want to verify the password right after they have typed it, without having to trigger the action separately. Notably, if the visualization differs from what the user expects, they would presumably want to be able to retype their password without unfocusing and then refocusing the password input field.

If we perform the visualization after every keystroke performed in the password field, this would allow the user to verify the visualization immediately. The computation time for a MosaicVisualHash image is nearly instantaneous on modern computers, thus it is completely feasible to visualize the hash continually, as the password is being typed. The advantage compared to variant one is a removal of a delay in the user's workflow and reduction of friction in the user experience. However, this approach introduces a big security problem: If the password hash is being instantly visualized as the password is typed, character by character, an attacker would be able to use a video recording of the visualization (whether taken over the shoulder or using screen recording software) to easily reconstruct the password. Passwords gain their security through the combinatorial explosion of possibilities for the characters in a password of sufficient length, but if only one character is added at a time, an attacker can easily test all the different characters on a typical keyboard until the visualization matches the previously captured one.

The usability and user experience of our approach is vitally important to us. That is why we developed several ideas to mitigate the security issue introduced by variant two in order to ensure a smooth user experience.

Minimum password length before visualization: As time goes on, the recommended minimum length for a password has increased. As of this article, a sensible minimum password requirement might be 12 characters, but this number might well increase over the years and decades. If we assume that any password shorter than the minimum password length is trivial, then it makes sense to delay the visualization until the user has entered enough characters to satisfy the minimum password length. This would significantly reduce the feasibility of the video-based attacks mentioned above.

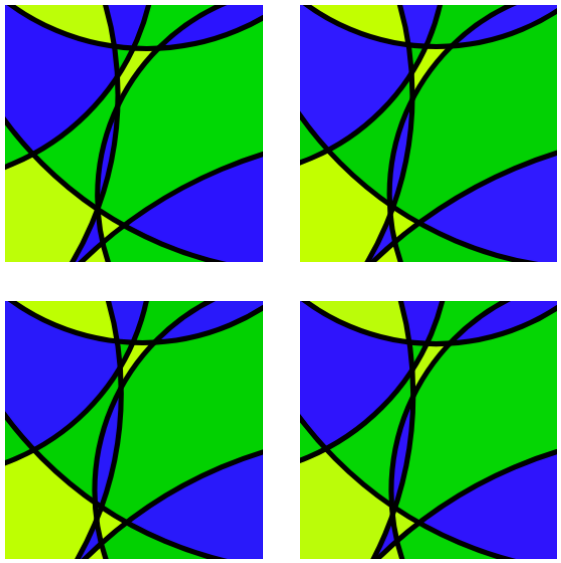

Figure 7: Four MosaicVisualHash output images for the same input at default jitter settings. Note the minor deviations in the colors and line positions in the image, intended to not be noticed by a human viewer comparing a current image to their recall of a previous one.

Artificial visualization delay: The algorithm is computationally simple enough to perform its task practically instantly. However, it could be wise to introduce an artificial delay into the visualization process. From previous research and practical experience, we know that the average time between user keystrokes is much shorter than the time it takes the user to shift their mental focus from the keyboard to the screen. If we delay the visualization by a short amount of time, such as a few hundred milliseconds, it would likely still be fast enough for a smooth user experience while also not exposing intermediate visualizations to video-based attacks.

Visual jitter: The images generated by the algorithm are to be compared by humans, so they do not need to have pixel-perfect accuracy to be perceived as "the same image". On the contrary, it makes sense to introduce minor deviations to the colors and the position of the image components in order to make it more difficult for an attacker to gain information about the password based on a screenshot of the hash visualization. We credit this idea to Paul Sawaya, who first documented it for his visual hashing work at Mozilla [24]. In MosaicVisualHash, there is a jitter setting with which the visual jitter parameters of the generated images are scaled. See figure 7 for an example.

5 SUMMARY

In this article we have presented a new hash visualization algorithm, which was also implemented in a demo application. Such algorithms open up the possibility to visualize password hashes in real time while the user enters a password. Thus the user is supported in recognizing and correcting errors within short timeframes. The proven human strength in processing structured images can be used by this application.

