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

Jun Xu, Richard Lipton, Irfan Essa, Min-Ho Sung
College of Computing
Georgia Institute of Technology
Atlanta, GA 30332-0280
{jx,rjl,irfan,mhsung}@cc.gatech.edu
Technical Report (GIT-CC-01-09)

May 11, 2001

**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, automated service and information theft, and 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.

# 1 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 hacker 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 [Tur50] 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 [MOV96]. 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) [SVK00], it is very easy for a hacker 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 [Bel90] 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.

In addition to countering password (or PIN) guessing attacks, we discover that the MHP scheme can also be used to solve other classes of security problems, namely, Denial of Service (DoS) and automatic service and information theft. These applications will be described in Section 2.

Although any grand challenge problems 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 3). 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 [UMa]. However, all efforts in this area (for example [Hai96, Sri92]) 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 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. We also study the techniques to strengthen the morphing algorithm against variants of known OCR algorithms specially tuned to recognize the morphed image.

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 3, 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 2 presents more applications of the MHP scheme. Section 3 explains why character recognition is chosen as the grand challenge problem for the MHP scheme. Section 4 presents the design, implementation, and evaluation of the character morphing algorithm in detail. Section 5 discusses the protocol used to administer the puzzle-solving process. Section 6 discusses related works. Section 7 concludes the paper.

## 2 More Applications of the MHP Scheme

In addition to countering password (or PIN) guessing attacks, an MHP scheme can also be used to solve other classes of security problems, namely, Denial of Service (DoS) and automatic service and information theft. In the following, we show examples from both classes of problems:

**Blocking automatic service and information theft:** The MHP scheme can effectively block automatic service and information theft, as shown in the following example. Bob, who owns an online store that sells the same products as Alice's, can systematically steal information from Alice's web server designed to allow potential human customers search for product pricing and availability as follows. Bob can write a computer program sending CGI (or other appropriate formats) queries to Alice's server, looking up price information of every single item, extracting the price from the query results, and making his price on the same item slightly cheaper than hers. The MHP scheme would effectively stop this theft: requiring Bob to solve a puzzle for every transaction would limit the information or service theft to how fast he can solve the puzzles and type in the answers.

**Stopping Denial of Service:** 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. 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 survey. 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 hacker 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 hacker: he can duplicate his programs and data, but not his human character recognition skills.

In all the above examples, services 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.

## 3 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:

- 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.
- 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.
- 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 a hacker 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.

# 4 Character Morphing 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.

## 4.1 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:

- 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.
- 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.
- Characters can be sheared (a rectangular morphed into a parallelogram) to the left or to the right.
- Characters may be composed of different fonts, styles, and sizes.
- 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.
- Characters may be randomly "filled."
- 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 cognizable as characters, and thus will not be confusing to humans, but may to a machine.

- Some noise may randomly be added to the graphic image.
- 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 [Sri92] while making no difference to human decision process.
- The letters can have 3-D shadows beside them.
- The letters can be based on one of many handwriting samples (written by many different people), and then morph them according to above techniques.

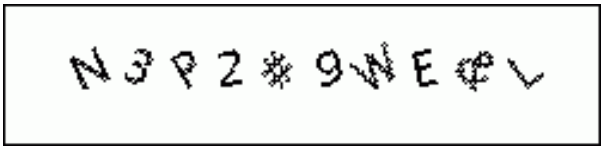
These primitives may be used in composition. For example, a character 'A' can first be assigned Helvetica font, size 12, and bold style, and then sheared to the right by fifteen degree, and then rotated to the left by twenty degree. Also, each character in the string can be morphed independently of others,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

### 4.2 Strength of the Algorithm against OCR Programs

As a convention in cryptography, the algorithm used to morph characters into a graphic representation is public. However, this is not expected to make the job of deciphering an image by an OCR program easier for four reasons. First, as in other cryptographic algorithms, the strength of the primitives lies primarily in the wide range of random parameters they can employ. Most aforementioned primitives can be parameterized and the composition (the product space) of these parameters can result in a huge "parameter space" to search if the hacker is determined to tune OCR programs for recognizing the string. Second, since any or all aforementioned morphing primitives may be randomly selected on a per character basis, the probability of guessing the string right is multiplicative of the probability of recognizing individual ones. Also, this non-homogeneity makes the determination of font, size, and style extremely hard for OCR algorithms. Third, characters may be random and context-free, i.e., they may not make up parts of cognizable words. Since OCR typically uses context-based error correction in its text recognition [Sri92], using random words renders this capability useless. Last, the morphing primitives such as putting the string in a bitmapped image background will confuse the OCR program even as to where the string is. For these reasons, we believe that the recognition of graphical image so generated would be a very hard task for OCR programs, if not impossible.

