

Mandatory Human Participation: A New Authentication Scheme for Building Secure Systems

Jun Xu, Richard Lipton, Irfan Essa, Minh Sung, Yong Zhu
College of Computing
Georgia Institute of Technology
Atlanta, GA 30332-0280
{jx,rjl,irfan,mhsung,yongzhu}@cc.gatech.edu

Abstract—Mandatory Human Participation (MHP) is a novel authentication scheme that asks the question “are you human?” (instead of “who are you?”), and upon the correct answer to this question, can prove a principal to be a human being instead of a computer program. MHP helps solve old and new problems in computer security that existing security measures can not address properly, including password (or PIN number) guessing attacks and application-level denial of service. A key component of this “are you human?” authentication process is a character morphing algorithm that transforms a character string into its graphical form in such a way that a human being won’t have any problem recognizing the original string, while a computer program (e.g., an Optical Character Recognition program), will not be able to decipher it or make a correct guess with non-negligible probability. The basic idea of the MHP scheme is to ask an agent to recognize the string before its login attempts or transaction requests can be honored. Here a protocol is needed to send a puzzle to an agent, check if the answer supplied by the agent is correct, and most importantly make sure that the agent can not cheat in the process. A number of system and security issues that relate to the protocol need to be addressed for the protocol to be secure, efficient, robust, and user-friendly. The MHP scheme contributes to the foundation of the computer security by faithfully implementing a novel security semantics, “human,” which existing cryptographic measures can not express accurately. As many real-world security applications involve the interaction between a human and a computer, which naturally contains “human” as a part of its protocol semantics, we believe that the MHP scheme will find many new applications in the future.

I. INTRODUCTION

Automatic attacks using computer programs are the central threat to computer security. For example, the cheapest home PC running a brute-force password (or PIN number) guessing program can generate and try tens of thousands of candidate passwords each second. While current security measures aim at improving the robustness of a system under such automatic attacks with limited success (e.g., advocating the use of random nonguessable password), we attack the problem more effectively from a new angle. Our approach would disable the automation part of any such attacks. Under this approach, to try to log in a remote computer by guessing passwords, the intruder would have to type in each and every candidate password using his/her own fingers. Obviously, few people would have the time and patience to do that.

Our approach is based on the following observation. In an application that involves the interaction between a computer system and a human being, if we were able to distinguish between two classes of transactions: those that were generated automatically (by a computer program) and those that were

generated manually (by a human), any automated attack would fail as long as we set the security policy to allow only transactions that are from humans. But, first of all, how to tell the difference between these two types of transactions?

The proposed solution to the problem is inspired by Turing’s test [1] for artificial intelligence. The fundamental idea of the solution is for a computer system to first ask the author of every transaction to solve a puzzle before accepting or executing the transaction. The content of the puzzle will be based on grand challenge problems in the domains of pattern recognition, visual interpretation, and natural language understanding. These problems have the essential property that people can solve them easily while computers are not likely to solve them in the foreseeable future. A typical puzzle would consist of the computer system sending the agent a bit-mapped image and the agent replying with an ASCII string. The image might include a picture and a question about that picture, such as “Please type the following handwritten word” or “Which of the objects in this picture are edible?” The computer system determines whether or not the transaction author is human based on the answer supplied.

This puzzle-solving process leads to a new scheme for building secure computer systems. In this scheme, a human has to be directly involved (by solving the puzzle and typing in the answer) in the authentication or other processes that are vulnerable to automatic attacks, referred to as Mandatory Human Participation (MHP). Apparently, no automatic attack to a protected process would be possible under this scheme. In our login example, if the agent who tries to log in is indeed a human who is authorized to access the account, then logging in is only slightly more onerous than it currently is. However, if the agent is a password (or PIN) guessing algorithm, then it would not get to try even a single password (or PIN) as it can not solve a single puzzle!

