PixelCAPTCHA
A Unicode Based CAPTCHA Scheme

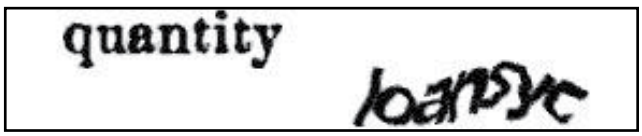
Gursev Singh Kalra, Salesforce.com

Abstract. This paper will discuss a new visual CAPTCHA [1] scheme that leverages the 64K Unicode code points from the Basic Multilingual Plane (plane 0) to construct the CAPTCHAs that can be solved with 2 to 4 mouse clicks. We will review the design principles, the security mechanisms and its various features. We will also discuss the potential attack vectors on the proposed CAPTCHA scheme. The proposed CAPTCHA scheme will also be available as an open source Java library in near future.

Introduction

A "Completely Automated Public Turing Test to tell Computers and Humans Apart," or "CAPTCHA," is used to prevent automated software from performing actions that degrade the quality of service of a given system. CAPTCHAs aim to ensure that the users of applications are human and ultimately aid in preventing unauthorized access and abuse.

There are several types of CAPTCHA schemes that present audio and/or visual challenges to the user. These challenges require a human to interpret them and supply the solution that is validated by the server to allow or disallow the request. Image shows a reCAPTCHA [2] example

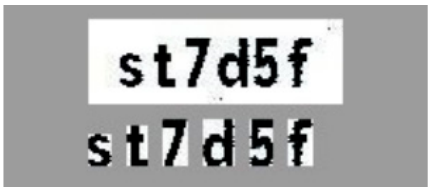


This paper has two main sections: the first discusses challenges with existing visual CAPTCHA schemes and the second section discusses the new PixelCAPTCHA scheme in detail.

Challenges with existing CAPTCHA schemes

Most of the visual CAPTCHAs rely on English alphabets and numerals, which makes them keyboard and locale sensitive. When the conforming fonts are used to build CAPTCHAs, solving the CAPTCHA can be as easy as running the Optical Character Recognition engines after removing the noise. The visual CAPTCHAs often have a similar threat model where the attackers follow three-step process to automatically solve them. The first step is to remove the noise, also called as preprocessing. Individual characters are then segmented in the second step followed by third step of character classification.

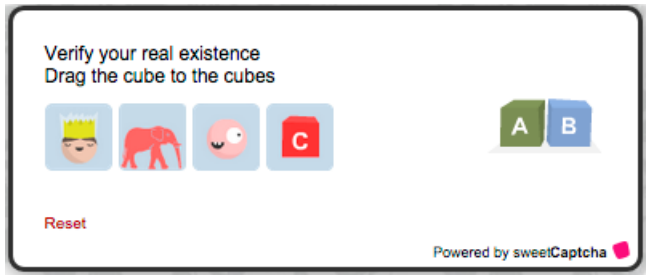
The image below shows a simple segmentation example [3].



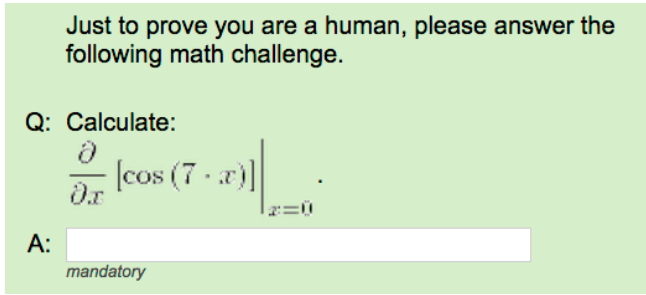
Character segmentation is typically the more difficult part of the automated CAPTCHA solving process. Once individual characters are segmented, computer algorithms can do a very good job of classifying the individual character images to corresponding alphabet/numeric characters. Given the small number of possible characters in the existing CAPTCHAs, automated classifiers can be written for a large number of the existing CAPTCHA schemes. The automated solvers can also use supervised machine learning algorithms to extract the features from the large number of test CAPTCHAs and solve new CAPTCHAs with high accuracy.