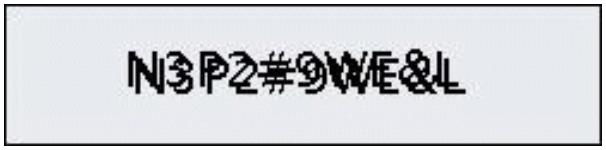
We implemented the baseline version (described in Section 4.3) of the proposed morphing primitives and used them alone or in composition to generate a set of test samples to be recognized by three off-the-shelf OCR software packages, namely, (a) CuneiForm'99 by Cognitive Technology, (b) SunmiPage ScanInsert 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



**Sample 1**

(a) (nothing) (0%)
(b) (nothing) (0%)
(c) ^J^2*9–<<*E<fv (30%)

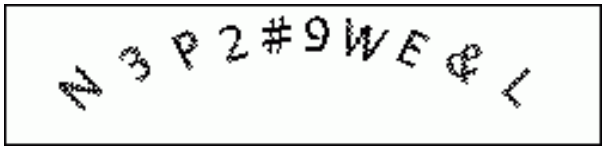


**Sample 2**

(a) (nothing) (0%)
(b) (nothing) (0%)
(c) mtmmmiL (10%)

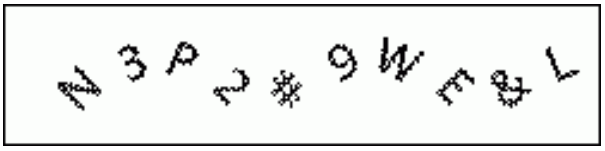
**Sample 3**

(a) (nothing) (0%)
(b) (nothing) (0%)
(c) NJflSRVE&t (30%)



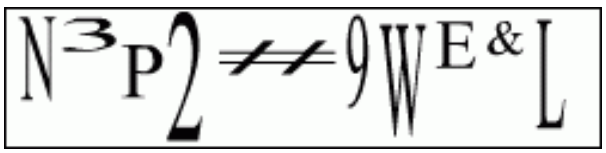
**Sample 4**

(a) ?"–g il'E (10%)
(b) (nothing) (0%)
(c) ^fl*9^* (0%)



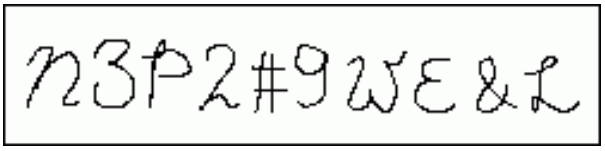
**Sample 5**

(a) (nothing) (0%)
(b) (nothing) (0%)
(c) ^"^.JlV^v (0%)



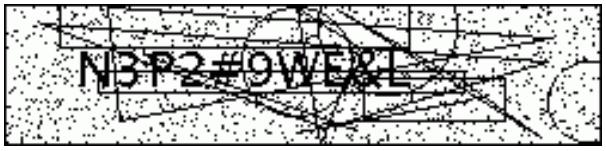
**Sample 6**

(a) (nothing) (0%)
(b) (nothing) (0%)
(c) PpMF'l L M " (0%)



**Sample 7**

(a) (nothing) (0%)
(b) (nothing) (0%)
(c) ft3Mft9hEWL (40%)

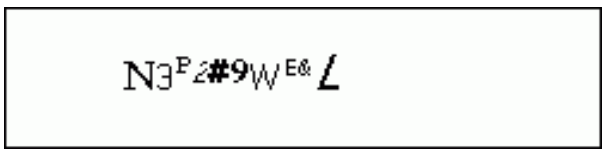


**Sample 8**

(a) (nothing) (0%)
(b) (nothing) (0%)
(c) ^raS^^jwi^^^ (0%)

**Sample 9**

(a) N?P2=9WEEEL (70%)
(b) (nothing) (0%)
(c) ifiSf^WfStV (10%)



**Sample 10**

(a) (nothing) (0%)
(b) N3Pi#'wl (40%)
(c) N3^#9WE*Z (70%)

