Echo Back in Implementation of Passphrase Authentication

Kanta Matsuura*
(Institute of Industrial Science, University of Tokyo, JAPAN)

Abstract

In spite of well-known vulnerabilities, password-based authentication is still widely used. One possible improvement is to use long passphrases. But unfortunately, the longer passphrases are, the more likely users mis-stroke. To make matters worse, since user-authentication interfaces are usually implemented without echo-back of stroked characters, users do not notice their mis-strokes before they finish the long inputs. In order to solve this problem, this paper proposes an echo-back scheme; the monitor displays a chain of hashed values instead of asterisks. Its effect is studied in terms of expected number of total strokes. The study suggests an optimal strategy for the chaining and echo-back. It is also suggested that we can use the same strategy without customizing it. As an extension, image-based echoes are discussed as well.

Keywords: user authentication, password, passphrase, hash function, echo-off problem.

1 Introduction

User authentication is a central security issue. A taxonomy of user authentication tells us that we have (1) token-based systems, (2) systems based on biometrics, (3) knowledge-based systems, and (4) recognition-based systems.

Token-based systems: The system regards the owner of a specific device or token as a legitimate user. If a user carelessly leaves their token at home, he could not use the system during the trip. If someone else steals a token or luckily finds a lost token on a public road, he may be able to use it. Thus the system is often combined with knowledge-based authentication; ATM authentication requires a bank card and a PIN (Personal Identification Number), for example.

Biometrics: The system verifies a specific biometrics property of a user; fingerprint, for example. Users may hate this due to privacy reasons. The system equipment tends to be expensive. A more significant problem is its unrevokability. If a PIN is compromised, the user can update it into a new one. But he couldn't replace his fingerprint with a new one.

Knowledge-based systems: Users show some knowledge such as PINs or passwords to the system. Simple or meaningful passwords are easier to remember but vulnerable to attacks; too short ones can be revealed by an exhaustive-search attack while too meaningful ones by a dictionary-search attack. Complex, arbitrary or meaningless, and long passwords are more secure themselves, but are difficult to remember. So users tend to write them down. This in turn reduces the system security in our real world. In addition, the more severe the requirement for PIN quality, the more likely they use similar or even identical PINs in different systems. This means that all the systems are only as secure as the most weakest one.

Recognition-based systems: Not a user but the system shows several items and the user makes appropriate response to them. This can be regarded as a challenge-response extension of knowledge-based systems but, from the view point of human factors, more secure when we use images which are difficult to write down or tell to other guys; images generated by Random Art[1], [2], for example.

Looking at current real systems, in spite of those well-known vulnerabilities and proposals of other mechanisms, password- or PIN-based systems are still widely used in user authentication either by themselves or in combination with other schemes.

*Correspondence to: Dr. K. Matsuura, Institute of Industrial Science, University of Tokyo, Komaba 4-6-1, Meguro-ku, Tokyo 153-8505, Japan. E-Mail: kanta@iis.u-tokyo.ac.jp

It is well known that there have been a lot of efforts to improve the security of password authentication [3]–[6]. After all, even with those efforts, our passwords need entropy sufficient for the required security level [7], [8]. However, since people are not good at remembering a long meaningless chain of characters, the entropy requirement might cause a compromise resulting from writing them down. One solution to this problem is the use of long meaningful passphrases [9], [10]. For instance, let us suppose a system in which a passphrase is used to generate 64-bit encryption key. In this case, according to the estimate of entropy in English language[11], [8], we need 49- or 50-character passphrases. For example, the title of this paper itself is composed of 50 characters. Typical users probably accept and can remember passphrases of this length.

Thus passphrases may provide higher security but unfortunately degrade usability; the longer they are, the more likely users mis-stroke during authentication procedure. To make matters worse, since it is a common practice to implement passphrase-authentication interface without echo-back of stroked characters[12], [13], users do not notice their mis-strokes before they finish the long inputs and the log-in attempts are rejected. In order to solve this echo-off problem, this paper proposes an echo-back scheme by the use of one-way hash function; during passphrase-authentication procedure, the monitor displays a chain of hashed values instead of echo-off asterisks. If the user remembers part of the hashed values, he may notice his mis-strokes before finishing all the typing-in.