Owing to a common threat model between the visual, text based CAPTCHAs, researchers have experimented with alternate forms of CAPTCHAs with different threat models. These CAPTCHAs require you to interpret real world images, videos, solve calculus problems, read advertisements etc. Several of these alternate CAPTCHA implementations require a prior manual generation effort into building a knowledge base on which the users are tested. A few examples are sweetcaptcha, random.irb.hr etc…

The image below shows an example sweetCaptcha [4].



The image below shows a calculus based CAPTCHA [5].



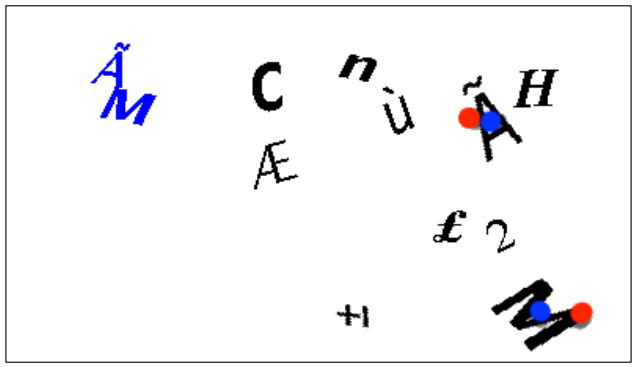
PixelCAPTCHA

Let us now look at a simple PixelCAPTCHA example and build an intuition around the proposed CAPTCHA scheme. On the image below, you will see 2 blue characters on the left and 10 characters in black on the right. You will also see the helper blue and red dots on the CAPTCHA. These dots are only for demonstration and to help build an intuition.

The blue characters are the challenge and the black characters contain the solution among other random characters. To solve the CAPTCHA, you will need to identify the black characters similar to the blue characters and click on the black ones. In the image below, the blue dots on the black characters are the actual solution coordinates and the red dots are the points where the user has clicked. The set of mouse click

coordinates makes your solution, which is submitted to the server for verification. The server computes the sum of minimum distance between the correct solution coordinates and the ones submitted by the user.

In the current example, the server will compute the two distances between the red and the blue dots and sum them up to arrive at a total distance/deviation from the correct solution coordinates. This deviation then will be compared against a pre-computed threshold for a given CAPTCHA to make a decision. The comparison threshold is different for each CAPTCHA and is calculated during the CAPTCHA generation process.



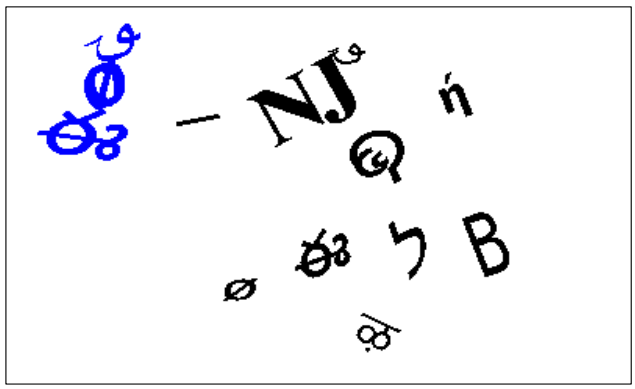
Features

Now that we have built an intuition for the CAPTCHA scheme, we will look at the various supported features in this section. These features allow you to control the CAPTCHA size, orientation, choose a character set and more.

Configurable Challenge and Response Count

The image discussed above was an example of a minimum complexity CAPTCHA that has 2 challenges and 10 responses to choose from, with a character set limited to 0 to 255 Unicode range. The PixelCAPTCHA library supports multiple configurations using which you can build CAPTCHAs that have 2 to 4 challenges, 10 to 12 responses to choose from and any set of characters from the Unicode Basic Multilingual Plane [6] (a.k.a. plane 0). Below is an additional PixelCAPTCHA example.

