A Text-Graphics Character CAPTCHA for Password Authentication

Matthew Dailey
Sirindhorn International Institute of Technology
Thammasat University
Patumtani, Thailand 12121
Email: mdailey@siit.tu.ac.th

Chanathip Namprempre
Electrical Engineering Department
Faculty of Engineering, Thammasat University
Patumtani, Thailand 12121
Email: nchanath@engr.tu.ac.th

***Abstract*— We propose a new construct, the Text-Graphics Character (TGC) CAPTCHA, for preventing dictionary attacks against password authenticated systems allowing remote access via dumb terminals. Password authentication is commonly used for computer access control. But password authenticated systems are prone to dictionary attacks, in which attackers repeatedly attempt to gain access using the entries in a list of frequently-used passwords. CAPTCHAs (Completely Automated Public Turing tests to tell Computers and Humans Apart) are currently being used to prevent automated "bots" from registering for email accounts. They have also been suggested as a means for preventing dictionary attacks. However, current CAPTCHAs are unsuitable for text-based remote access. Our TGC CAPTCHA fills this gap. In this paper, we define the TGC CAPTCHA, prove that it is a (secure) CAPTCHA, demonstrate its utility in a prototype based on the SSH (Secure Shell) protocol suite, and provide empirical evidence that the test is easy for humans and hard for machines. We believe that the system will not only help improve the security of servers allowing remote terminal access, but also encourage a healthy spirit of competition in the fields of pattern recognition, computer graphics, and psychology.**

I. Introduction

Password authentication is one of the most common building blocks in implementing access control. Each user has a relatively short sequence of characters commonly referred to as a *password*. To gain access, the user provides his/her password to the system. Access is granted if the password is correct; it is denied otherwise.

A common attack against password authenticated systems is the *dictionary attack*. An attacker can write a program that, imitating a legitimate user, repeatedly tries different passwords, say from a dictionary, until it finds one that works.

There are several well-known ways to cope with dictionary attacks. For example, the system can deny access for the user in question after some number of tries, a technique known as account locking. However, this invites a denial of service attack: an attacker can lock anyone out of the system by submitting a sequence of incorrect passwords on behalf of the victim. Other solutions also have their own shortcomings [1].

In this paper, we present an alternative defense against dictionary attacks. The idea is to make it harder for automated programs to mount dictionary attacks by requiring the attacking programs to pass a test that is easy for humans but is hard (in terms of accuracy and/or compute time) for computer programs. A construct with this property is called a CAPTCHA (Completely Automated Public Turing test to tell Computers and Humans Apart) [2]. In particular, if there exists a program that can pass a CAPTCHA with high probability, then that program can be used to solve a hard AI problem. CAPTCHAs are already in use in some systems that benefit from distinguishing between humans and "bots" [3]. Recently, they have also been suggested as a means for deterring dictionary attacks in password authenticated systems [1].

One CAPTCHA suitable for password authenticated systems displays a degraded image of a word to the human, who then responds by typing the word he or she sees. A similar CAPTCHA using a sound clip instead of an image can also be used. However, these CAPTCHAs are not suitable for systems that allow remote access via consoles or dumb terminal programs. Our goal is to make it possible for such minimal systems to obtain the same benefits from CAPTCHA-assisted password authentication as systems with graphical displays and/or speakers.

To this end, we propose a new CAPTCHA based on Text-Graphics Characters. We call it the *TGC CAPTCHA*. A *text-graphics character* is an image of a character rendered on a text-only screen, namely a screen in which ASCII characters are used to represent pixels. Unfortunately, images rendered on a text-only screen are necessarily low resolution. This limits the number of characters that can be put on a single screen without hindering recognition. In fact, we find that on an 80x24 screen, it is unreasonable to display more than one or two distorted English characters. Without the contextual information afforded when complete words are displayed, humans have a harder time recognizing what is on each screen. Additionally, neither color nor grayscale are usually available to help provide visual clues. These limitations together make it difficult to generate text-graphics characters that are easy for humans but hard for computers to recognize.

Contributions. First, we precisely define the TGC CAPTCHA and prove that it is indeed a (secure) CAPTCHA according to the definitions in [2]. Specifically, we offer a reductionistic proof to show that, if a particular class of problems believed to be hard is in fact a hard AI problem as

defined in [2], then our TGC CAPTCHA is secure. As such, the term "prove" here is used in the same sense as that in the area of *provable security* [4], namely that the security of a construct, i.e. the CAPTCHA, is proved based on the security of a primitive, i.e. the corresponding hard AI problem.