In contrast, traditional countermeasures against password (or PIN) guessing are generally less effective in the Internet environment. For example, the good old way to counter PIN number guessing attack on ATM machines is to confiscate or lock the card after a few failed trials [2]. The counterpart of this confiscation in online banking is to freeze an account after a few failed attempts to the account. However, this would lead to denial of service. Given that bank account numbers are quite predictable (e.g., contiguous) [3], it is very easy for an intruder to freeze thousands of accounts in a matter of minutes with a brute-force attack. Protection based on IP address is not quite effective in solving this problem because this can be bypassed

by IP spoofing [4] and/or attacks can be launched from a large number of compromised hosts simultaneously. It is not clear whether other measures such as introducing delay into a transaction can solve the dilemma between broken accounts and denial of service. The MHP scheme can also be used to solve application-level Denial of Service (DoS) problem. This application will be described in Section II.

Although any grand challenge problem may be used to distinguish transactions submitted by a human from those submitted by a computer program, we choose character recognition, a subclass of pattern recognition, to implement the MHP scheme. The reason is that, for practical use in most real-world applications, the grand challenge problem used by the MHP scheme needs to satisfy some security and system requirements (discussed in Section III). By far, character recognition is the only grand challenge problem we have discovered that would readily satisfy these requirements. MHP scheme based on character recognition would be asking the agent to recover the original ASCII string, given the string in its transformed graphical form. The resulting image needs to have the following two properties. First, it must be easy for a human being to correctly recover the string from the graphical image. If it requires a lot of time, no one would be very happy deciphering the image. Second, it must be hard for a computer to do the same thing. The computer should not be able to make a correct guess with high probability, or is able to do so only after a lot of computation.

We designed a character morphing algorithm to generate character string images that would satisfy these two properties. It applies a set of morphing and distortion primitives such as font, size, style, rotation, scaling, sheaving, noise, and 3-D shadow on the characters with random parameters in an effort to make it still easy for humans to recognize, but inordinately difficult for computers. Machine interpretation of the text presented in a graphical form is possible by using Optical Character Recognition (OCR), which is a well-developed and at present easily available technology [5]. However, all efforts in this area (for example [6], [7]) have been aimed at recognizing text image in a fairly regular form such as those that appear in a document. Very accurate and reliable OCR for random text presented in such a morphed and distorted form is still considered a very difficult task for machines. We conduct a comprehensive survey of existing OCR algorithms which shows that our morphing algorithm poses tremendous challenges for them to recognize morphed images with reasonable accuracy. These survey and results are omitted here due to lack of space and can be found in our recent technical report [8]. We implemented the baseline version of the character morphing primitives and used them alone or in composition to generate a set of test samples to be recognized by off-the-shelf OCR software. We show that most of these samples indeed confuse off-the-shelf OCR without affecting the usability, i.e., readability to humans.

In addition to the morphing algorithm that generates the character recognition puzzles, the MHP scheme also contains a protocol to send puzzles to agents, check if the answers submitted by the agents are correct, and most importantly make sure that no client can cheat in the process. As we will show in Section V, a set of nontrivial system and security

issues need to be addressed in the design of the protocol, e.g., how to resolve the apparent conflict between the need to counter replays (by computer programs) of previously solved (by humans) <puzzle, answer> pairs and the need to make the system stateless for better flexibility, efficiency, and robustness. We developed techniques to address these issues and integrated them into our prototype web-based protocol implementation.

The rest of the paper is organized as follows. Section II presents more applications of the MHP scheme. Section III explains why character recognition is chosen as the grand challenge problem for the MHP scheme. Section IV presents the design, implementation, and evaluation of the character morphing algorithm in detail. Section V discusses the protocol used to administer the puzzle-solving process. Section VI discusses related works. Section VII concludes the paper.