The image below shows a CAPTCHA with 3 challenges and 11 responses derived from 0-4095 Unicode code points

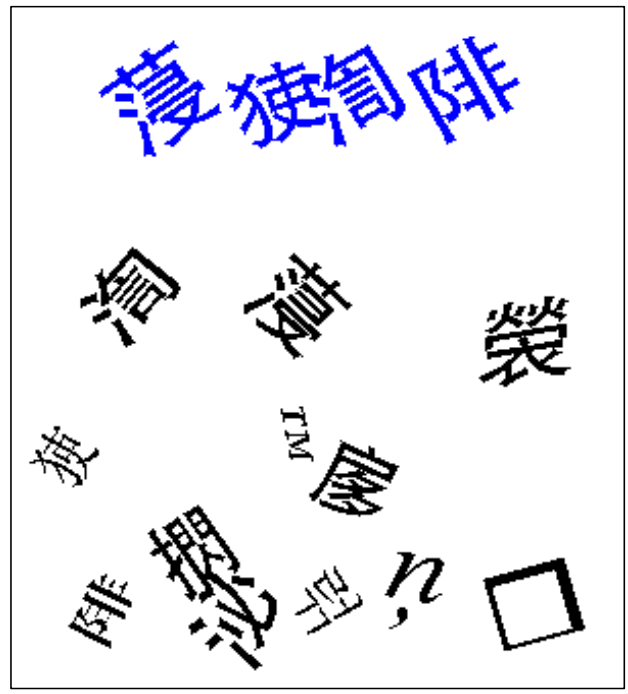


CAPTCHAs for Various Device Types

The PixelCAPTCHA library accepts custom dimensions and calculates the minimum and maximum Font sizes to be used to draw individual characters for the CAPTCHAs in addition to adjusting the challenge text orientation as discussed below.

Horizontal Orientation: When the CAPTCHA width is more than the height, the CAPTCHA gets a horizontal orientation, and the challenge text is drawn vertically to the left side of the CAPTCHA. The horizontal orientation is more appropriate for desktops or devices with larger screen areas. The examples discusses so far are of the horizontal orientation.

Vertical Orientation: When the CAPTCHA height is more than the width, the CAPTCHA gets a vertical orientation, and the challenge text is horizontally drawn to the top of the CAPTCHA. I may be easier for mobile users to view and solve vertical CAPTCHAs on their devices. The image below shows an example of vertical CAPTCHA generated by the PixelCAPTCHA library.



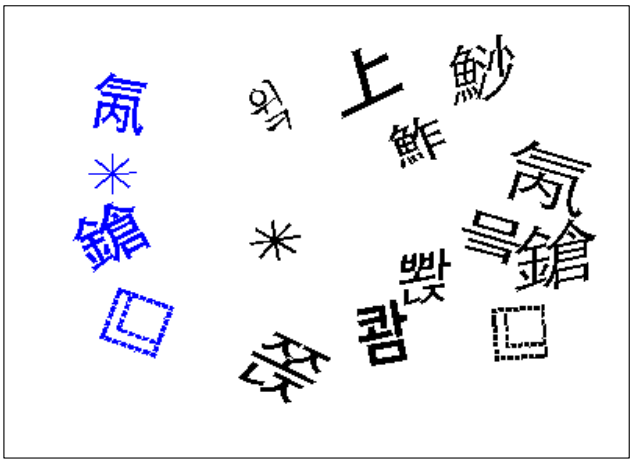
Configurable Character Set

Based on your requirements you can choose the character set for your CAPTCHA. The character set can be a list of Unicode code points from the Unicode plan 0 or multiple character ranges. This allows you to test out user acceptability before finalizing the CAPTCHA configuration for your application. A few examples to custom character sets are shown below:

- 0-255 – configures the PixelCAPTCHA library to use characters only from 8-bit ASCII character range.
- 0-4095 - configures the PixelCAPTCHA library to use characters only from 0 to 4095 Unicode code points.
- 65,66,67,68,69,70,71,72,73,74,75,76,77,78,79,80,81,82,83, 84,85,86,87,88,89,90 – instructs the library to include only the uppercase characters

It is however recommended that the entire Unicode plane 0 code points be used for enhanced security as discussed in the security analysis section below.

The image shows a CAPTCHA with 4 challenges and 12 responses derived from the Unicode plane 0.