This technique is not strictly limited to the use case of non-networked password validation outlined above. For example, it can also be used in addition to standard network-based authentication procedures in order to allow the user a self-governed password check before they submit their data.

This article provides a detailed description of the new algorithm and simultaneously embeds it in a real use case[3]. It describes measures that have been taken to best support usable hash visualization for humans and at the same time protect against various attacks on the underlying system.

From the point of view of security, it should be of interest in future work whether the diversity of the images produced by the algorithm is sufficient. This would be particularly important in order to minimize MosaicVisualHash's susceptibility to visual collisions (cases where distinct inputs produce visually indistinguishable images). Similarly, it would be worth evaluating whether the algorithm parameters (number of curves and number of colors) should be adjusted to optimize the usability and the security of the process in regards to image perception and recall. In addition, an empirical study on the general usability or user experience of the generated images could also be conducted to determine to what extent they appeal to the user and thus create a positive user experience compared to other HVAs and password visualization/validation methods.

In summary, the developed MosaicVisualHash supports the user in secure authentication by visualizing their password hash in real time and simultaneously strengthening the security of a system by minimizing input errors due to the exploitation of human strength in image processing.

ACKNOWLEDGMENTS

We thank Prof. Dr. Michael Koch for constructive feedback on an early version of this article, and Mohamed Khamis for a thought-provoking discussion of password validation at CHI 2019 as well as encouraging us to pursue the publication of our ideas.

[3] MosaicVisualHash has been used for password hash visualization in an open-source password generator that is available to the public in a non-academic setting and has a small number of users.

REFERENCES

[1] Martin Ankerl. 2009. How to Generate Random Colors Programmatically. (Dec. 2009). https://martin.ankerl.com/2009/12/09/how-to-create-random-colors-programmatically/

[2] Andrej Bauer. 1998. Gallery of random art. (1998). http://andrej.com/art/ Original website defunct at the time of printing, now located at http://www.random-art.org/.

[3] Robert M. Boynton and D. E. Boss. 1971. The effect of background luminance and contrast upon visual search performance. *Illuminating Engineering* 66, 4 (1971), 173.

[4] Stuart K. Card, Thomas P. Moran, and Allen Newell. 1986. The model human processor: An engineering model of human performance. *Handbook of perception and human performance* 2, 45-1 (1986).

[5] R. Chatterjee, A. Athayle, D. Akhawe, A. Juels, and T. Ristenpart. 2016. pASSWORD tYPOS and How to Correct Them Securely. In *2016 IEEE Symposium on Security and Privacy (SP)*. 799–818. https://doi.org/10.1109/SP.2016.53

[6] Tyler Cipriani. 2017. Ssh Key Fingerprints, Identicons, and ASCII art. (2017). https://tylercipriani.com/blog/2017/09/26/ssh-key-fingerprints-identicons-and-ascii-art/

[7] Terrence Cole. 2011. Vash. (July 2011). http://www.thevash.com/ Website defunct at the time of printing, archive copy available at https://web.archive.org/web/20120428001217/http://thevash.com/.

[8] Chris Dary. 2009. HalfMask. (July 2009). http://lab.arc90.com/2009/07/08/halfmask-an-experiment-in-password-masking/ Website defunct at the time of printing, archive copy available at https://web.archive.org/web/20120205091026/http://lab.arc90.com/2009/07/08/halfmask-an-experiment-in-password-masking/.

[9] Chris Dary. 2009. HashMask. (July 2009). http://lab.arc90.com/2009/07/09/hashmask-another-more-secure-experiment-in-password-masking/ Website defunct at the time of printing, archive copy available at https://web.archive.org/web/20120226055300/http://lab.arc90.com/2009/07/09/hashmask-another-more-secure-experiment-in-password-masking/.

[10] Colin Davis. 2011. Robohash. (2011). https://robohash.org/

[11] Rachna Dhamija. 2000. Hash Visualization in User Authentication. In *CHI '00 Extended Abstracts on Human Factors in Computing Systems (CHI EA '00)*. ACM, New York, NY, USA, 279–280. https://doi.org/10.1145/633292.633455

[12] Nils Gruschka and Luigi Lo Iacono. 2010. Password Visualization beyond Password Masking. In *Proceedings of the Eighth International Network Conference (INC 2010)*, Udo Bleimann, Paul S. Dowland, Steven Furnell, and Oliver Schneider (Eds.). University of Plymouth School Of Computing, Communications And Electronics, 179–188.