II. MORE APPLICATIONS OF THE MHP SCHEME

In addition to countering password (or PIN) guessing attacks, an MHP scheme can also be used to stop Denial of Service at the application level. Denial of Service (DoS) is one of the most challenging problems in computer security. The MHP scheme is very effective against DoS at the application layer. We recently discovered a new security vulnerability. If this vulnerability is not properly addressed, it could lead to the next wave of DoS attacks, the damage of which can be more severe and long-lasting than its current versions (e.g., SYN flood). We illustrate this vulnerability with the following example. At the time this paper was first written [9], to establish a free email account at yahoo.com, a user only needs to come up with a previously unused username and a password for the account, and fill in a brief survey¹. This process can be automated using random <username, password> pairs and random answers to the survey to establish hundreds of thousands of bogus email accounts a day. After these bogus accounts are created, another automated process can then stuff terabytes of bogus email into these accounts. Moreover, it is very hard for Yahoo to automatically remove these bogus accounts since they can be made to look just like the regular accounts, e.g., even time of the most recent access can be faked by an automated email “touch” process. So the damage caused by this type of DoS is not just minutes or hours of no service (Imagine the situation in which millions of good accounts need to be manually separated from billions of bad accounts). As a matter of fact, many web sites allow this type of new account registration. Clearly, the MHP scheme would effectively disable this automated process if every user has to solve a puzzle to be able to create a new account. Moreover, the MHP scheme is especially effective against distributed forms of such attacks in which the intruder replicates attack programs on hundreds of victimized machines and launch attacks in parallel. Since the replicated programs are not humans and thus not being able to solve the MHP puzzles, the MHP scheme strikes the Achilles’ heel of the intruder: he can duplicate his programs and data, but not his human character recognition skills.

¹Yahoo.com has independently designed and implemented similar schemes in 2001, as discussed in Section VI.

In the above example, service should be granted only to humans, but this implicit “human” semantics is generally not well addressed by existing security mechanisms, thus leading to security vulnerabilities. The MHP scheme effectively addresses these vulnerabilities by faithfully implementing this semantics. Since many real-world security applications involve the interaction between a human and a computer, which naturally contains “human” as a part of its protocol semantics, we believe that we will find many more applications of the MHP scheme in the future.

III. MOTIVATION FOR CHOOSING CHARACTER RECOGNITION

For practical use in most real-world applications, the MHP scheme needs to satisfy the following key system and security requirements:

- 1) In real-world applications where hundreds of transactions may have to be processed per second, only a computer is fast enough to grade the answers to the puzzles. So the answer needs to be unambiguous or among one of very few choices.
- 2) For most applications, the set of possible answers has to be large and unpredictable. Otherwise, guessing from the possible answer set will be quite effective against the MHP scheme.
- 3) There should be a systematic way for the MHP scheme to automatically generate fresh puzzles. The puzzles have to be fresh: having a puzzle database generally won't work as an intruder would potentially obtain this database and then would be prepared for every puzzle in the database. The process has to be automated: hundreds of fresh puzzles will have to be given out per second and no human will be able to design fresh puzzles that fast.

The character recognition puzzles are chosen because they readily satisfy these requirements. In the character recognition problem, the answer can easily be made unambiguous (e.g., by allowing case insensitivity) and the space of potential answers can be huge even with a string as long as ten characters. Also, fresh images can be generated automatically and efficiently, which will be described in the next section.

In contrast, it is not trivial to engage other grand challenge problems for the MHP scheme. For one example, natural language understanding in the context of Turing test does not appear to be a suitable problem for the following reasons. First, it is generally very hard, if not impossible, for a computer program to generate a large number of fresh yet meaningful questions. Also, with Turing test questions, there typically can be many possible answers. While a human would have no problem telling whether the answer appears to be from another human, it is in general hard for a computer to do so. While special classes of puzzles such as “Another word for ‘canine’” or “The opposite of ‘cold’” are easy to generate and easy to grade using a lexicon, they are not hard for a computer program to answer either.