The rest of this paper is organized as follows. First, we describe how to echo back and how to remember the echo in Section 2. Next, in Section 3, the effect of the scheme is studied in terms of expected number of total strokes required for one successful log-in. After discussion of the results and implementation in Section 4, Section 5 concludes the paper with some remarks.

2 Echo-Back of Passphrase

We consider a passphrase composed of C characters and divide it into L blocks. Each of the first $L-1$ blocks has ΔC characters where $L = \lceil C/\Delta C \rceil$, and the last block has $C - \Delta C(L-1)$ or (C mod ΔC) characters. The j-th block will be denoted by p_j, in the following. In our echo-back scheme, the monitor displays a hashed value which is generated from a local secret, the user ID, and blocks which are already typed. This display is updated block by block, *i.e.*, when the user finishes typing a block. The user remembers part of the hashed values. If he finds a mismatch between the remembered value and the displayed value, he restarts the log-in procedure.

The details are as follows.

When a user X finishes typing the first block, the monitor displays

$$h_1 = h(h(ID_X, K), p_1)) \tag{1}$$

where ID_X is his user ID, K is a local secret of the machine for salting, and $h(\cdot)$ is a one-way hash function. The salting prevents off-line exhaustive search attacks after peeping. h_1 appears on the screen in d-ary representation. We assume that the output of the hash function is too long for X to memorize all the digits. So X remember not all but only m_1 digits of h_1, and if he finds a mismatch in them, he clears the input and restarts to type his passphrase from the first character. If no mismatch is found, he goes on to the second block. On the assumption that the hash function outputs random values, the probability of a mis-stroke being noticed as a mismatch is

$$1 - \left(\frac{1}{d}\right)^{m_1}. \tag{2}$$

When X finishes typing the second block, the monitor displays

$$h_2 = h(h_1, p_2). \tag{3}$$

Again, X remember not all but only m_2 digits of h_2, and if he finds a mismatch in them, he clears the input and restarts to type his passphrase from the first character. If no mismatch is found, he goes on to the next block. The same procedure is operated until the second final block, that is,

$$h_j = h(h_{j-1}, p_j) \quad (j = 2, 3, \cdots, L-1). \tag{4}$$

When X finishes typing the final block, the monitor does not need to echo; it just tells X the authentication result either by allowing the log-in or by rejection.

Thus, with a certain high probability, the user can notice his mis-stroke before finishing all the input. The effect of this "earlier notice and restart" will be studied in the next section.

3 Evaluation

In this section, we evaluate the effect of the proposed echo-back scheme in terms of expected number of total strokes required for one successful log-in. This number is denoted by S, in the following.

We assume that the memory capacity of the user is limited to a certain amount;

$$M = m_1 + m_2 + \cdots + m_L \tag{5}$$

where M is a constant integer. Since the hashed values are random, it is easier for users to remember the same number of digits per block. Therefore, we consider a memory strategy such that

$$m_1 = m_2 = \cdots = m_k = m \quad \text{(positive constant)}, \tag{6}$$

$$m_{k+1} = M \bmod m, \tag{7}$$

and

$$m_{k+2} = m_{k+3} = \cdots = m_L = 0. \tag{8}$$

In general, users tend to type more carefully and slowly after an unsuccessful log-in attempt. For simplicity, we consider two types of users;

Type 1 users become careful enough to type correctly in the second attempt.

Type 2 users make mis-strokes with the same probability in the second attempt as in the first attempt but become careful enough to type correctly in the third attempt.

