An Attack on Hollow CAPTCHA Using Accurate Filling and Nonredundant Merging

Jun Chen, Xiangyang Luo, Jianwei Hu, Dengpan Ye and Daofu Gong

1State Key Laboratory of Mathematical Engineering and Advanced Computing, Zhengzhou, People’s Republic of China; 2School of Computer, Wuhan University, Wuhan, People’s Republic of China

ABSTRACT

As one of the effective access control mechanism, CAPTCHA can provide privacy protection and multimedia security for big data. In this paper, an attack on hollow CAPTCHA using accurate filling and nonredundant merging is proposed. Firstly, a thinning operation is introduced to repair character contour precisely. Secondly, an inner-outer contour filling algorithm is presented to acquire solid characters, which only fills the hollow character components rather than noise blocks. Thirdly, the segmentation to the solid characters produces mostly individual characters and only a small number of character components. Fourthly, a minimum-nearest neighbor merging algorithm is proposed to obtain individual characters without redundancy. Lastly, the convolutional neural network (CNN) is applied to acquire the final recognition results. The experimental results based on the real CAPTCHA data sets show that comparing with the existing attack methods on hollow CAPTCHA, the proposed method has higher success rate and superior efficiency.

KEYWORDS
Privacy protection; Multimedia security; Big data; Hollow CAPTCHA; Attack; Inner-outer contour; Minimum-nearest neighbor

1. INTRODUCTION

With the proliferation of network, huge amounts of data are being explosively generated every day. How to protect the security and privacy for big data becomes a key issue in this field. As one of the effective access control mechanisms, CAPTCHA (Completely Automated Public Turing test to tell Computers and Humans Apart [1]) is widely applied to many large websites, including Google, Yandex, and so on. Nowadays, CAPTCHA has played an increasingly important role in privacy protection and multimedia security for big data.

At present, the frequently used CAPTCHAs are text-based CAPTCHA, image-based CAPTCHA, short-message-service-based CAPTCHA, audio-based CAPTCHA, etc. Among them, the most widely-used CAPTCHA is the text-based schemes [2], an image containing several distorted characters. The typical types of text-based CAPTCHA are solid CAPTCHA, hollow CAPTCHA, 3D CAPTCHA, etc.

In order to verify the security and reliability of CAPTCHA, some corresponding attack technologies came into being. The research on CAPTCHA attack is always a win-win result [2]. Some CAPTCHAs are hard to break, and their features can be used as an important theoretical reference for designing high-security CAPTCHA systems. Some CAPTCHAs can be successfully broken, relevant techniques can be extended to other related fields, such as document recognition [3], speech recognition [4] and network localization [5]. It is of great significant to research the CAPTCHA attack technique.

In this paper, we propose an attack on hollow CAPTCHA using accurate filling and nonredundant merging (AF&NM). The main contributions of this paper are as follows.

(a) A new framework of hollow CAPTCHA attack is presented. Within this framework, five steps are effectively integrated, including preprocessing, filling, segmentation, merging, recognition. It can improve efficiency due to a classification of segmented images. If the segmented image is an individual character, it will be recognized directly; otherwise, it will be merged firstly and then recognized.

(b) A thinning operation based on removing pixels iteratively is introduced in the preprocessing stage. It can repair the character contour precisely while retaining the original structure.

(c) An inner-outer contour filling algorithm based on 8-neighborhood pixel detection is proposed in the filling stage. It can accurately distinguish character blocks from noise blocks, and quickly convert hollow characters into solid characters.

(d) A minimum-nearest neighbor merging algorithm of character components is presented in merging stage. According to structural features and connection
order, the algorithm can combine character components into individual characters without redundancy. It can improve the attack efficiency by reducing candidate characters’ number.

2. RELATED WORK

There are two known ways of text-based CAPTCHA attack: whole-based strategy and segmentation-based strategy.

In whole-based strategy, the character image of CAPTCHA can be recognized without segmentation. Some CAPTCHA attack methods have been proposed. For example, Mori and Malik [6] use the shape context matching technique to break the Gimpy scheme with a success rate of 33%. Bursztein et al. [7] showed that 13 kinds of CAPTCHAs on popular websites were vulnerable to automated attacks with Spatial Displacement of the Neutral Network, and they proposed a universal method for global recognition of the overlapping and adherent CAPTCHAs.

In a segmentation-based strategy, the boundary of the characters should be determined first, and then the CAPTCHA image can be segmented into individual character images to be recognized finally. The key to the strategy is whether CAPTCHA images can be segmented correctly. Starostenko et al. [8] defined three-color bar code by skillfully using the number of vertical pixels to break reCAPTCHA scheme with a success rate of 54.6%. Ahmad et al. [9] segmented characters by thoroughly analyzing distinctive shape patterns, and achieved 46.75% success on Google scheme and 33% success on reCAPTCHA scheme. Moy et al. [10] developed distortion estimation techniques to attack EZ-Gimpy of CMU with a success rate of 99% and four-letter Gimpy-r with a success rate of 78%. Nachar et al. [11] used edge corners and fuzzy logic segmentation/recognition technique to break the CAPTCHAs of eBay, Wikipedia, reCAPTCHA, Yahoo!, and the success rates were 68.2%, 76.7%, 62.5% and 57.3% respectively.