IV. THE ALGORITHM: DESIGN, IMPLEMENTATION, AND EVALUATION

In this section, we present the design and implementation of the character morphing algorithm, and discuss how we validate

its strength against OCR algorithms. Evaluation of its usability and performance aspects is also discussed.

A. Design of the character morphing algorithm

The objective of the character morphing algorithm is to transform a text string into its graphical form that is easy for humans, but hard for OCR programs, to recognize. To achieve this goal, the algorithm generates a string of random characters embedded in a graphic image using one or more of the following morphing primitives:

- 1) The spaces between the characters may have been removed, and non-uniform spaces might exist between other characters. The letters may even overlap slightly with each other to the extent that is still recognizable.
- 2) Some characters, randomly, may be stretched and others compressed so that characters are of a non-uniform width and height. The stretching or compression can also happen in a nonuniform fashion on a single character. For example, the factor of stretching in X coordinate can be a decreasing function of the Y coordinate so that the top of the letter is stretched more than the bottom of the letter.
- 3) Characters can be sheared (a rectangular morphed into a parallelogram) to the left or to the right.
- 4) Characters may be composed of different fonts, styles, and sizes.
- 5) Some characters may be rotated to the left or right, and the characters may be wrapped on a cubic spline or presented in a non-linear fashion.
- 6) Characters may be randomly “filled.”
- 7) Characters are presented in a graphic bitmapped image, but randomly located within the image. Other areas of the image may contain squiggles, curly-cues, circles, arcs, and various other nonsensical writings or symbols which are not recognizable as characters, and thus will not be confusing to humans, but may to a machine.
- 8) Some noise may randomly be added to the graphic image.
- 9) A lossy image compression like JPEG can be used to reduce the quality of the image to make the OCR decision process more susceptible to errors [7] while making no difference to human decision process.
- 10) The letters can have 3-D shadows beside them.
- 11) The letters can be based on one of many handwriting samples (written by many different people), and then morph them according to above techniques.

These primitives may be used in composition. For example, a character ‘A’ can first be assigned Helvetica font, size 12, and bold style, and then sheared to the right by fifteen degree, and then rotated to the left by twenty degree. Also, each character in the string can be morphed independently of others, i.e., each character may use any reasonable combination of morphing primitives with any allowable parameters.

B. Empirical effectiveness of the algorithm

In this section, we demonstrate the empirical effectiveness of character morphing algorithm against “Off-the-shelf” OCR programs. We implemented the baseline version (described in Section IV-C) of the proposed morphing primitives and