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

best recognition percentage (per character) below 30%, which is translated into an accuracy of less than $6 * 10^{-6}$ 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. Existing OCR algorithms can be classified into two broad categories:

**Structure-Based:** This type of algorithms extract the geometric structure of a letter and match it with templates of different characters [Fuk90]. Here we illustrate it by an oversimplified example. To recognize the image of an English letter, the image is fitted into a 4x4 grid and the "density" of each grid is determined. The densities in these grids, viewed as a vector of 16 scalars, are compared with characteristic vectors of various characters to see which one offers the best match according to a metric that defines the closeness of two vectors (e.g., Euclidean distance). It is not hard to see that most of the morphing primitives (e.g., rotation) alone would defeat this naive algorithm. For more sophisticated algorithms in this class, we will carefully study how they can be tuned to recognize the morphed images and refine the morphing primitives to defeat this process. We would expect primitives such as rotation, sheaving, 3-D, and filling that destroy or randomize the structure information needed for such algorithms will be most effective against structure-based algorithms.

**Texture-Based:** This type of algorithms converts the image into its frequency domain representation, through wavelet transformation [Nag00]. Texture-based algorithms would be very effective in defeating some of the morphing primitives that do not scramble the frequency domain information. For example, rotation alone would not be able to confuse texture-based OCR algorithms since the frequency domain components are rotation invariant, assuming individual characters can be cleanly separated from each other through edge detection [Fuk90]. We expect primitives that could scramble the frequency domain information such as stretching, shearing, hiding in the picture, slight overlapping, and introducing noise would be useful against texture-based algorithms. Our future research will study how to effectively use these primitives to scramble and destroy the frequency domain information the OCR algorithm depends on. For example, texture-based OCR algorithms typically can be tuned to eliminate noise by filtering out frequency components that are not likely to be contributed by the characters. One countermeasure here is to find the ways to introduce noise that has frequency components significantly overlapping with those that come from the character string. Another countermeasure is to find morphing techniques to spread the frequency components of a character as widely as possible so that filtering would inevitably lead to loss of signal.

We can see that morphing primitives that are effective against one OCR algorithm may not be effective against another one, e.g., rotation. Since multiple OCR algorithms can be used in

combination to decipher the image, it is important to effectively combine primitives to counter all possible attacks. The goal of our future research is to find a proper set of morphing primitives coupled with a proper set of parameters so that an adversary, who has all existing OCR algorithms at his/her disposal, will not be able to decipher the string beyond a certain degree of accuracy, or is able to do so only after a lot of computation. The only constraint is that the resulting string image should not confuse humans.

No matter how much validation is performed, we acknowledge that there is still uncertainty that some day a new algorithm may be proposed to recognize the morphed image much more efficiently and accurately than the current ones. We will proactively search for new OCR algorithms that will pose a threat to our character transformation algorithm and strengthen the algorithm to counter that. As a tradition in cryptography research, we plan to publish our morphing algorithm online (with a web interface) to motivate the research community to try to break it. This would help us realize new vulnerabilities of the algorithm and find ways to fix them.

### 4.3 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.

Usability is one of our major concerns. What range should the parameters fall into so that the resulting image will be "friendly" to humans? We heuristically set the "reasonable" parameter range for some primitives (e.g., -45 to 45 for rotation, -30 to 30 for sheaving, and 0.5 to 2 for stretching) and we found that the resulting image from their composition is easily readable most of the time (over 98% among 1000 samples). Note here that we only need to make this probability large, since the client can simply request another puzzle if the image is not clear enough to read. We will conduct more experiments to determine the maximum allowable range of these parameters. Here we only consider usability to people with normal vision (including those who can obtain near normal vision with the aid of glasses). We concede it is not quite friendly to computer users with eye problems who depend on braille terminals and voice synthesizers. However, in this special case, special provisions such as using strong authentication (e.g., based on public key infrastructure) may be necessary.

Performance of the morphing algorithm is another 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.

## 5 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. A number of key technical issues that relate to the protocol need to be addressed. In the following, we discuss these issues in the context of world wide web applications: a web server asks a web client to solve a puzzle before its transaction requests are granted.