Second, we also implement a prototype as part of the popular Secure Shell (SSH) protocol suite by incorporating our CAPTCHA into the user authentication mechanism of SSH, thus hardening SSH servers against dictionary attacks. The prototype is available as a source code patch for OpenSSH 3.6.1 at [5].

Finally, we provide empirical evidence that our CAPTCHA is easy for humans and relatively hard for machines by running the CAPTCHA test against human subjects and an Optical Character Recognition (OCR) program [6], respectively. The results are encouraging: on a character-by-character level, humans achieve 95% accuracy after two practice trials whereas the OCR program's accuracy is less than 35%. We believe that improving the OCR system's accuracy to approach that of humans will require more effort on the part of OCR system designers, and in any case, attackers using OCR systems to mount automated dictionary attacks will require a substantial amount of total compute time.

THE TGC CAPTCHA. Our CAPTCHA presents a sequence of k distorted characters, one at a time, to the user and accepts only correct responses ($k = 8$ in our implementation). To make the test easy for humans, we use only uppercase English characters. For each character, we pick n_d "distracters" and apply several operations to both the character and the distracters before laying the former over the latter. The operations are scaling, rotation, translation, and "row sliding," an operation involving horizontally moving each pixel in incremental steps, row-wise. The characters, the distracters, the operations, and all of the parameters are chosen at random from configurable ranges. The resulting image is then displayed to the user.

THE PROTOTYPE IN SSH. All our modifications are compliant with SSH specifications [7]. The server program, when configured with CAPTCHA support, informs the client that CAPTCHA-based authentication is a valid method. The client, when configured with CAPTCHA support, requests CAPTCHA authentication. The server then transmits a sequence of transformed images of characters to the client. The client displays the received images one by one and collects responses from the user before sending them all back to the server with the user's password (thus minimizing the impact of network delays). The server grants access to the user only if he/she *both* passes the CAPTCHA test *and* enters the correct password. The server denies access otherwise.

FEATURES OF THE TGC CAPTCHA AND THE PROTOTYPE. The TGC CAPTCHA provides a defense against online dictionary attacks without resorting to other countermeasures, e.g. account locking, with well-known disadvantages. Also, as discussed in [1], even if solving the underlying AI problem, in this case the problem of recognizing distorted text-graphics characters, turns out to be easy for OCR systems, it would still be useful to use the TGC CAPTCHA in password authenticated systems. The reason is that an attacker mounting an online dictionary attack still needs *both* to solve the CAPTCHA *and* to find the password in order to login successfully. In this situation, our approach reduces to *pricing via processing*, a paradigm originally proposed to combat senders of bulk junk email ("spam") [8]. In order to mount a dictionary attack, the adversary must perform a moderately complex computational task on each password authentication attempt. Thus, even if the attacker only spends a few compute cycles on each trial, the cumulative effect becomes significant over the many trials required for a dictionary attack.

We stress here that the use of the TGC CAPTCHA in password authentication is only one possible application of our CAPTCHA. Like other CAPTCHAs, the TGC CAPTCHA can be used in any application in which it is useful to distinguish human users from bots [2], [3]. Furthermore, the TGC CAPTCHA extends these benefits to applications with console-based interfaces. For example, it can help prevent bots from signing up for free email accounts or help conduct online polls via, say, text-based web browsers. Search engines that disallow bots by requiring graphical CAPTCHAs to be recognized will no longer end up automatically rejecting legitimate users who access the sites via a text-based interface, if the TGC CAPTCHA is made available to those users.

RELATED WORK. von Ahn et al. [2] were the first to propose the concept of CAPTCHAs. They formally defined the CAPTCHA construct and its security notion and specified two classes of AI problems. Our CAPTCHA is an instance of their second problem class. Pinkas and Sanders proposed the use of *Reverse Turing Tests* (RTTs), constructs similar to CAPTCHAs, to cope with dictionary attacks [1]. They focus on usability and scalability by using a persistent data structure and by requiring users to solve RTTs only some fraction of the time, respectively. Xu et al. proposed the use of character recognition to separate humans from machines [9]. However, they did not assume authenticated, replay-resilient channels. Consequently, the communication had to be protected via message authentication codes and serial numbers, timestamps, or state information. In contrast, our CAPTCHA challenges and responses are sent over SSH channels which are already encrypted, authenticated, and replay-resilient [10]. This dramatically simplifies our protocol.

