

The robustness of a new CAPTCHA

Ahmad Salah El Ahmad
School of Computing Science
Newcastle University, UK
Ahmad.Salah-El-
Ahmad@ncl.ac.uk

Jeff Yan
School of Computing Science
Newcastle University, UK
Jeff.Yan@ncl.ac.uk

Lindsay Marshall
School of Computing Science
Newcastle University, UK
Lindsay.Marshall@ncl.ac.uk

ABSTRACT

CAPTCHA is a standard security technology that presents tests to tell computers and humans apart. In this paper, we examine the security of a new CAPTCHA that was deployed until very recently by Megaupload, a leading online storage and delivery website. The security of this scheme relies on a novel segmentation resistance mechanism. However, we show that this CAPTCHA can be segmented using a simple but new automated attack with a success rate of 78%. It takes about 120 ms on average to segment each challenge on a standard desktop computer.

Categories and Subject Descriptors

D.4.6 Security and Protection, H.1.2 User/Machine Systems.

General Terms

Security.

Keywords

CAPTCHA, Robustness, Segmentation attack, Gestalt perception.

1. INTRODUCTION

A CAPTCHA (Completely Automated Public Turing Test to Tell Computers and Humans Apart) is a program that generates and grades tests that are human solvable, but beyond the capabilities of current computer programs [1]. CAPTCHA is now almost a standard security mechanism for defending against undesirable or malicious Internet bot programs, such as those that spread junk email and grab thousands of free email accounts. It has found widespread application on many web sites including Google, Yahoo, and Microsoft's MSN.

It is widely accepted that a good CAPTCHA must address two main requirements: robustness and usability. Robustness is its capability to resist computers attacks, and usability is the ease with which a human can pass its challenges. In the case of CAPTCHAs, the race between designers and attackers concentrates mainly on robustness, and has attracted considerable attention in the research community, for example in [3, 4, 7, 9, 10]. In this paper, our discussion will focus on the robustness of the most widely used type of CAPTCHAs – the so-called text-

based schemes.

Early research [14] showed that computers can beat humans at individual character recognition. In other words, if a character's position is known, computers can beat humans at recognizing the character. However, the problem of segmentation, which is to identify the location and the right order of characters amongst other clutter, is still challenging for state-of-the-art computer vision and machine learning research. On the other hand, human eyes can do such segmentation easily. The state-of-the-art of CAPTCHA design suggests that the robustness of a text-based CAPTCHA should rely on the difficulty of segmenting its characters. That is, such CAPTCHAs should be "segmentation-resistant". Clearly, *if breaking a CAPTCHA challenge can be reduced to the problem of recognizing its characters, then this CAPTCHA is effectively broken.*

In our early research we have shown that some segmentation-resistant CAPTCHAs could be broken, including the high-profile ones used by Microsoft, Google and Yahoo [5, 6]. In this paper we examine the robustness of a new CAPTCHA, which was until very recently deployed by *Megaupload.com*, one of the largest file sharing and uploading websites [11]. This CAPTCHA is based on a new segmentation-resistance mechanism different to that used by Microsoft, Google and Yahoo.

This paper is organized as follows. Section 2 discusses related work. Section 3 provides an overview of the targeted CAPTCHA scheme. Section 4 describes our segmentation attack in 5 steps, followed by our results in Section 5. Section 6 discusses limitations of our attack, and the lessons we have learned from it.

2. RELATED WORK

Using sophisticated object recognition algorithms, Mori and Malik [3] broke two early CAPTCHAs: EZ-Gimpy (92% success) and the GIMPY (33% success). Chellapilla and Simard [2] attacked a number of CAPTCHAs taken from the web, achieving a success rate ranging from 4.89% to 66.2%. PWNtcha [12] demonstrated the inefficiency of many CAPTCHA implementations.

The principle of "segmentation-resistance" was established by a Microsoft research team, and is now widely accepted. The CAPTCHAs used by Microsoft, Yahoo and Google represents the following three major styles of segmentation resistant mechanisms: using "random arcs" as clutters (by Microsoft), "connected random lines" (by Yahoo), and "crowding characters together" (by Google). Figure 1 shows an example for each of the mechanisms.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

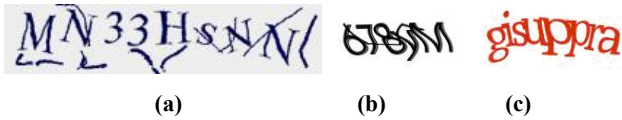


Figure 1. Segmentation-resistant CAPTCHAs from (a) Microsoft, (b) Yahoo, and (c) Google.