Both have their restart from the first character of the passphrases after noticing the mismatch of the echo. We do not consider users who restart their typing from the previous block rather than the very first character. This is because long passphrases are difficult to restart on the way (*e.g.* it is difficult to remember what is the "26th" character of a long passphrase). Then, assuming that mis-stroke probability e (in the first attempt and in the second attempt of Type 2 users) is common for every character, the expected number of strokes S is given by

$$S = C(1-e)^C + \sum_{j=1}^{L-1} \beta_j \left(j\Delta C + C\right) + 2C\beta_L \tag{9}$$

in the case of Type 1 users, and

$$\begin{aligned} S &= C(1-e)^C + 2C\beta_L(1-e)^C \\ &\quad + \sum_{j=1}^{L-1} \beta_j \left(j\Delta C + C\right)(1-e)^C \\ &\quad + \sum_{i=1}^{L-1}\sum_{j=1}^{L-1} \beta_i\beta_j((i+j)\Delta C + C) \\ &\quad + 2\sum_{j=1}^{L-1} \beta_j\beta_L(j\Delta C + 2C) + 3C\beta_L^2 \end{aligned} \tag{10}$$

in the case of Type 2 users. β_j is the probability of "notice and restart" after the input of the j-th block, and can be recursively derived from

$$\alpha_1 = 0 \tag{11}$$

$$\alpha_j = \frac{(1-e_{\text{block}})^{j-2}\, e_{\text{block}} + \alpha_{j-1}}{d^{m_{j-1}}} \qquad (j = 2, 3, \cdots, L) \tag{12}$$

$$\beta_j = \left\{(1-e_{\text{block}})^{j-1}\, e_{\text{block}} + \alpha_j\right\}\left(1 - \frac{1}{d^{m_j}}\right) \qquad (j = 1, 2, \cdots, L-1) \tag{13}$$

$$\beta_L = (1-e_{\text{block}})^{L-1} e_{\text{last}} + \alpha_L \tag{14}$$

where $e_{\text{block}}=1-(1-e)^{\Delta C}$ is the probability that mis-strokes occur in a block of length ΔC while

$$e_{\text{last}} = \begin{cases} 1-(1-e)^{\Delta C} & (\text{if } \Delta C | C) \\ 1-(1-e)^{C \bmod \Delta C} & (\text{otherwise}) \end{cases} \tag{15}$$

is that in the final block.

Given the passphrase length C, the memory capacity M, and the mis-stroke probability e, the expected number of total strokes S depends on the memory strategy determined by ΔC and m. We of course want to use an optimum strategy to make best use of our memory capacity. Therefore, by exhaustive search, we find the optimum ΔC and m which minimizes S for different values of user's mis-stroke probability.

Throughout the following study, we use a display representation of d=10. This is because users are accustomed to memorizing a numerical character from 0 to 9 in the case of meaningless object[1].

Figure 1 shows the evaluation results when C=50 and M=10. As mentioned earlier, the passphrase length C=50 corresponds to 64-bit key generation. The additional memory burden of M=10 means that users make an effort as heavy as remembering telephone numbers. As for users with better or larger memory, we also studied the case of C=50 and M=12. The result is shown in Fig. 2. In those figures, the performance of our echo-back scheme is plotted by marks ("o" in the case of Type 1 users while "x" in the case of Type 2 users) and that of conventional echo-off implementation is plotted by lines (solid line in the case of Type 1 users while dotted line in the case of Type 2 users).

The difference between the lines and the marks shows how significantly the performance is improved by the proposed scheme. This significance depends on the mis-stroke probability e per character. In most likely probability region between e=0.1% and 10%, the reduction in the expected number of total strokes is larger than 50%.

[1]We say "meaningless" when the object has no understandable linguistic meaning for the user.