The above attack methods are used on solid CAPTCHAs and not aimed at hollow CAPTCHAs directly. In 2013, Gao et al. [12] first analyzed hollow CAPTCHAs’ robustness and proposed a general method to break a variety of hollow CAPTCHAs, with the success rates ranging from 36% to 89% and the average time per challenge ranging from 1.23s to 5.30s. In addition, Gao et al. [13] effectively broke both hollow and solid CAPTCHAs based on 2D Log-Gabor filters and achieved a success rate ranging from 5% to 77% and the average time per challenge ranging from 2.81s to 28.56s.

Despite having advantages, there is some room for Gao’s two methods to improve.

(a) Recognition accuracy can be improved. Both methods are “jigsaw puzzle” [12] thinking, all character blocks that are likely to be grouped together are taken as candidate characters. Since the number of candidate characters is large, the relative proportion of correct characters is small. Naturally, the recognition accuracy would be reduced.

(b) The attack time cost can be decreased. In [12], the breakpoint location algorithm used in repairing contour needed traverse all characters. It leads to high time cost and much redundancy. In [13], hollow characters are segmented into a large number of segments. It can further produce a huge set of possible combinations during the recognition stage. It would increase the time and difficulty on attack.

3. PROPOSED AF&NM METHOD

In order to solve the above problems, this paper proposes an attack on hollow CAPTCHA using accurate filling and nonredundant merging. By observing and analyzing the features of hollow CAPTCHAs, we propose a principal framework (Figure 1) and some algorithms to improve the accuracy and reduce the complexity, mainly including 8-neighborhood detection outer contour algorithm, inner-outer contour filling algorithm and minimum-nearest neighbor merging algorithm.

3.1 Principal Framework and Main Steps

The principal framework of AF&NM method mainly includes five steps: preprocessing, filling, segmentation, merging and recognition.

(a) Preprocessing: Convert images into binary ones, then thin and repair the character contour.

(b) Filling: First, mark all connected components with different labels. Second, acquire noncharacter components by eliminating outer contour. Third, change noncharacter components’ labels to the same as background’s label. Lastly, fill character components.

(c) Segmentation: According to the different connected component labels, segment solid character image obtained in step b) into most of individual characters and a small number of character components.
Figure 1: Principal framework of proposed method

Figure 2: Two examples of Tencent CAPTCHAs. (a) “BRED” (b) “DKWK”

Figure 3: Binarized images. (a) “BRED” (b) “DKWK”

Figure 4: Thinned images. (a) “BRED” (b) “DKWK”

(d) Merging: Merge character components obtained in step c) into individual characters.
(e) Recognition: Recognize individual characters by CNN.

The principles and the algorithms involved in our method will be described in detail in the following subsections.

3.2 Preprocessing

The purpose of preprocessing is to highlight the information related to characters in the given image and to weaken or eliminate interfering information. In our method, the preprocessing includes image binarization, image thinning and repairing contour. We take two CAPTCHA images in Figure 2 as examples to illustrate the whole process.

Firstly, image binarization is to highlight interesting targets’ contour and remove noises in the background. Using Otsu’s algorithm [14], we can get binary images, i.e. white-and-black images (see Figure 3).

Image thinning is to process character contours as the skeleton, yet which won’t change the adhesion of characters (Figure 4). The main effect of this step lies in two aspects. First, in a CAPTCHA image, the contours with irregular and uneven thickness are converted into single-pixel-width ones by thinning. It can reduce redundant points. Second, compared to unprocessed characters, the thinned characters have clearer contours without shadows. Thus, the subsequent processing will be simpler, too.

The method in [15] is fast and effectively, thereby we adapt it to thin CAPTCHA images. By image thinning, we obtain hollow characters with one-pixel-thick contours. Furthermore, we find some broken contours.

To get the closed hollow characters, we use the endpoint detection algorithm in [16] to mark the endpoints of each line. Then examine and draw a one-pixel-thick line to connect those neighboring pairs.

After the above steps, we get preprocessed CAPTCHAs, which are binary images containing several closed hollow characters with single-pixel-width contour (see Figure 5).

3.3 Filling

For the thinned images, we fill the closed hollow characters acquired in Section 3.2 to solid ones. The main
purpose of filling is to enrich the character information and prepare for segmentation. During this phase, the key is to fill the character components accurately rather than noncharacter components. The filling method mainly consists of labeling connect regions, removing noise blocks and filling hollow character.