Although Microsoft’s design goal was that “automatic scripts should not be more successful than 1 in 10,000” attempts, i.e. a success rate of 0.01%, by using low-cost attacks, we were able to segment the Microsoft CAPTCHA with a success rate higher than 90% [5]. We also broke the Google CAPTCHA and two versions of Yahoo CAPTCHAs [5, 6].

While the camera-ready version of the current paper was prepared, one anonymous reviewer drew to our attention some scripts [16, 17,18] available online for attacking earlier versions of Megaupload CAPTCHA. These earlier versions were much simpler than the one we examine in this paper. For example, the versions that were attacked by [16,18] (see Figure 2 (a) and (c)) used a much lower level of distortion, and barely connected characters with each other. Similarly, the version attacked by [17] (see Figure 2 (b)) used different colors for characters to protect against automated attacks by Optical Character Recognition software. None of these three versions were designed to be segmentation resistant.

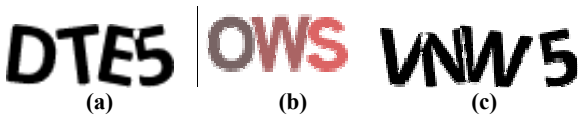


Figure 2. Earlier versions of Megaupload CAPTCHA, attacked by (a) Plowshare, (b) Megaupload-dl (c) Mu-captcha.

3. MEGAUPLOAD CAPTCHA

In what seems to be a clever design, the Megaupload CAPTCHA (see Figure 3 for examples) implements a new segmentation resistance mechanism that relies on camouflaging the areas connecting characters with the challenge’s background. The following key characteristics were observed after studying 100 sample challenges generated by this CAPTCHA:

- Each challenge uses only the colors black and white.
- Characters connect within one another horizontally. In addition, rotation is applied to each character.
- Some parts of characters are removed in the area where they connect with other characters.
- Challenges have a fixed 4 character length (using both letters and numbers).



Figure 3. Megaupload CAPTCHA: 3 example challenges with the correct answers being NAQ6, UCX9, and ZEZ8, respectively.

The Megaupload scheme could be implemented as follows.

- Pick randomly 4 characters as a challenge text;
- Rotate each character to a proper angle;
- Except for the first character, shift all other characters horizontally to the left until they all connect together.

- Finally, change the color of the connection area between characters to that of the image background.

This new segmentation resistance mechanism, which we call “Megaupload style”, bases its robustness on the combination of “connecting characters together” and the “Gestalt Perception” principle. “Connecting characters together” is the process of merging characters horizontally with one another, and humans perform well in segmenting connected characters, but this is still a problem for current computer algorithms [14]. In a sense this is similar to the “crowding characters together” mechanism used by the Google CAPTCHA, but with the difference that the characters are physically merged and not just touching or crowded next to one another. On the other hand, the “Gestalt Perception” principle is used to hide some contents of the characters, but at the same time it suggests that humans can reconstruct individual characters mentally, while current computers still find it a difficult task [8,13].

Whilst writing this paper, we noticed that ReCAPTCHA [15], a well known CAPTCHA service, has started to adopt a new segmentation resistance mechanism, which is similar to the one used in the Megaupload CAPTCHA (see Figure 4).



Figure 4. reCAPTCHA scheme: a sample challenge.

4. A SEGMENTATION ATTACK

In Megaupload CAPTCHA, characters connect with each other horizontally and where they connect, parts are removed. This breaks characters into multiple components, some of which consist of solid black pixels, others consist of solid white pixels and some consist of pixels of varied shades of grey. Using the key observations from section 3, we have developed a segmentation attack that can effectively and efficiently segment challenges generated by the Megaupload CAPTCHA.

A high level description of our attack is as follows:

- Locate black components – these are colored black and belong to individual characters and are never shared with neighboring characters.
- Locate white components – these are colored white color and some are shared by adjacent characters (connected characters), but others are not, such as those found in loops and the outside of the image text.
- We identify the black components of each character and the shared white components, then we put the shared white components in the right location to merge with corresponding black components to form each complete character.

The two main requirements for a segmentation attack are effectiveness and efficiency. We measure attack effectiveness by testing using a random set consisting of 400 sample challenges – no knowledge from this set was embedded in the design of our attack – and we measure its efficiency by calculating the time it takes to segment a sample challenge.

Our attack includes 5 steps, each of which is explained in details in the following sections.