[13] Jack Holmes. 2014. Remove password masking. (Sept. 2014). http://passwordmasking.com/

[14] Luke Jefferson and Richard Harvey. 2006. Accommodating Color Blind Computer Users. In *Proceedings of the 8th International ACM SIGACCESS Conference on Computers and Accessibility (Assets '06)*. ACM, New York, NY, USA, 40–47. https://doi.org/10.1145/1168987.1168996

[15] Mohamed Khamis, Tobias Seitz, Leonhard Mertl, Alice Nguyen, Mario Schneller, and Zhe Li. 2019. Passquerade: Improving Error Correction of Text Passwords on Mobile Devices by Using Graphic Filters for Password Masking. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems (CHI '19)*. ACM, New York, NY, USA, Article 686, 8 pages. https://doi.org/10.1145/3290605.3300916

[16] Haim Levkowitz and Gabor T. Herman. 1993. GLHS: A Generalized Lightness, Hue, and Saturation Color Model. *CVGIP: Graph. Models Image Process.* 55, 4 (July 1993), 271–285. https://doi.org/10.1006/cgip.

1993.1019

[17] Dirk Loss, Tobias Limmer, and Alexander von Gernler. 2009. The drunken bishop: An analysis of the OpenSSH fingerprint visualization algorithm. (2009). http://dirk-loss.de/sshvis/drunken_bishop.pdf

[18] George A. Miller. 1956. The magical number seven, plus or minus two: some limits on our capacity for processing information. *Psychological Review* 63, 2 (1956), 81–97. https://doi.org/10.1037/h0043158

[19] Jakob Nielsen. 2009. Stop Password Masking. (June 2009). https://www.nngroup.com/articles/stop-password-masking/

[20] Don Park. 2007. Visual Security: 9-block IP Identification. (Jan. 2007). https://web.archive.org/web/20080703155519/http://www.docuverse.com/blog/donpark/2007/01/18/visual-security-9-block-ip-identification

[21] Adrian Perrig and Dawn Song. 1999. Hash visualization: A new technique to improve real-world security. *International Workshop on Cryptographic Techniques and E-Commerce* (1999), 131–138. https://users.ece.cmu.edu/~adrian/projects/validation/validation.pdf

[22] Richard E. Reynolds, Raymond M. White, and Robert L. Hilgendorf. 1972. Detection and recognition of colored signal lights. *Human Factors* 14, 3 (1972), 227–236.

[23] SÃĪbastien Sauvage. 2011. VizHash GD - a visual hash. (2011). https://sebsauvage.net/wiki/doku.php?id=php:vizhash_gd

[24] Paul Sawaya. 2011. Visual Hashing. (Nov. 2011). https://wiki.mozilla.org/Identity/Watchdog/Visual_Hashing

[25] Bruce Schneier. 2009. The Pros and Cons of Password Masking. (July 2009). https://www.schneier.com/blog/archives/2009/07/the_pros_and_co.html

[26] Joshua Tan, Lujo Bauer, Joseph Bonneau, Lorrie Faith Cranor, Jeremy Thomas, and Blase Ur. 2017. Can Unicorns Help Users Compare Crypto Key Fingerprints?. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems (CHI '17)*. ACM, New York, NY, USA, 3787–3798. https://doi.org/10.1145/3025453.3025733

[27] Mattt Thompson. 2009. Chroma-Hash. (July 2009). https://github.com/mattt/Chroma-Hash/

[28] L. G. Williams. 1966. The effect of target specification on objects fixated during visual search. *Perception & Psychophysics* 1, 5 (1966), 315–318.

[29] Bang Wong. 2011. Points of view: Color blindness. *Nature Methods* 8, 441 (May 2011). https://doi.org/10.1038/nmeth.1618

[30] Luke Wroblewski. 2012. Mobile Design Details: Hide/Show Passwords. (Nov. 2012). https://www.lukew.com/ff/entry.asp?1653