NOTATION. Let k be a positive integer. We denote by $x_1, \ldots, x_k \xleftarrow{\$} X$ the act of sampling each element x_i uniformly and independently from the set X.

II. TEXT-GRAPHICS CHARACTER CAPTCHA

We call our CAPTCHA a Text-Graphics Character (TGC) CAPTCHA. We present it here following the formalization in [2]. Let $\mathcal{I}$ be a set of images of all upper case English characters, $\mathcal{T}$ be a set of transformations on images, λ be the map from an image of a character to the (ASCII ID of) the character portrayed in the image, and τ and k be the security

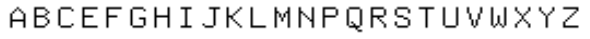

Fig. 1. Reference image set $\mathcal{I}$.

parameters. A TGC CAPTCHA instance is a tuple $\mathsf{TGC} = (\mathcal{I}, \mathcal{T}, \lambda, \tau, k)$ defining the following test. First, the verifier (i.e. server) draws $i_1, \ldots, i_k \xleftarrow{\$} \mathcal{I} - \{\text{'O'}, \text{'D'}\}$ and $t_1, \ldots, t_k \xleftarrow{\$} \mathcal{T}$. Then, it sends to the prover (i.e. user) the transformed images $t_1(i_1), \ldots, t_k(i_k)$ and sets the timer for τ. The prover responds with the labels $l_1, \ldots, l_k$, each of which is (the ASCII ID of) a character in the English alphabet. The verifier accepts if $l_j = \lambda(i_j)$ for all $1 \leq j \leq k$ and if the timer has not expired. It rejects otherwise.

We describe here our choices for the sets $\mathcal{I}$ and $\mathcal{T}$. The reference image for each character, from the standard X Window System "9x15" font, is shown in Figure 1. The transformation process involves the following steps. First, n_d distracters are chosen uniformly with replacement from the set of all distracter images. In our implementation, we use $n_d = 5$ samples from a set of 26 9x15 bitmaps that share some features with English letters but are easily classified as non-letters by humans. After converting each target and distracter bitmap to a 9x15 ASCII *charmap*, we perform the following operations on each charmap:

1) Scale: The charmap is scaled by a random factor between 1.3 and 1.7.
2) Rotate: The charmap is rotated by a random angle between -20 and 20 degrees.
3) Translate: The charmap is translated to a random location on the $n_c \times n_r$ screen with the constraint that the entire character must still be visible.
4) Row Slide: Each row of the charmap is optionally slid left or right relative to the row above. We shift left with probability 0.33, right with probability 0.33, and otherwise do not shift.

In our implementation, all of the random samples are from a uniform distribution and rely on the standard C library `random()` function (seeded with the system time). Finally, the transformed target and distracter images are overlaid as follows. Each overlay is given an opaque whitespace boundary, and the target image is laid down last, to ensure that the target stands out clearly from the background.

We believe that this set of transformations gives sufficient variability in the output images to make recognition fairly difficult for machines while preserving the necessary property that it is easy for humans. Experienced users (e.g. the authors) consistently classify the characters with 100% accuracy, and as we shall see in Experiment 1 below, even naive users perform well enough to make our construction practical for deployment in real systems.

III. CAPTCHA-Based Password Authentication

The TGC CAPTCHA can complement any interactive password authentication scheme, so long as a sufficiently large text console is available for displaying the CAPTCHA. In a TGC CAPTCHA-enabled authentication session, the user is challenged by a CAPTCHA test and asked to enter his/her password. If the user passes both the CAPTCHA test and enters the correct password, access is granted. Otherwise, access is denied.

We have implemented a prototype TGC CAPTCHA password authentication method compatible with the SSH user authentication protocol [7]. The implementation is based on OpenSSH 3.6.1, but the method could easily be incorporated into any SSH-compliant client or server. Here we describe the essential steps in a CAPTCHA-enabled password authentication exchange. The authentication protocol relies on the lower-level SSH transport protocol [10] to provide integrity and confidentiality.