4.1 Extracting Black Components

Extracting black components is important for our segmentation attack, as they define most of the actual character content. We use “Color Filling” Segmentation to extract all black components. “Color Filling” Segmentation or CFS algorithm segments components in an image based on their color. To detect all black components in an image, CFS works as follows. First it detects a black pixel in the image and then traces all of its black neighbors until all pixels in the component are traversed. That is, a component has been segmented. Next, the algorithm locates a black pixel outside the area of the segmented component(s), and starts another traversal to identify the next component. This continues until all black components are segmented. Then, segmented components are stored as a collection of points for manipulation. This process is effectively like using a distinct color to flood each component and it could be used to segment against any color, so we call it “Color Filling” segmentation. Figure 5 (b) shows how each component is now identified by its color.

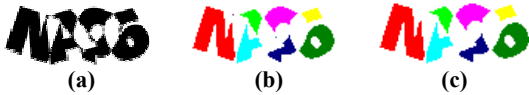


Figure 5. Extracting black components. (a) original image, (b) after CFS black components are identified by color, (c) after removing small black components.

After extracting all black components, we identify and remove small black components by using their pixel count (defined as the total number of pixels inside a given component). Removing these components can simplify further tasks of our segmentation attack without influencing the original shape of characters. Figure 5 (c) show the image output after removing small black components.

4.2 Extracting White Components

In this step we aim to detect all white components, white components could be classified as the connection area between characters (shared white components), the main image background, and loops that occur normally as part of a character’s shape and loops that occur due to the distortion method used in the Megaupload scheme (further discussion about loops will follow in section 4.3). As in the previous step, we use CFS again, but this time we use it against white to extract all white components. Figure 6 (b) shows the effect of CFS.



Figure 6. Extracting white components. (a) original image, (b) after CFS white components are identified by color.

One of the novelties of our attack is this use of the CFS algorithm, where we use it to extract only components that are of interest to our attack, that is by applying it to black and white only, we automatically discarded all components in the CAPTCHA image with colors other than black and white, as those components are not important to our attack.

4.3 Identifying Shared White Components

Although at this stage we have extracted all of the white components, one of the most important steps is to keep only white components which are shared between characters. Other components which we consider as clutter such as the “image background”, and most importantly, the “loops” must be removed

to enhance the effectiveness of our attack. For this we have developed the following methods:

- Image background removal. To remove the image background, we use pixel count to locate the largest white component. The image background does not add any significance to the characters or to the connection area between them. Figure 7 (b) shows the result of removing the image background, notice the difference with respect to Figure 6 (b).

- Loop removal. Loops are white in color and classified into two types: Loops of the first type occur as part of the character shape itself, such as those found in “A”, “Q” and “6” in Figure 5. The other type of loop occurs as a result of connecting characters together, for example, the connection between “C” and “X” will lead to the creation of a loop (see Figure 2 for an example). It is very important to locate and remove loops as they could be mistaken for part of the connection area between characters (i.e. shared white components), this in turn could lead to false segmentation. We have developed the following method to detect and remove loops:

- First, using pixel count, locate the largest 3 white components.
- Second, remove all white components which satisfy both the following conditions: one, if the pixel count of a component is larger than 75 pixels (statistically speaking, most loops have a pixel count larger than this value), and two, if a component is in close proximity to one of the largest 3 white components. Figure 7 (c) shows the results of loop removal. Notice that after loop removal, only white components that are shared by adjacent characters are left.



Figure 7. Identifying shared white components (a) original image, (b) after removing the image background, (c) after removing loops.

4.4 Determining Boundaries of Shared White Components

Even though, the remaining white components constitute all of the connection areas between characters, we still need to determine the corresponding connection areas for each white component. We used the following two observations to locate the left and right boundaries of each connection area. First, white components that are juxtaposed vertically with each other must belong to the same connection area. Second, four characters connected horizontally should typically produce three connection areas between them.

We map the image consisting of all the remaining white components (e.g. Figure 8) into a vertical histogram that represents the total number of white pixels in each column. To locate the left boundaries, we scan the histogram and find columns that have no white pixels but where the column to their right does have some. To locate right boundaries, we scan the histogram for columns that do not contain white pixels but the column to their left does. This process typically returns six boundaries lines as represented in dotted red vertical lines in Figure 8.



Figure 8. Connection areas boundaries shown between dotted vertical lines.