**Grading without remembering the correct answer:** The first important issue is how does a server check whether a client's answer is correct? A simple and obvious solution 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 the puzzle and the answer to the puzzle. This would result in a large amount of state information to be stored and searched on a busy server. Also, in many E-commerce applications, it is desirable that multiple servers handle transaction requests in parallel to maximize throughput. So it is highly likely that the server who generates the puzzle is not the server who grades the puzzle. A stateful approach would require costly reliable replication or sharing of state among these servers. In contrast, making the server stateless would eliminate both problems. Also, a stateless system is more robust since it is much less likely to lose critical information due to system crash. However, a stateless server sounds counterintuitive: how can the server check if an answer to a puzzle is correct without knowing the correct answer itself? Note here that solving the puzzle by the server is not an option since the morphing algorithm is meant to confuse computer programs.

Our solution is to ask the client to remember and carry the correct answer in an encrypted form. 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) [BCK96, PO95] using a secret key (shared among all transaction servers in the aforementioned multi-server case). Digital signature [MOV96] may also be used, but MAC is preferred since it is thousands of times faster [Dai00]. 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 answer to the puzzle and 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.

**Countering replay attacks:** Our solution above relieves the system of having to remember the answers to the puzzles. However, some other state information still needs to be kept to guard against the replay attacks: a human hacker could first solve one puzzle and then reuse the <puzzle, answer> pair together with thousands of program-generated malicious transactions. All these malicious transactions will be accepted by the server because the answer is correct for the puzzle. A countermeasure is to keep one bit for each generated puzzle that indicates whether or not the puzzle has been solved (consumed). A lifespan limit on all puzzles would prevent the amount of state from going "infinite." Though storing a bit is much more storage efficient than storing the whole answer (saving more than 98%), it is no longer a stateless approach and for the multi-server case, costly reliable replication and sharing of state information among servers are still needed.

**Making the system completely stateless:** Fortunately, most applications in which MHP scheme will be used satisfy a nice property called idempotency, which combined with our MACed state approach described above, would make the system both completely stateless and replay-resistant. The system is idempotent if the effect of executing the same transaction multiple times is the same as executing it once. Most of our aforementioned examples are idempotent: in password guessing attack, guessing the same username/password pair multiple times will not give the hacker

more information than trying it just once; the similar can be said about the automatic service and information theft example; in the Yahoo email account DoS example, trying to create accounts with the same name will only get the "account already exists" answer from Yahoo. Our approach to make the server completely stateless without suffering from replay attacks is the following. The client first submits a transaction with no puzzle attached. Upon the receipt of the transaction, the server then generates a puzzle and a MAC that binds together the key arguments (e.g., username and password) of the transaction and the answer to the puzzle. The client will then submit the the same transaction (this state kept by the client) again, but this time together with the answer to the puzzle and the MAC received from the server. The client can not reuse a <puzzle, answer> pair for another transaction with different arguments because the MAC binds the answer to the arguments of the transaction. In this new protocol, the user is not required to do any extra work and the server does not need to keep any state. This approach does introduce one more round of protocol interaction, which results in more network traffic. However, this overhead is much smaller compared to the benefits brought by the statelessness of the system.

**Making the system user-friendly:** So far, only when the client submits the exact answer to the puzzle will the answer be considered correct. The answer can certainly be made case-insensitive, but this is as user-friendly as we can go. Considering that an average of 2% (mentioned in Dr. Peter's Nuemann's CACM short article) of our typing is in error, we may want to tolerate approximate answers using techniques such as counting the length of the longest common sequence between the client's answer and the correct answer. The solution is, instead of hashing the answer into MAC, which is one-way, encrypting the answer using symmetric encryption algorithms such as DES so that the server can recover the answer through decryption and performing approximity test. Since symmetric encryption algorithms are typically much slower than MAC based on hash functions [Dai00], the overhead to be user-friendly is higher computational cost to generate the puzzle, decrypt the encrypted state, and perform the test.

**Making the system adaptive:** The MHP scheme can be used in adaption to the system security needs. For example, when the load on the server is low and the Denial of Service attack is not suspected, the system may allow users to submit transactions without asking them to solve puzzles. Also, the difficulty of the puzzle can also be adaptive, e.g., longer string and stronger morphing primitives/compositions will be used when the system is under attack.