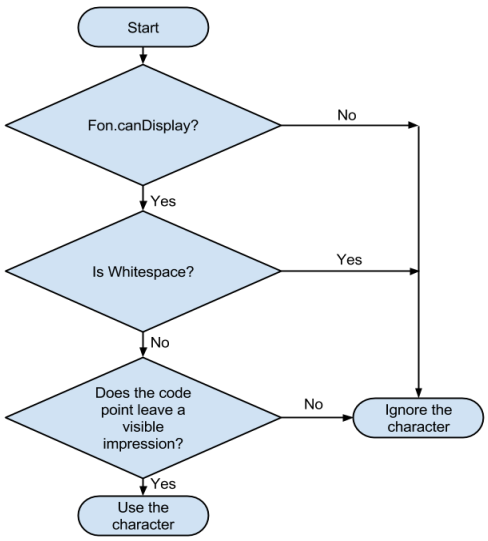


Printable Character Identification

Since the CAPTCHA relies on character correlation in order to be solved, it is very important that the characters leave a distinct impression on the CAPTCHA images. To be able to visually represent the various characters for correlation on the CAPTCHA image, the challenge and response Unicode code points could not be whitespaces and the Font [7] used to draw those must have valid glyphs [8] for each of the chosen Unicode code points. After checking for these conditions, the generated CAPTCHAs continued to have missing characters and the effect was more pronounced with smaller Unicode code points.

For example, when 0 to 255 Unicode code point range was used, more number of CAPTCHAs had missing characters. Further analysis revealed that Unicode code points 0 through 32 did not leave any visible imprints on the CAPTCHA images.

To ensure that the generated CAPTCHAs had visible and distinct characters, the following checks (shown in image below) were applied to each code point for all the characters in the Unicode plane 0 to identify those characters.



The table shows the results of the analysis. To conclude: in addition to code points corresponding to white space and non-valid glyphs, there was a significant number of code points that would not leave a visible imprint on the images.

Unicode Range	Total # of characters	# Of Whitespaces	# Of Font.canDisplay	Expected printable characters	Actual Count
0-255	256	10	256	246	189
0-4095	4096	10	3080	3070	2958
0-65535	65536	26	51878	51852	51580

Inbuilt Cache

The PixelCAPTCHA library has its own, in memory cache to store CAPTCHA solutions and corresponding identifiers. The current release does not require any persistent storage configuration. The storage times out individual CAPTCHA solutions and expires CAPTCHAs on single access, providing better security.

Service Provider Interface

The PixelCAPTCHA library also provides a new Service Provider Interface that you can implement to create wrappers around other CAPTCHA libraries and use them in your code without making extensive changes to your primary application. This allows you to decouple PixelCAPTCHA from your main application code.

Functional and Usability Benefits

The proposed CAPTCHA scheme offers the following additional benefits:

- It allows users to avoid typing and solve CAPTCHAs with a few clicks. This will offer usability improvements on the mobile devices where typing is a challenge and few taps can be used to solve the CAPTCHAs.
- The CAPTCHA scheme is independent of language, keyboard style and locale.
- The CAPTCHA generation process is completely automated.

Security Analysis

The suggested CAPTCHA scheme has been designed with several security features in mind and this section will visit those in some detail.

Probabilistic Analysis of Protection Against Random Guessing

In the very first image of this whitepaper, we discussed a CAPTCHA that has 2 challenges and 10 possible solution options to choose from. Let us assume that an attacker tries to solve that CAPTCHA by randomly selecting coordinates from any 2 of the 10 solution characters. The probability of correctly guessing the solution is $2/(10*9) = 1/45$. This assumes that the order of the mouse clicks is irrelevant. That is, the attacker can click on the potential solution in any order.

For example, lets say that the two challenge characters are A and B, drawn vertically in that order, with A on the top. Consider the following two scenarios:

First: To solve the CAPTCHA, a user may be required to click on A and B in the solution area in any order, making up two possible solutions. The server will check for two possible solutions. This has a benefit of better usability, but offers less security in cases when attackers plan to randomly guess solution coordinates.

Second: To solve the CAPTCHA, a user will be required to click on A and B in the solution area in that particular order, making up only possible solution. The server will also check for only one solution. This has a benefit of higher security, but has a usability challenge.