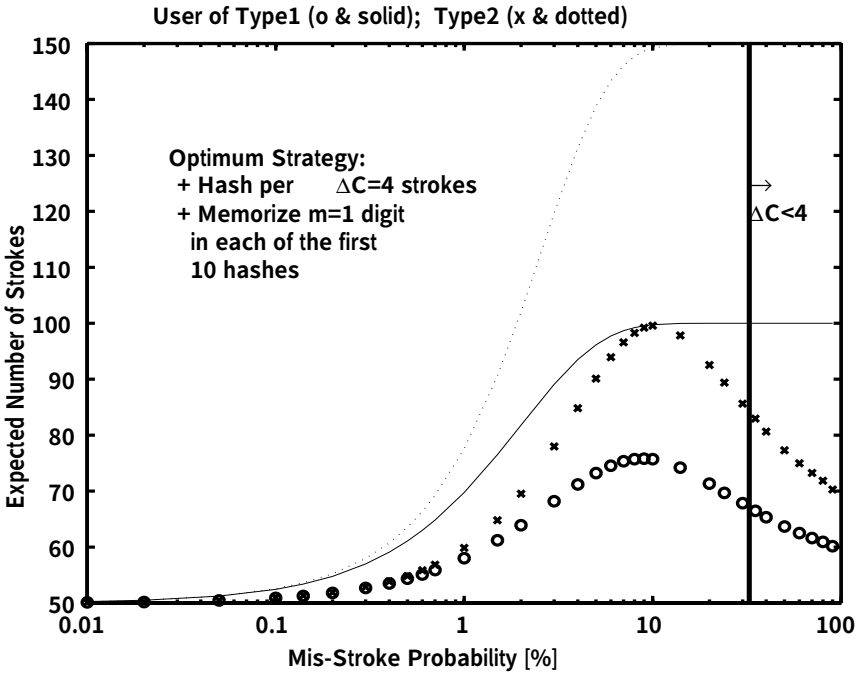

Figure 1. Expected number of total strokes required for one successful log-in by the optimum strategy. Users are assumed to be able to memorize M=10 digits in echo. When the mis-stroke probability is smaller than 32.5%, the optimum strategy is to update the hashed value every ΔC=4 inputs and remember m=1 digit per hash. For reference, the solid line shows the expected number of strokes in the case of Type 1 users with echo off. This is reduced to the circle marks (o) by the proposed echo-back scheme. Likewise, the dotted line is in the case of Type 2 users with echo off. This is reduced to the cross marks (x) by the proposed echo-back scheme.

Regarding users' burden, there is only a little difference between the performances for M=10 and M=12. So we can recommend users the more light one, *i.e.*, to remember 10 digits in the echo.

4 Discussion

4.1 Implementation

Here we discuss the implementation of our scheme. In the case of the recommended memory burden of M=10 (Fig. 1), fortunately, we can see a very stable result; in the region of $e < 32.5\%$, the optimum strategy is $(\Delta C, m)$=(4, 1) for both Type 1 users and Type 2 users. Since this range of mis-stroke probability seems to cover carelessness of most real users, we can in practice use the same strategy where a hash chain is presented every 4 characters typed in and users are recommended to remember one digit (probably the first or the last digit is the easiest for most users) of each of the first 10 hashed values. If users feel frustrated to see a display of all the hashed values including digits which are not remembered, an actual implementation can display only one digit of each hashed value, or alternatively, display the sum of all the digits modulo 10. This implementation is sketched in Fig. 3. For users with better memory, the implementation continues to echo until the second last block.

The followings are extensions which would need user studies with actual implementation although theoretically we can use Eqn.(9) and Eqn.(10). The need is due to the cognitive issues, which are not trivial.

First, for security reasons against peeping, one might be able to attach dummy echoes on the screen. For example, 10 rows of numerical characters are displayed, among which only one row represents the actual echo. When the user registrates himself, he determines which row should play the role.

Second, practitioners can consider simple graphics or images in place of numerical characters for the echo. Let us suppose 10 simple but distinguishable images. Instead of displaying the digits, the screen displays the corresponding images. Random arts [1], [2] would be