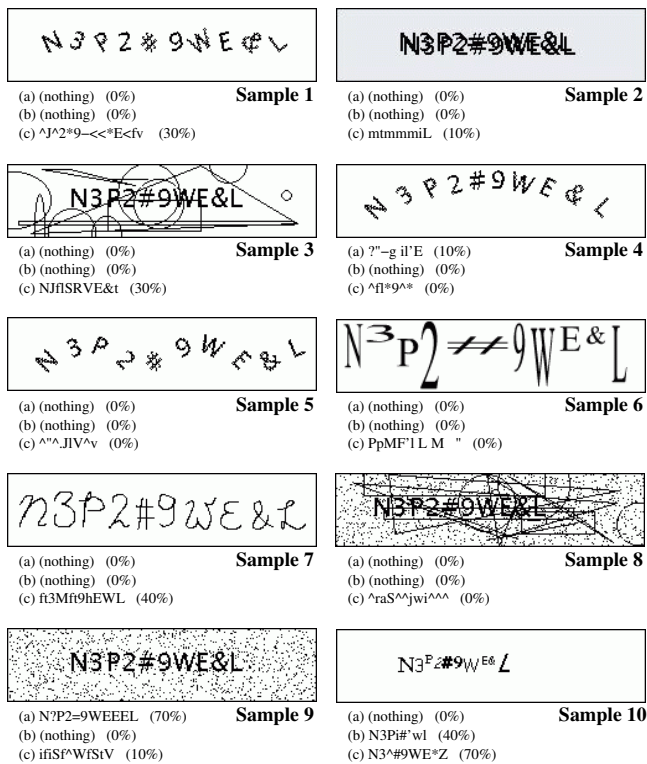


Fig. 1. Samples and Recognition Results: (a) CuneiForm (b) SunmiPage (c) ABBYY

used them alone or in composition to generate a set of test samples to be recognized by three off-the-shelf OCR software packages, namely, (a) CuneiForm'99 by Cognitive Technology, (b) SunmiPage ScanInsert 1.0 by ComputerTek Enterprises, and (c) ABBYY Fine Reader 6.0 by Optical Character Recognition System. In Fig. 1 we present these samples and show the contents and percentage of the characters these software packages managed to recover.

Except perhaps the sample 8 in Fig. 1, all these samples are easily readable by humans. Most of these samples indeed confuse off-the-shelf OCR software: eight among them are able to keep the best recognition percentage (per character) below 30%, which is translated into an accuracy of less than $6 \cdot 10^{-6}$ with a string of ten letters; with some samples (sample 5, 6, and 8) no OCR software was able to even recognize a single character correctly.

This hardly comes as a surprise since off-the-shelf OCR software is tuned to recognize regular texts such as those that can be found in a document. To conclusively validate the effectiveness of the morphing primitives, we have yet to analyze all known types of OCR algorithms to see whether they can be modified and/or tuned to decipher the morphed images. Through this study, we will be able to minimize the vulnerability of the transformation algorithm by identifying, refining, and sticking to most effective compositions of primitives and choice of parameters, and if necessary, introducing new and more effective primitives. Comprehensive survey of the state-of-art OCR algorithms can be found in our recent technical report [8].

C. Implementation and evaluation

We implemented the baseline version of the character morphing algorithm. Its current manual version takes a description file as its input, and generates an image file in the BMP format as its output. The first part of the description file specifies how the characters in a string should be aligned/located on the canvas. The relative alignment of the letters could be manually specified using its relative location in the image, or automatically generated using built-in functions (with default parameters) such as waveform or spline. The second part of the file specifies how each letter is morphed. Each line specifies a character and the primitives and parameters to be applied on it and the sequence they are applied. For example, “char_morph -stretch 0.8 -sheave 15 -rotate -30 a” command generates letter ‘a’ with default font, size, and style (TimesRoman, 12, Normal), then compresses its height to 80% of its original, sheaves it 15 degrees to right, and rotates it 30 degrees to left. The font, size, and style could be set explicitly using options such as “-font helvetica.” The third part, optionally, specifies the type and intensity of the noise introduced to the picture. Currently, we have implemented the “dot noise” (just black dots) and “object noise” (triangles, circles, rectangulars, etc.). In the dot noise, the location of the dots is random and its intensity is determined by a tunable parameter. In the object noise, both the type of the object and their relative locations can be random.

Currently only this interactive/manual version of the algorithm is implemented. After we have found the set of most effective compositions of primitives and the corresponding set of effective parameter ranges, we will implement its automatic version that can pick a composition of primitives randomly from the former set, obtain a set of random parameters according to the ranges specified in the latter set, and generate a random string image using the chosen compositions and parameters. So far, we have obtained a small set of “good compositions” that can confuse OCR algorithms without causing much usability (i.e., readability to humans) problems, and parameter ranges corresponding to these compositions.

Performance of the morphing algorithm is a major concern. We measured the amount of time it takes each primitive to transform a ten-character-long random string into a 250 by 60 bitmap image (24 bits each pixel) on a 733 MHz Pentium III. The result is the following: 213 microseconds for normal letters, 761 microseconds for rotation, 337 microseconds for sheaving, 313 microseconds for stretching/compression, 383 microseconds for introducing symbols (e.g., triangles, circles, etc.), 769 microseconds for arranging the letters into the banner wave. As with different parameters these primitives may take different amount of time, all data were obtained by averaging 10000 random instances. We also found that composition of the primitives typically does not take longer time to compute than their individual constituents combined because some common operations only need to be executed once. We measured that a typical composition of three to four primitives would take less than 1500 microseconds to execute. Note all these performance data do not include file I/O time. We measured that it takes an average of 340 microseconds to save a 45 KB (250*60*24 bits) image file to disk and this I/O typically can concur with the morphing process since the latter

is the performance bottleneck. So we expect that an image can be generated well within 2000 microseconds. One dedicated puzzle server should be able to generate more than 500 puzzles per second, fast enough for even the most popular websites like yahoo.com.