We have proposed techniques to build a generic, stateless, replay-resistant, and multi-server MHP scheme to be parameterized for use in real-world applications. We have implemented a baseline version of the protocol that implements all the aforementioned features except user-friendliness and adaptivity. In our implementation, we chose HMAC-MD5-32 (output truncated to 32 bits) [BCK96, KBC97] for generating the message authentication code (implemented in Dai's publicly available Crypto++ 4.0 library[Dai00]) because its (HMAC-MD5) properties are carefully validated and it can be computed very fast. However, other keyed cryptographic hash functions [PO95] with similar properties may also be used. We truncate the output to 32-bit instead of the 80-bit recommended minimum in [KBC97] because (1) the birth day attack, which compromises nonrepudiation, is not an issue in this protocol, (2) 32 bits still make the probability of "guess it right" negligible,

and (3) this makes the MAC key harder to guess, as shown in [PO95].

## 6 Related Work

The idea of the Mandatory Human Participation (MHP) is inspired by Turing's test for artificial intelligence [Tur50]. 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 3.

An idea similar to the MHP scheme, which was briefly described in an unpublished manuscript [Nao96], was brought to our attention. However, among the types of puzzles suggested in [Nao96], 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 3. Except this, we are not aware of any other similar ideas being presented in the research literature.

Among the established cryptographic measures, biometrics is somewhat related to the MHP scheme. However, biometrics enforces a much stronger semantics, "who are you?" (instead of "are you human?"). Also, the system requirements for biometrics are much more stringent than MHP: they typically require a human user to authenticate himself/herself directly to a physically secured device. Currently, this is not feasible in the Internet where users and customers can be at any corner of the earth.

Juels and Brainard's "client puzzles" [JB99] 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, a hacker 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 a hacker, the speed he can perform I/O as a human.

## 7 Conclusions

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

- 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.

- 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.

- 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.

- 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.

The direct impact of this research is evidenced by its immediate applications in computer security. By accurately implementing new and important security semantics, "human," the MHP scheme solves security problems such as password guessing attacks, automatic information and service theft, and denial of service. These problems are not well addressed by existing cryptographic measures. We expect that the number of applications in which the MHP scheme would be useful will grow since many Internet applications involve the interaction between a human user and a computer system, where "human" is a natural part of the implied protocol semantics.

Character morphing and recognition is an interesting problem in its own merit and carries an indirect impact. Our work is expected to create a game between the character recognition community and the security community. The security community wins this game if they can formalize "unsolved" or poorly understood aspects of human character recognition skills. Conversely, the pattern recognition community wins the game if they can solve the most difficult aspects of pattern recognition. The expected outcomes of such a game are improved pattern recognition algorithms together with an accurate understanding of the limits of such algorithms.

## References


[BCK96] M. Ballare, R. Canetti, and H. Krawczyk. Keyed hash functions and message authentication. In *Proceedings of Crypto'96*, pages 1–15, 1996.

[Bel90] S. Bellovin. Security problems in the tcp/ip protocol suite. *ACM Computer Communication Review*, 19(2), 1990.

[Dai00] W. Dai. *Crypto++ 4.0 Benchmarks*, June 2000.

[Fuk90] K. Fukunaga. *Introduction to Statistical Pattern Recognition.* Academic Press, Inc., 2 edition, 1990.

[Hai96] S. Haigh. Optical character recognition (ocr) as a digitization technology. *Network Notes 37*, 1996.

[JB99] A. Juels and J. Brainard. Client puzzles: A cryptographic countermeasure against connection depletion attacks. In *Proc. of NDSS'99.* Internet Society, March 1999.

[KBC97] H. Krawczyk, M. Bellare, and R. Canetti. *RFC2104: HMAC: Keyed-Hashing for Message Authentication*. Network Working Group, February 1997.

[MOV96] A. Menezes, P. Oorschot, and S. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1996.

[Nag00] K. Nagy. Twenty years of document image analysis. *IEEE Trans. Pattern Anal. Machine Intell*, 22(1):38–62, 2000.

[Nao96] M. Naor. Verification of a human in the loop or identification via the turing test. http://www.wisdom.weizmann.ac.il/ñaor/PAPERS/human_abs.html, September 1996.

[PO95] B. Preneel and P. Oorschot. Building fast macs from hash functions. In *Proceedings of Crypto'95*, pages 1–14. Springer-Verlag, 1995.

[Sri92] S. Srihari. High-performance reading machines. *Proceedings of the IEEE*, 80(7):1120–1132, July 1992.

[SVK00] A. Dos 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*, November 2000.

[Tur50] A. Turing. Computing machinery and intelligence. *Mind*, pages 433–460, 1950.

[UMa] Document and image understanding image server. http://documents.cfar.umd.edu.