4.5 Merging Components

In this final step, we merge the black and white components to form each individual character. Again, 4 characters that are connected horizontally will typically produce 3 connection areas between them. Using the number of characters and the boundaries of the connection areas, we devised the following heuristics to combine black components with shared white components forming individual complete characters.

- First character: All shared white components inside the first connection area and all black components to its left.
- Second character: All shared white components inside the first and second connection areas and all black components between them.
- Third character: All shared white components inside the second and third connection areas and all black components between them.
- Fourth character: all shared white components inside the third connection area and all black components to its right.

Finally, we convert the color of merged components to black and space characters away from each other horizontally. Figure 9 shows a segmented result.



Figure 9. Putting black and shared white components together to form individual characters.

5. RESULTS

5.1 Attack Success

Our attack achieved a success rate of 82% on a sample set of 100 challenges. Following a common practice in the areas of computer vision and machine learning, we tested our attack on 400 independent samples from a test set and achieved a success rate of 78.25%. We did not examine any sample in the test set for designing our attack, as the test set aims to evaluate that our attack is not specific to the sample set. That is, the attack is generic enough to all challenges generated by this version of Megaupload CAPTCHA.

Given that the state-of-the-art can achieve a success rate of 95% in recognizing individual segmented characters [14], our attack implies that it could lead to an overall (segmentation and then recognition) success rate of 63.7% ($78.5 * 95^4$) for breaking this Megaupload CAPTCHA.

5.2 Attack Speed

Our attack was implemented using Java (little effort was spent in optimizing the run-time of code), and tested on a desktop computer with a 1.86 GHz Intel Core 2 CPU and 2 GB RAM. We ran the attack 10 times on both the sample and test sets to compute its speed, and on average our attack took 118 ms to segment a challenge. Table 1 shows our attack speed.

Table 1. Attack speed

Speed (ms/challenge)	Min	Max	Average
Sample set	112.9	144.5	118.6
Test set	117.8	137.8	121.6

6. DISCUSSIONS

6.1 Segmentation Failure Analysis

6.1.1 Failure of “Extracting BLACK Components”

In this case, the algorithm extracts black components belonging to more than one character (as black components belonging to different characters are still connected). In Figure 10, one black component from the third character “X” is still connected to that of the fourth character “6”.



Figure 10. Failure when extracting black components. (a) original image, (b) a black component is shared by the last 2 characters as identified by color, (c) final segmentation output. Challenge reads “MRX6”.

A possible improvement is to add conditions to ensure that components that belong to different characters are not connected to more than one character, for example, if the overall width of a black component is too large to fit only one character, then further segmentation attempts could be applied to it, for example, in Figure 10, only one pixel connects “X” and “6”.

6.1.2 Failure of “Loop Removal”

This failure is about incorrectly identifying loops. In this case shared white components from connection areas are falsely detected as loops. For example, in Figure 11, all of the shared white components between the second character “Y” and the third character “S” are removed due to this failure.



Figure 11. Failure of loop removal. (a) original image, (b) a lost connection area between the 2nd and 3rd characters, (c) final segmentation output. Challenge reads “CYS2”.

A possible improvement can be derived from counting the total number of connection areas and their location. For instance, we can safely assume that connection areas must exist in an interval of space along the x-axis. If none exist and their total count is less than 3, then this failure has occurred. As a counter measure, more processing could be applied, for instance we could restore some of the lost shared white components by locating any white components situated in the “expected” connection area. An example of this failure is shown in Figure 11. It shows two connection areas located within a relatively large distance of each other, and by using this observation we can safely assume that the missing connection area must exist around the mid-point between them.

6.1.3 Failure of “Determining Boundaries of Shared White Components”

In this failure, as a result of false identification of shared white components boundaries, several connection areas are treated as only one. In Figure 12, the connection area between the second character “E” and its left neighbor “X” and the connection area between the second character “E” and its right neighbor “P” are treated as only one connection area between the first character “X” and the third character “P”.



Figure 12. Failure of determining boundaries of shared white components. (a) original image, (b) failure to locate the connection area boundaries between the 1st and 2nd character and between the 2nd and 3rd character, (c) final segmentation output. Challenge reads “XEP6”.

It is possible to detect this failure by counting the total number of connection areas and their individual widths. If the count is less than 3 and one of connection areas is too large in width (to be contained in only one connection area), then a possible recovery from this failure could be to split the widest connection area into 2 or 3 connection areas depending on their total count. For instance, in Figure 12, only 2 connection areas are found, but since the first connection area to the left of the image in Figure 12 (b) has the larger width compared to the other area, then recovery from this failure can be made by splitting it into two connection areas, one for “X” and “E” and another for “E” and “P”.