1) The server informs the client about which authentication methods are available.
2) Assuming that CAPTCHA-enabled password authentication is available, the client requests CAPTCHA password authentication by sending a `SSH_MSG_USERAUTH_REQUEST` message. The message specifies "captcha-password" authentication and additionally contains a username and service to execute.
3) On receipt, the server creates a TGC CAPTCHA using configurable parameters k, n_c, and n_r (8, 80, and 24, respectively, in the default configuration) and sends the CAPTCHA challenge in a `SSH_MSG_CAPTCHA` message.
4) The client displays the received CAPTCHA to the user, records her response, prompts her for her password, and sends a `SSH_MSG_USERAUTH_CAPTCHA_RESPONSE` message containing the response and password.
5) On receipt, if both the CAPTCHA response and password are correct, the server sends a `SSH_MSG_USERAUTH_SUCCESS` message. Otherwise, it sends a `SSH_MSG_USERAUTH_FAILURE`. In the case of success, the authentication protocol is complete. Otherwise, the client is free to re-attempt authentication until the server decides to close the connection.

IV. Theoretical Result

We instantiate a problem from the class $P2_{\mathcal{I},\mathcal{T},\lambda}$ of [2] and reduce the security of the TGC CAPTCHA to the hardness of the problem. We describe roughly the definition of $P2_{\mathcal{I},\mathcal{T},\lambda}$ and some key terms. See [2] for more precise definitions.

The family of AI problems $P2_{\mathcal{I},\mathcal{T},\lambda}$ is indexed by the distributions of images $\mathcal{I}$ and $\mathcal{T}$ and by the solution λ which is a map of images to their corresponding labels. Intuitively, it is the following problem: given a transformed image $t(i)$ where $i \in \mathcal{I}$ and $t \in \mathcal{T}$, find the label $\lambda(i)$. A problem is (δ, τ)-*hard* if no current program can solve it with probability at least δ in time at most τ. A test is (α, β)-*human executable* if at least an α portion of the human population can pass it with at least β success probability. A (α, β, η)-*CAPTCHA* is a test that is (α, β)-human executable and has the property that, if an algorithm B passes it with a success probability of at least η, then B can be used to solve a hard AI problem.

Theorem 1: Let k be the security parameter, and let $\mathcal{I}, \mathcal{T}, \lambda$ be as previously defined in Section II. Let $\delta, \tau, \alpha, \beta$ be non-

negative real numbers. Assume that $\mathsf{TGC} = (\mathcal{I}, \mathcal{T}, \lambda, \tau, k)$ is (α, β)-human executable. If $P2_{\mathcal{I},\mathcal{T},\lambda}$ is $(\delta, \tau + O(k))$-hard, then TGC is a (α, β, δ)-CAPTCHA.

Proof: Given an algorithm B that has a success probability of at least δ over TGC in time τ, we construct an algorithm A that is a $(\delta, \tau + O(k))$ solution to $P2_{\mathcal{I},\mathcal{T},\lambda}$. This proves the theorem.

On input a transformed image t^*, the algorithm A simply draws $g \xleftarrow{\$} \{1, \ldots, k\}$ and sets $w_g = t^*$. Then, for all j such that $1 \leq j \leq k$ and $j \neq g$, it draws $i_j \xleftarrow{\$} \mathcal{I}$ and $t_j \xleftarrow{\$} \mathcal{T}$, then sets $w_j = t_j(i_j)$. Next, it submits $w_1, \ldots, w_k$ to B and obtains the answers $l_1, \ldots, l_k$ from B. Finally, it outputs l_g as its answer.

It is easy to see that A simulates B perfectly. Also, A runs in time taken by B plus time linear in k. Finally, if B has the probability of δ of getting all of its answers right, then A's probability of success is at least δ. ∎

V. Experimental Results

We performed two experiments to assess the difficulty of the TGC CAPTCHA for humans (Experiment 1) and machines (Experiment 2).

Experiment 1. Twelve naive subjects were recruited from the faculty and staff of Thammasat University. Each subject, in a session lasting approximately 5 minutes, responded to 12 TGC CAPTCHAs (two practice trials and 10 test trials) using the same parameters as the SSH prototype ($k = 8, n_c = 80, n_r = 24$). The CAPTCHAs were displayed in standard terminal windows on PCs running Linux or Mac OS X.

The subjects' average per-character accuracy p_h on the test trials was 0.948. Their average word-level accuracy (the number of 8-letter TGC CAPTCHAs answered with 100% accuracy) was 0.708. (Assuming independence and $p_h = 0.948$, we would expect a word-level accuracy of $(p_h)^k = 0.652$.)