The PixelCAPTCHA library can be configured to run in either of the two modes and instructed to honor or ignore the order of the clicks. The table below shows different CAPTCHA configurations and corresponding probabilistic analysis of protection against random guessing when ordered or unordered solution clicks are required.

		Probability	
Challenge Count	**Response Count**	**Unordered Clicks**	**Ordered Clicks**
2	10	1/45	1/90
2	11	1/55	1/110
2	12	1/66	1/132
3	10	1/120	1/720
3	11	1/165	1/990
3	12	1/220	1/1320
4	10	1/210	1/5040
4	11	1/330	1/7920
4	12	1/495	1/11880

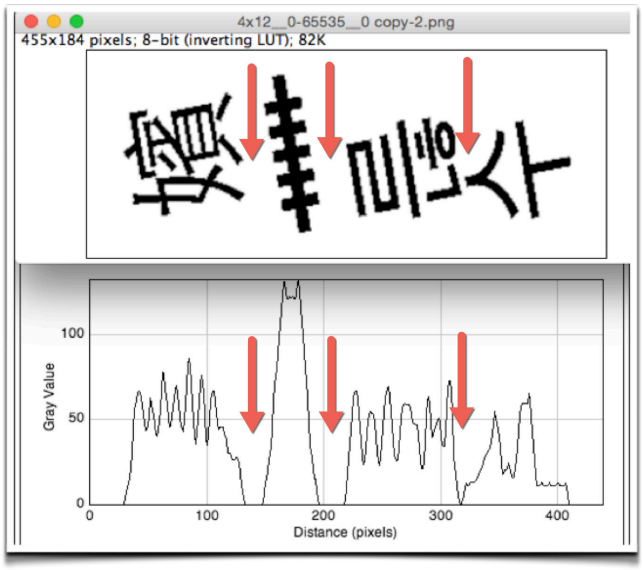
Higher Classification Complexity

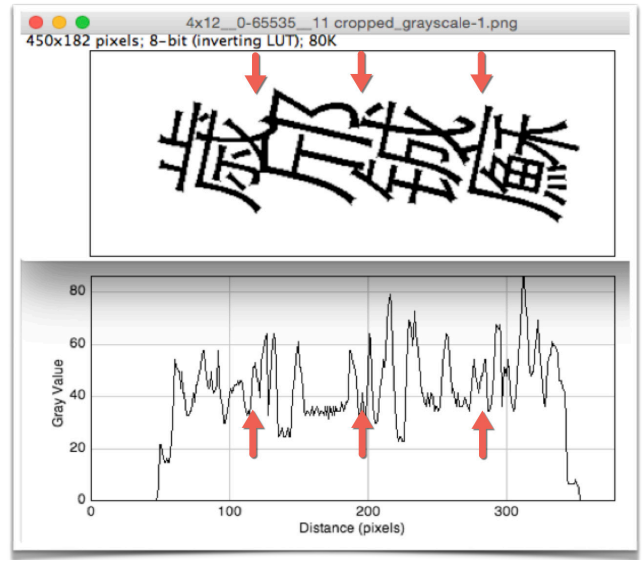
The Basic Multi Lingual Unicode plain consists of 65536 code points. It was concluded during the analysis that out of the possible 65536 Unicode code points, 51,000+ code points could be reliably drawn on the CAPTCHA. Having a large character space increases the complexity to write reliable classification algorithms. The Printable Character Identification section above has discussed how these Unicode code points were identified.

Collapsed Challenge Characters to add Segmentation Complexity

Individual character segmentation is an important aspect of automatically solving CAPTCHAs and one of the PixelCAPTCHA features is to add segmentation difficulty to the challenge characters; achieved by a random overlap between successive challenge characters.

The images below show the challenge component of two different CAPTCHAs. The top portion of each image shows the plotted challenge text and the lower portion shows the black pixel distribution [9] along the vertical axis. Since the challenge characters were not collapsed in the upper image, automated algorithms will be able to segment individual characters because of lower number of black pixels along the vertical axis. However, the bottom image has a random overlap between the challenge characters and there was no visible dip in the black pixel count along the vertical axis. This increases the difficulty to segment individual characters and provides better security.