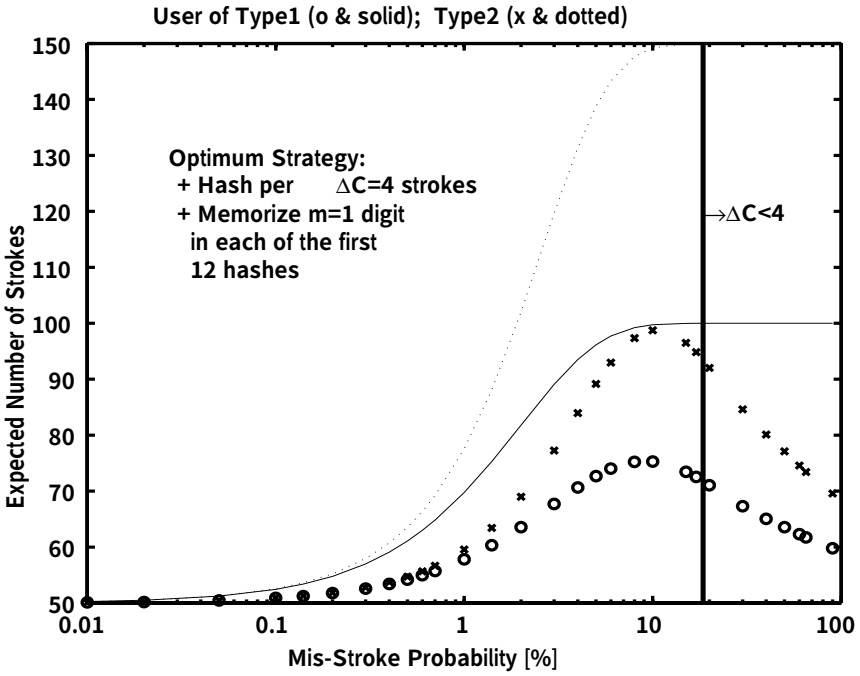

Figure 2. Expected number of total strokes required for one successful log-in by the optimum strategy. Users are assumed to be able to memorize M=12 digits in echo. When the mis-stroke probability is smaller than 18.5%, the optimum strategy is to update the hashed value every ΔC=4 inputs and remember m=1 digit per hash. For reference, the solid line shows the expected number of strokes in the case of Type 1 users with echo off. This is reduced to the circle marks (o) by the proposed echo-back scheme. Likewise, the dotted line is in the case of Type 2 users with echo off. This is reduced to the cross marks (x) by the proposed echo-back scheme.

worth a try. The next subsection will discuss this extension more in detail.

4.2 Graphic Echoes

When we want to display some graphics or images as echoes, at least the following questions would occur:

- How many distinguishable images can be used? That is, we want to know an appropriate assignment of the parameter d.
- How many images can be memorized? That is, we want to know an appropriate assignment of the parameter M.
- Are the images easy to tell other guys? We prefer difficult-to-tell images for security reasons.

The final question recommends the use of Random Arts. Random Art is a prototypical solution for hash visualization[1]:

Definition 4.1 (HVA) *A hash visualization algorithm (HVA) is a function h_I which has, as a minimum, the following two properties:*

1. *Image-generation: h_I maps an input x of arbitrary finite length, to an output image $h_I(x)$ of fixed size.*
2. *Ease of computation: Given h_I and an input x, $h_I(x)$ is easy to compute.*

Definition 4.2 (Near Images) *Two images I_1 and I_2 are said to be* near*, denoted as $I_1 \simeq I_2$, if the two images are perceptually indistinguishable.*

Regarding the use of Random Arts for recognition-based authentication, a user study[2] suggests that users can tell $M = 5$ out of $d = 25$ distinguishable images with a burden and performance similar to those for 4-digit PINs. Therefore, we studied the effect of our echo-back for the parameter assignment of $(M, d) = (5, 25)$. The situation is summarized as follows:

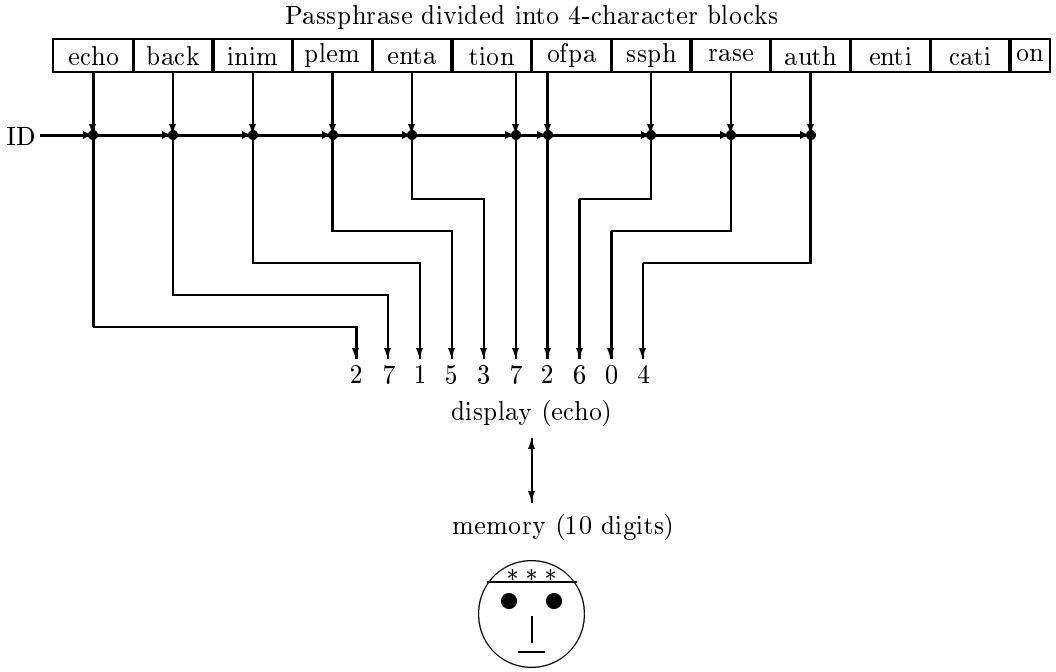

Figure 3. An optimum implementation of echo-back for authentication by 50-character passphrase. The sample passphrase is the title of this paper (without blanks).

- There are $d = 25$ distinguishable images or Random Arts I_1, I_2, $\cdots$, I_{25}.
- After the first block input, the machine represents h_1 as a base-25 integer.
- Each base-25 digit is displayed as an image. For example, if $h_1 = (2A3 \cdots 6E)_{25}$, then I_{14}, I_6, $\cdots$, I_3, I_{10}, and I_2 are displayed.
- The rest of the blocks are processed in the same way.
- The user remembers $M = 5$ displayed images in total.

Figure 4 shows the evaluation results. As usual, the performance of our echo-back scheme is plotted by marks ("o" in the case of Type 1 users while "x" in the case of Type 2 users) and that of conventional echo-off implementation is plotted by lines (solid line in the case of Type 1 users while dotted line in the case of Type 2 users). The improvement from the echo-off scheme is similar to that in the case of numeral display. By contrast, the optimum strategy is very different. In most likely mis-stroke probability region between e=0.1% and 10%, the echo increases every $\Delta C = 9$ or 8 inputs. This is less frequent than in the case of numeral display.

5 Concluding Remarks

User authentication is an important entrance to secure systems. Currently, despite well-known vulnerabilities, password- or PIN-based systems are widely used either alone or in combination with other authentication methods. Long meaningful passphrases can improve security but degrade usability due to its long input with echo-off on the screen. In order to solve this echo-off problem, this paper proposed an echo-back scheme with the help of one-way hash function.

Since users can notice their mis-strokes before they finish the long inputs, the proposed scheme can significantly reduce the expected number of total strokes required for one successful log-in. This reduction rate depends on the mis-stroke probability per character. In most likely probability region between 0.1% and 10%, the reduction is larger than 50%. If a user finds it difficult to accept the additional memory burden for the echo, he can just ignore the echo; the resultant performance is as good as that in the conventional echo-off implementation, anyway.

Fortunately, in a wider area including the mis-stroke probability region, we can use the same strategy of hashing every 4 characters and remembering one digit per hash. This can simplify the implementation. In an