V. THE PUZZLE-SOLVING ADMINISTRATION PROTOCOL

Making sure that the morphing algorithm is hard to break by OCR algorithms is the necessary but not the sufficient condition for the MHP scheme to work well in real-world applications. A protocol is needed to send a puzzle to an agent, check if the answer supplied by the agent is correct, and most importantly make sure that the agent can not cheat in the process. In the following, we discuss a puzzle-solving administration protocol that we have designed and implemented.

In this protocol, there are two key technical issues to be addressed:

- 1) How does server check whether the client's answer is correct?
- 2) How replay of a previous challenge/answer pair is prevented?

A simple and obvious answer to check the correctness of the answer is to remember answers to all the puzzles that are given out. However, this stateful approach would require the storage of state information for every outstanding puzzle including the serial number that identifies a puzzle and the answer to the puzzle. This would result in a large amount of state information to be kept on a busy server. A better alternative is a stateless (for the server) approach in which whether the puzzle is correctly solved or not can be directly inferred from the client's answer and the encrypted state information kept by the client. There are situations where the *stateless* approach would be ideal. For example, in E-commerce applications, one server may be responsible for generating puzzles while other servers process the transactions. So the server that gives out a puzzle may not be the server that checks if the answer is correct.

The way to make the server stateless is as follows. When a puzzle is given out, the correct answer (the string) to the puzzle, along with the information that uniquely identifies the puzzle such as serial number and timestamp will be hashed into a Message Authentication Code (MAC) using a secret key (shared among all transaction servers in the aforementioned multi-server case). We use MAC instead of digital signature [2] since the latter is order of magnitude slower. This MAC will be a part of the puzzle to be given to the client and should be returned to the server along with the transaction request. In this way, the client keeps the state (the MAC) so that the server does not need to keep them. The client can not tamper with the MAC because it is encrypted using the secret key of the server. To check if the puzzle is correctly solved, the server only needs to check if the message authentication code computed from the client's answer matches the MAC received.

Now we are ready to describe our generic humanizer protocol. Here we assume that a single server processes all the transactions. As mentioned above, the stateless approach makes the multi-server design easier. Each puzzle $h(STR)$ is generated from a random string STR using the Turing-resistant hash function h . The puzzle is uniquely identified by

a serial number SRN , which increases by one every time a new puzzle is given out (no wraparound). The server uses a secret key k and a keyed cryptographic hash function F such as HMAC-MD5 [10], [11] to generate the MAC . A timestamp TMP is associated with every puzzle. The maximum lifespan of a puzzle is MLS . An answer to a puzzle that is older than MLS will no longer be honored. The current time is denoted by T . The protocol between a client and a server is executed as follows:

- 1) The client sends an HTTP request to the server for the CGI form DOC .
- 2) The server generates a random string STR , computes $h(STR)$ and $MAC = F_k(SRN||TMP||STR)$ and sends DOC , SRN , TMP , MAC , and $h(STR)$ to the client.
- 3) The client recovers STR' (may not equal to STR if the client is not human) from $h(STR)$ and sends DOC' (with fields filled out), SRN , TMP , MAC , and STR' back to the server.
- 4) The server rejects the query and notifies the client if any of the following happens: (1) $T - TMP > MLS$, which indicates that the puzzle has expired, (2) $F_k(SRN||TMP||STR')$ does not match MAC , which indicates that the puzzle has not been correctly answered, and (3) the answer is a replay, the detail of which will be discussed next. Otherwise, the server executes the query and sends the result to the client.