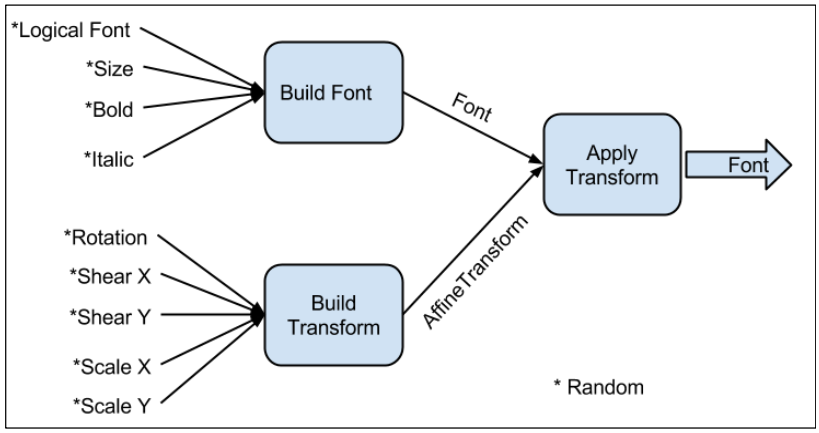
Random Font Construction

Randomly generated Fonts are used for each CAPTCHA character to augment the character correlation/classification complexity. The bullet points and the image below depict the process used to generate a separate Font for each CAPTCHA character.

- Pick a random logical Font from the list below. The current version uses Logical Fonts [10] as they are always present. The Physical Fonts [10] that may differ between systems.
  - Font.SANS_SERIF
  - Font.SERIF
  - Font.DIALOG
  - Font.DIALOG_INPUT
  - Font.MONOSPACED
- Choose random values for the following attributes:
  - Size
  - Bold (boolean)
  - Italic (boolean)

- Construct a Font from the values in step 1 and 2.
- Choose random values for the following
  - Font size multipliers for X and Y-axis (two separate values
  - Rotation in degrees (positive or negative)
  - Shear values for X and Y-axis (two separate values).

  To aid text readability and identification, the absolute shear values are restricted to be less than 0.5.
- Use the random scaling, rotation and shear parameters to construct an AffineTransform [11].
- Apply the AffineTransform to the Font constructed in step 3 to get a new Font, which is then used for drawing a character on the CAPTCHA.



Potential Attacks

Having both challenge and the solution text are part of the same CAPTCHA image presents a security risk. A typical attack pattern may be segmentation of the challenge characters and correlation between the segmented challenge and the solution characters. Computer vision and machine learning algorithms may also be leveraged to solve the proposed scheme.

REFERENCES

1. http://<en.wikipedia.org/wiki/CAPTCHA>
2. http://<www.google.com/recaptcha/intro/index.html>
3. http://<citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.96.3886&rep=rep1&type=pdf>
4. http://<sweetcaptcha.com/>
5. http://<random.irb.hr/signup.php>
6. http://<en.wikipedia.org/wiki/Plane_%28Unicode%29>
7. http://<docs.oracle.com/javase/7/docs/api/java/awt/Font.html>
8. http://<en.wikipedia.org/wiki/Glyph>
9. http://<imagej.nih.gov/ij/>
10. http://<docs.oracle.com/javase/tutorial/2d/text/fonts.html>
11. http://<docs.oracle.com/javase/7/docs/api/java/awt/geom/AffineTransform.html>

ABOUT THE AUTHOR

Gursev Singh Kalra is a Sr. Product Security Engineer at Salesforce.com. He worked with McAfee as a Senior Principal Consultant and led multiple software security service lines. He has authored free security tools like JMSDigger, TesserCap, Oyedata, SSLSmart etc. He has written several security-related whitepapers and his research has been voted among the top ten web hacking techniques of 2011 and 2012. He has spoken at conferences like BlackHat, OWASP AppSec, NullCon, Focus, ToorCon, and Infosec Southwest etc.
E-mail: gkalra@salesforce.com