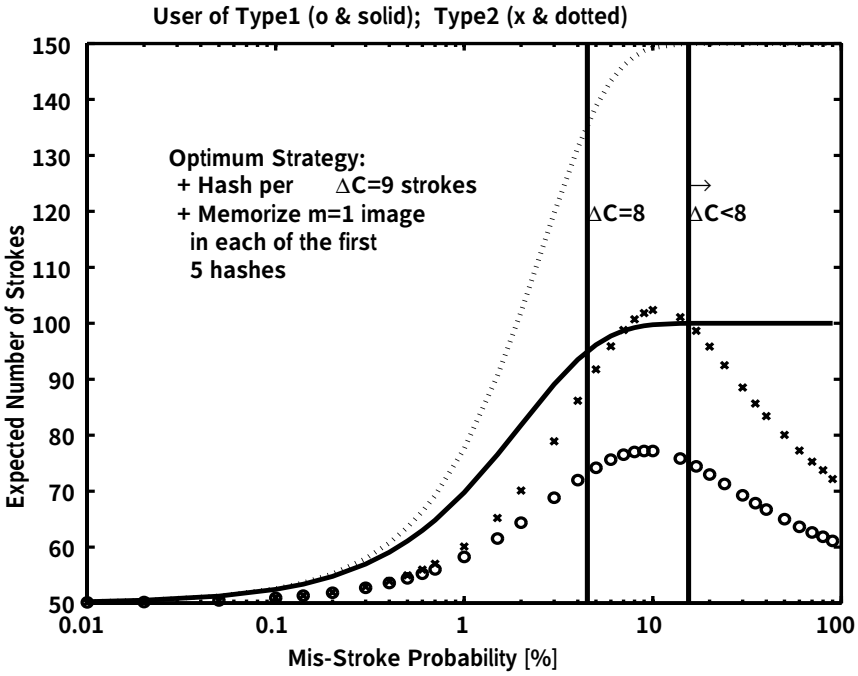

Figure 4. Expected number of total strokes required for one successful log-in by the optimum strategy. Users are assumed to be able to memorize M=5 images in echo. For reference, the solid line shows the expected number of strokes in the case of Type 1 users with echo off. This is reduced to the circle marks (o) by the proposed echo-back scheme. Likewise, the dotted line is in the case of Type 2 users with echo off. This is reduced to the cross marks (x) by the proposed echo-back scheme.

extended discussion for the replacement of the numerical echoes with simple but distinguishable graphics, we found that the optimal echoes would be updated less frequently.

As a further research, a user study is needed as well as a real implementation. The experimental study shall include performance analysis not only in the number of strokes but also in the net time per log-in. It could also include extensions of the proposed scheme: (1) the use of graphics or images instead of numerical characters, and (2) the parallel display of dummy echoes.

Acknowledgment

This work is partly supported by Research Assistance Program 2000 of the Asahi Glass Foundation.

References

[1] A. Perrig and D. Song. "Hash Vsualization: A New Technique to Improve Real-World Security". In *Proceedings of the 1999 International Workshop on Cryptographic Techniques and E-Commerce (CryTEC'99)*, Hong Kong, 1999.

[2] R. Dhamija and A. Perrig. "Déjà Vu: A User Study Using Images for Authentication". In *Proceedings of 9th USENIX Security Symposium* (to appear)

[3] A. Evans, W. Kantrowitz, and E. Weiss. "A User Authentication Scheme Not Requiring Secrecy in the Computer". *Communications of the ACM*, Vol. 17, No. 8, pp. 437–442, August 1974.

[4] G. P. Purdy. "A High Security Log-In Procedure". *Communications of the ACM*, Vol. 17, No. 8, pp. 442–445, August 1974.

[5] R. Morris and K. Thompson. "Password Security: A Case History". *Communications of the ACM*, Vol. 22, No. 11, pp. 594–597, November 1979.

[6] G. Brassard. *Modern Cryptology: A Tutorial.* Springer-Verlag, 1988. Lecture Notes in Computer Science 325.

[7] D. C. Feldmeier and P. R. Karn. "UNIX Password Security — Ten Years Later". In *Advances in Cryptology — CRYPTO'89*, pp. 44–63, 1990. Springer-Verlag. Lecture Notes in Computer Science 435.

[8] Bruce Schneier. *Applied Cryptography Second Edition: Protocols, Algorithms, and Source Code in C.* John Wiley & Sons, 1996.

[9] S. Porter. "A Password Extension for Improved Human Factors". In *Abstracts of CRYPTO'81*, p. 81, August 1981.

[10] C. P. Pfleeger. *Security in Computing.* Prentice Hall, 1989.

[11] C. Shannon. "Prediction and Entropy of Printed English". *Bell System Technical Journal*, Vol. 30, No. 1, pp. 50–64, January 1951.

[12] P. R. Zimmermann. *The Official PGP User's Guide.* MIT Press, 1995.

[13] P. Loshin. *Personal Encryption Clearly Explained.* Academic Press, 1998.