Now we need to address the problem of the replay of a solved puzzle. To counter replay, the server needs to remember which puzzle has been solved. The server maintains a bit array $A[0..N-1]$ of N entries. N should be made comfortably larger than maximum number of puzzles that are given out during any period of length MLS (the maximum life span of a puzzle). In protocol step two, when a puzzle with serial number $SRN1$ is given out, $A[SRN1 \bmod N]$ will be set to 0, indicating that it has yet to be answered. In protocol step four, when a puzzle with serial number $SRN2$ is received, the answer will be rejected if $A[SRN2 \bmod N]$ is 1, indicating that it has been answered. Otherwise, if the answer is honored, $A[SRN2 \bmod N]$ will be set to 1. It is not hard to see that the way N is chosen prevents the "wraparound" ambiguity from happening.

We recommend HMAC-MD5-32 (output truncated to 32 bits) [10], [11] for generating the message authentication code because its (HMAC-MD5) properties are carefully validated and it can be computed very fast. However, other keyed cryptographic hash functions [12] with similar properties may also be used. It is reported in [13] that using Dai's Crypto++ 4.0 library, a 850 MHz Intel Celeron processor can generate HMAC-MD5 for about one hundred million bytes of preimage per second. This speed is translated into generating MACs for millions of puzzles per second, which is fast enough for even a high-end transaction server. We truncate the output to 32-bit instead of the 80-bit recommended minimum in [11] because (1) the birth day attack, which compromises nonrepudiation, is not an issue in this protocol, (2) 32 bit still makes the probability of "guess it right" negligible, and (3) this makes the MAC key harder to guess, as shown in [12].

As a security measure, please enter the characters you see in the box on the left into the box on the right. (The characters are not case sensitive.) [Help?](#)



Fig. 2. Dialog box used by paypal.com

VI. RELATED WORK

The idea of the Mandatory Human Participation (MHP) is inspired by Turing's test for artificial intelligence [1]. The Turing test asks whether it is possible for a computer to fool a human into thinking that it is a human. The Turing test is usually formulated in the context of natural language understanding, in which a human asks a machine questions to see if it can always come up with coherent, logical, and meaningful answers. However, Turing's test in its present form can not be readily applied to computer security, as shown in Section III.

An idea similar to the MHP scheme, which was briefly described in an unpublished manuscript [14], was brought to our attention. However, among the types of puzzles suggested in [14], some are no longer grand challenge problems (e.g., gender recognition, facial recognition, and speech recognition), and none of them in its present form may be able to satisfy constraints imposed by the practical considerations, as described in Section III. Except this, we are not aware of any other similar ideas being presented in the research literature.

Concurrent with our work [9], Paypal.com and Yahoo.com have implemented similar schemes on their websites. For example, a required step for registering a new account with `paypal.com`, is for a user to copy a few characters from a JPG image into a dialog box (see Fig. 2). The intent here is similar to what we are proposing: only a human user will be able to do something like this, while a computerized system will fail. However, images shown in Fig. 2 is not hard to recognize, since "integration" on the amount of black pixels vertically and horizontally can be used as a feature to systematically remove the straight lines that cross these letters. Also concurrent with our work, CAPTCHA project [15] have been proceeding by Ahn et al in Carnegie Mellon University.

Juels and Brainard's "client puzzles" [16] for countering a generalization of TCP SYN flood attack bears some similarity to our approach on the denial of service problem. In their scheme, a client also has to solve a puzzle, which may take about a couple of seconds, before its connection request is honored. The difference is that they use computational puzzles that aim to consume a client's CPU resource while we use "are you human?" puzzles. The weakness of using computational puzzles lies in the fact that the speeds of computers can differ by orders of magnitude. If the slowest client (say on a 386) has to be able to solve the puzzle in a few seconds, an intruder with a high-end workstation or even a cluster will be able to solve the puzzle in a few milliseconds and will still be able to cause denial of service. From the resource allocation point of view, our approach taxes the ultimate resource of an intruder, the speed he can perform I/O as a human.