The fact that naive users achieve such high accuracy rates justifies the use of TGC CAPTCHAs in live systems. Frequent users would very rapidly adapt to the statistics of the character set, achieving much higher accuracy rates.

Experiment 2. We tested the open source Optical Character Recognition program GOCR [6] on our TGC CAPTCHA. Using the same parameters as Experiment 1, we generated 100 TGC CAPTCHAs and converted each of the resulting text-graphics screens to bitmaps. We then ran GOCR 0.39 in its default configuration on the resulting 800 images. We call this the *Naive GOCR* adversary to emphasize that different configurations could in principle yield better adversaries.

Unlike our human subjects, Naive GOCR is not constrained to respond with exactly one English letter for each English letter of the CAPTCHA. We therefore graded the OCR system using two criteria. As a conservative criterion, we judged a response correct if it contained the expected letter *and no other letters*. As a less conservative criterion, we judged a response correct as long as it contained the expected letter.

Naive GOCR had a per-character accuracy p_m of 0.241 and 0.329 by the first (conservative) and second (loose) criterion, respectively. The word-level accuracy was 0 by both criteria.

We believe p_m could be significantly improved by incorporating problem-specific knowledge of the TGC CAPTCHA algorithm. We leave this as an exercise for interested readers. Clearly, however, the results demonstrate that Naive GOCR is unsuitable for mounting dictionary attacks against TGC CAPTCHA-enabled, password authenticated systems.

VI. Conclusion

The TGC CAPTCHA shows promise as a construct for improving the security of password authenticated systems like SSH. We have shown that the test is relatively easy for humans but would be difficult for "Naive GOCR" adversaries. Of course, this only puts an *upper bound* on the difficulty of the problem. We have not proven that no adversary can do better. (After all, certain AI pundits believe that ALL tasks humans can perform today will be performed equally well by machines in the not-so-distant future!)

In any case, clearly, the security of a CAPTCHA password authenticated system increases as the gap between human and machine performance on the test widens. The TGC CAPTCHA can thus serve as a modest cross-disciplinary challenge in the fields of pattern recognition, system security, computer graphics, and even psychology. This is the approach championed by [2]. Through friendly competition, we hope to encourage not only new OCR algorithms, but also a better understanding of the strengths and weaknesses of the human visual system relative to the best present-day machine vision systems.

Acknowledgments

We would like to thank the faculty and staff of the Faculty of Engineering and Sirindhorn International Institute of Technology, Thammasat University for their help with Experiment 1.

References

[1] B. Pinkas and T. Sander, "Securing passwords against dictionary attacks," in *Proc. of the 9th CCS*, V. Atluri, Ed. ACM Press, Nov. 2002, pp. 161–170.

[2] L. von Ahn, M. Blum, N. J. Hopper, and J. Langford, "CAPTCHA: Using hard AI problems for security," in *EUROCRYPT 2003*, ser. LNCS, E. Biham, Ed., vol. 2656. Springer-Verlag, May 2003, pp. 294–311.

[3] The CAPTCHA Project, `http://www.captcha.net`.

[4] M. Bellare, "Practice-oriented provable security," in *Lectures on Data Security: Modern Cryptology in Theory and Practice*, ser. LNCS, I. Damgård, Ed., vol. 1561. Springer-Verlag, 1998, pp. 1–15.

[5] M. Dailey and C. Namprempre, `http://www.siit.tu.ac.th/mdailey/captcha/`.

[6] J. Schulenburg *et al.*, "GOCR: open-source character recognition, version 0.39," 2004, available at `http://jocr.sourceforge.net/index.html`.

[7] T. Ylonen, "SSH authentication protocol," 2002, draft 18, available at `http://www.ietf.org/html.charters/secsh-charter.html`.

[8] C. Dwork and M. Naor, "Pricing via processing or combatting junk mail," in *CRYPTO 1992*, ser. LNCS, E. Brickell, Ed., vol. 740. Springer-Verlag, Aug. 1992, pp. 139–147.

[9] J. Xu, R. Lipton, I. Essa, M. Sung, and Y. Zhu, "Mandatory human participation: A new authentication scheme for building secure systems," in *Twelfth International Conference on Computer Communications and Networks*, IEEE, Ed. IEEE Press, Oct. 2003.

[10] T. Ylonen, T. Kivinen, M. Saarinen, T. Rinne, and S. Lehtinen, "SSH transport layer protocol," 2003, draft 17, available at `http://www.ietf.org/html.charters/secsh-charter.html`.