6.1.4 Failure of “Merging Components”

This failure is caused by falsely adding the same black component(s) to more than one character. This failure was observed when narrow characters connect with much wider characters, in which black components from wider characters span the boundaries of narrow characters. In Figure 13, the second character “X” is connected with the much wider character “W”, and since some of the black components, which belong solely to “W” are within the connection area boundaries of “X”, the merging algorithm falsely assumes that the same black component belongs to “X” as well.

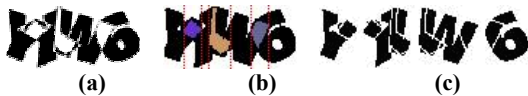


Figure 13. Failure of merging. (a) original image, (b) connection areas boundaries (c) final segmentation output. Challenge reads “YXW6”.

We could detect this failure by checking if any of the black components were merged into more than one character. To recover from this failure, we could remove the redundant black component from all characters except from the widest characters. For example, in Figure 13, the additional black component in the second character “X” could be removed to enhance the overall output of the segmentation method.

6.2 Failure Comparison

Figure 14 compares the failure rate of our attack between the sample and test sets. In the sample set, 4% of the failures were due to “extracting black components” (versus 7 out of 400, i.e. 3.5%

in the test set), 7% are due to failure of “loop removal” (versus 10.5% in the test set), 2% are due to failure of “determining boundaries of shared white components” (versus 1.25% in the test set), and finally 5% are due to failure of “merging” (versus 6.5% in the test set).

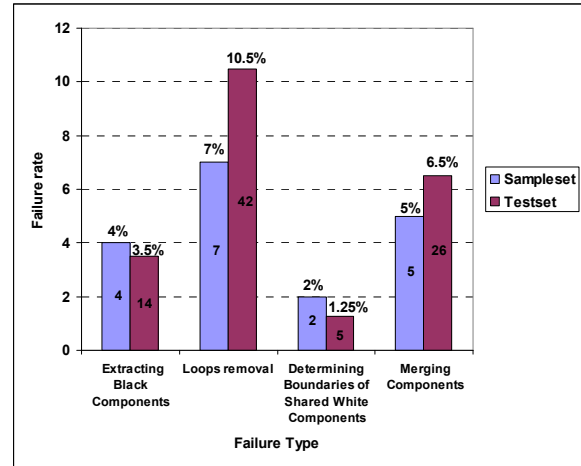


Figure 14. Analysis of failure cases.

6.3 Defense

It is possible to make Megaupload and similar schemes more resistant to our segmentation attack. For example, all of changes listed below could further complicate our attack and effectively lower its segmentation success rate or even defend against it:

- Randomly varying the width of individual characters could confuse some parts of our attack.
- Making it as hard as possible to know the actual number of characters in each challenge could reduce our attack effectiveness.
- Make it harder to distinguish between loops and shared white components.
- Letting characters connect with each other more randomly.

On the other hand, designing a CAPTCHA that exhibits both strong robustness and good usability (in the sense of its friendliness to Humans) is still challenging. Many schemes have attempted such a goal, but have failed to provide robustness, usability or both. For instance the MSN CAPTCHA failed to provide good security [5], whereas, the Google CAPTCHA failed to provide good usability [13]. That said, it might not be as straight forward for a CAPTCHA to change its original design, as any changes must satisfy both requirements of robustness and usability, for instance, increasing the CAPTCHA challenge length will make it slower for humans to solve, on the other hand, adding more distortion will make it harder (and slower) to solve.

6.4 Lesson

The method of connecting characters together, if implemented properly, offers good segmentation resistance. Similarly, applying the Gestalt perception principle can also offer good segmentation resistance. Therefore in principle it is a good idea to combine both methods in the expectation of achieving better segmentation resistance that either can provide. However, the implementation of

this Megaupload CAPTCHA turns out to reduce such an expected “double insurance” into an easily solvable problem: identifying shared white components and merging them with adjacent black components. We recommend future designs to keep two methods to enforce rather than weaken each other.

7. CONCLUSION AND FUTURE WORK

Our attack has demonstrated that a new segmentation resistance mechanism for text-based CAPTCHAs that depends partially on the “Gestalt Perception” principle is flawed.

Each individual technique used in our attack is standard, but the combination of them is new. In particular, Megaupload represents a new segmentation resistance mechanism and our work for the first time shows that this mechanism can be defeated with a high success rate. Our results are significant in the sense that they offer insights on the security of this mechanism and will help CAPTCHA designers to avoid making similar mistakes.