VII. CONCLUSIONS

The key contributions of this research can be summarized as follows:

- 1) We identify security vulnerabilities caused by the lack of security mechanism to enforce the "human" semantics, some of which are not previously identified or publicized (e.g., Yahoo email DoS) and could be catastrophic if they are not dealt with. We show that these vulnerabilities are well addressed by the MHP scheme.
- 2) After identifying practical constraints for the MHP scheme to be useful in real-world applications, we introduce a concrete and feasible mechanism (character morphing algorithm) to implement the MHP scheme under these constraints. We have implemented the baseline version of this mechanism, validated and studied its strength against OCR programs, and evaluated its performance and usability aspects.
- 3) We introduce a protocol to oversee the puzzle-solving process, which is an integral part of the MHP scheme. System issues in the design of the protocol such as flexibility, performance, and robustness, are also addressed.
- 4) The scheme is completely compatible with all existing protocols and their implementation base. In particular, it does not entail any modification to web browser software.

VIII. ACKNOWLEDGEMENTS

The work was supported in part by the National Science Foundation under grant ITR/SY 0113933.

REFERENCES

- [1] A. Turing, "Computing machinery and intelligence," *Mind*, pp. 433–460, 1950.
- [2] A. Menezes, P. Oorschot, and S. Vanstone, *Handbook of Applied Cryptography*. CRC Press, 1996.
- [3] A. D. Santos, G. Vigna, and R. Kemmerer, "Security testing of the online banking service of a large international bank," in *Proceedings of the First Workshop on Security and Privacy in E-commerce*, Nov. 2000.
- [4] S. Bellovin, "Security problems in the tcp/ip protocol suite," *ACM Computer Communication Review*, vol. 19, no. 2, 1990.
- [5] "Document and image understanding image server," <http://documents.cfar.umd.edu>.
- [6] S. Haigh, "Optical character recognition (ocr) as a digitization technology," *Network Notes* 37, 1996.
- [7] S. Srihari, "High-performance reading machines," *Proceedings of the IEEE*, vol. 80, no. 7, pp. 1120–1132, July 1992.
- [8] J. Xu, R. Lipton, I. Essa, M. Sung, and Y. Zhu, "Mandatory human participation: A new authentication scheme for building secure systems," GIT-CC-03-36, Georgia Institute of Technology, Tech. Rep., July 2003.
- [9] J. Xu, R. Lipton, and I. Essa, "Hello, are you human," Technical Report GIT-CC-00-28, Georgia Institute of Technology, Tech. Rep., Nov. 2000.
- [10] M. Ballare, R. Canetti, and H. Krawczyk, "Keyed hash functions and message authentication," in *Proceedings of Crypto'96*, 1996, pp. 1–15.
- [11] H. Krawczyk, M. Bellare, and R. Canetti, *RFC2104: HMAC: Keyed-Hashing for Message Authentication*, RFC 2104, Network Working Group, Feb. 1997.
- [12] B. Preneel and P. Oorschot, "Building fast macs from hash functions," in *Proceedings of Crypto'95*. Springer-Verlag, 1995, pp. 1–14.
- [13] W. Dai, *Crypto++ 4.0 Benchmarks*, June 2000.
- [14] M. Naor, "Verification of a human in the loop or identification via the turing test," http://www.wisdom.weizmann.ac.il/naor/PAPERS/human_abs.html, Sept. 1996.
- [15] "The captcha project," Carnegie Mellon University, <http://www.captcha.net>.
- [16] A. Juels and J. Brainard, "Client puzzles: A cryptographic countermeasure against connection depletion attacks," in *Proc. of NDSS'99*. Internet Society, Mar. 1999.