We are continuing to analyze the security of this new segmentation resistance mechanism as implemented in reCAPTCHA, which is similar to, but appears to be simpler than, the implementation described in this paper.

Interesting future work includes an analysis of the sensitivity of our attack toward changes in the Megaupload CAPTCHA (for example, changes in the font size and/or type, color deployed, the number of characters). Such an analysis is likely to reveal valuable lessons for the design of more robust CAPTCHAs and for the design of a more generic attack.

8. ACKNOWLEDGMENTS

We thank our anonymous reviewers for their helpful comments and for bringing to our attention related work that we were not aware of. Finally we would like to thank Miss Alex Sparks for proofreading an early version of this paper.

9. REFERENCES

- [1] von Ahn, L., Blum, M., and Langford, J. 2004. Telling humans and computers apart automatically. *Commun. ACM* 47, 2 (Feb. 2004), 56-60. <http://doi.acm.org/10.1145/966389.96639>
- [2] K Chellapilla, K Larson, P Simard and M Czerwinski, “Designing human friendly human interaction proofs”, *ACM CHI’05*, 2005.
- [3] Greg Mori and Jitendra Malik. “Recognising Objects in Adversarial Clutter: Breaking a Visual CAPTCHA”, *IEEE Conference on Computer Vision and Pattern Recognition (CVPR’03)*, Vol 1, June 2003, pp.134-141.
- [4] J Yan and A S El Ahmad. “Breaking Visual CAPTCHAs with Naïve Pattern Recognition Algorithms”, in *Proc. of the 23rd Annual Computer Security Applications Conference (ACSAC’07)*. FL, USA, Dec 2007. IEEE computer society. pp 279-291.
- [5] J Yan and A S El Ahmad. “A Low-cost Attack on a Microsoft CAPTCHA”, *15th ACM Conference on Computer and Communications Security (CCS’08)*. Virginia, USA, Oct 27-31, 2008. ACM Press. pp. 543-554.
- [6] J Yan and A S El Ahmad. “Is cheap labour behind the scene? - Low-cost automated attacks on Yahoo CAPTCHAs”, *School of Computing Science Technical Report*, Newcastle University, England. Apr, 2008.
- [7] K Chellapilla, K Larson, P Simard and M Czerwinski, “Building Segmentation Based Human-friendly Human Interaction Proofs”, *2nd Int’l Workshop on Human Interaction Proofs*, Springer-Verlag, LNCS 3517, 2005.
- [8] M Chew and HS Baird. “BaffleText: a human interactive proof”. *Proc. of 10th IS&T/SPIE Document Recognition & Retrieval Conference*, 2003.
- [9] AL Coates, H S Baird and RJ Fateman. “PessimPrint: A Reverse Turing Test”, *Int’l. J. on Document Analysis & Recognition*, Vol. 5, pp. 158-163, 2003.
- [10] HS Baird, MA Moll and SY Wang. “A highly legible captcha that resists segmentation attacks”. *Proc. of Second Int’l Workshop on Human Interactive Proofs (HIP’05)*, ed. by HS Baird and DP Lopresti, Springer Verlag. LNCS 3517, Bethlehem, PA, USA, 2005.
- [11] Reviewcentre. <http://www.reviewcentre.com/reviews169598.html> . Accessed in Feb, 2010.
- [12] PWNtcha. <http://caca.zoy.org/wiki/PWNtcha> . Accessed in Feb 2010.
- [13] J Yan and A S El Ahmad. “Usability of CAPTCHAs or usability issues in CAPTCHA design”, *Proceedings of the 4th Symposium on Usable Privacy and Security. SOUPS ’08*, vol. 337. ACM, NY, pp. 44-52. DOI= <http://doi.acm.org/10.1145/1408664.1408671>
- [14] K Chellapilla, K Larson, P Simard and M Czerwinski, “Computers beat humans at single character recognition in reading-based Human Interaction Proofs”, *2nd Conference on Email and Anti-Spam (CEAS)*, 2005.
- [15] reCAPTCHA. <http://recaptcha.net/> . Accessed in Feb 2010.
- [16] Plowshare. <http://code.google.com/p/plowshare>. Accessed in March 2010.
- [17] Megaupload-dl. <http://code.google.com/p/megaupload-dl/>. Accessed in March 2010.
- [18] Mu_captcha . http://herecomethelizards.co.uk/mu_captcha/. Accessed in March 2010.