### 3.3.1 Labeling Connect Regions

For closed hollow characters, we use Haralick’s run algorithm [17] to label different numbers of connected components (as Figure 6). Now a label matrix of the same size as the original image matrix is generated. In this matrix, all black pixels are labeled as 0, white pixels are labeled as the ordinal of connected components, such as 1, 2, 3, etc.

### 3.3.2 Removing Noise Blocks and Filling Hollow Character

In this paper, noise block refers to the noncharacter connected component. The noise blocks have two types such as Figure 6(a): One is the connected component inside a character, such as No. 3, 4, 6, 11. The other is the connected component between adherent characters, such as No. 8, 9. They have the common feature: both are circled by character components. As can be seen from Figure 6, the contour of character components is adjacent to the background area; and because noise blocks are surrounded by character components, their contour will not be adjacent to the background area. Therefore, we can distinguish character components from noise blocks according to this feature.

Let the contour $L$ is a set of pixels that are labeled as 0.

$$L = \{ p | p_{x,y} = 0 \}$$

According to whether their contours are adjacent to the background area, the contour $L$ can be divided into two parts: the outer contour and the inner contour. The set of those pixels which are adjacent to the background area is called the outer contour $L_{out}$. Namely, among the neighborhoods of outer contour, there are pixels whose labels of connected components are 1.

$$L_{out} = \{ p | \begin{cases} p_{x,y} = 0 \\ x_1 = 1 || x_2 = 1 || x_3 = 1 || x_4 = 1 \end{cases} \}$$

The part of contour apart from the outer contour called the inner contour $L_{in}$. The pixels of the inner contour are not adjacent to the background area. Among the neighborhoods of inner contour, there are pixels whose labels of connected components are not 1.

$$L_{in} = L - L_{out}$$

In order to obtain connected components circled by inner contour and find out noise blocks accurately, we proposed an 8-neighborhood detection outer contour algorithm (Algorithm 1) to remove the outer contour, and then fill character connected components. Its main steps are as follows.

1. For each contour pixel $p_{x,y} = 0$
2. If there are background pixels in the eight neighbors of the current pixel:
   $x_1 = 1 || x_2 = 1 || x_3 = 1 || x_4 = 1 \Rightarrow p_{x,y} = 1$;
3. Then the current pixel label number is changed to be the background pixel label: $p_{x,y} = 1$; // Remove the outer contour pixels
4. }

In the originallabel matrix, the labels of noise blocks are changed to 1, and reorder character connected components, as shown in Figure 8.
Fill character connected components, and hollow characters become solid ones (see Figure 9).

In summary, our inner-outer contour filling algorithm is summarized in Algorithm 2.

**Algorithm 2**: Inner-outer Contour Filling Algorithm

**Input**: hollow character image;  
**Output**: solid character image.

1. Using Haralick’s run algorithm [17] to label each connected component, and the total connected component is \( \text{sum} \);
2. The label of black pixel is 0;
3. White pixels which belong to \( r \)th connected component are labeled as \( ID(r) = t \), \( t = 1, 2, 3, \ldots, \text{sum} \);
4. Using Algorithm 1 on the copy of the label matrix to detect the labels of noise blocks;
5. \( \text{sum}' = \text{sum} \);
6. for \( r = 2 : \text{sum}/\text{The first connected component is background}
7. \{ If \( r \)-th connected component is noise block,
8. \( ID(r) = 1; /\text{Label the } r \)-th connected component as background
9. \( \text{sum}' = \text{sum}' - 1; \)
10. \}
11.  
12. \}
13. Reorder the connected components whose labels are not 1;
14. for \( t = 2 : \text{sum}/\text{Label the } t \)-th connected component as an object
15. \{ \( ID(t) = 0; /\text{Label the } t \)-th connected component as an object
16. \}

3.4 Segmentation

For the solid character images obtained in the above subsections, we will segment them to individual characters or character components.

Because each connected component corresponds to a character component, our attack uses different labels of connected components to segment the characters.

For each character connected component, first, create an all-1 matrix, which has the same size as the original CAPTCHA image matrix. Then, obtain position coordinates of each pixel from the current character component, and then set pixel value to 0 for those pixels that have same position coordinates in all-1 matrix.

After execution, each character component is stored in a separate image, and the original image is segmented to several character component images (such as Figure 10). If the number of character components obtained by segmenting a CAPTCHA image is equal to the number of individual characters contained in the CAPTCHA, there is no need to perform a merging operation because the image is segmented into individual characters (as Figure 10(a)). Otherwise, it needs to perform a merging operation because the image may be segmented into individual characters and character components (as Figure 10(b)).

3.5 Merging

For the character components obtained in Section 3.4, we need to merge them into individual characters before recognition. As we know, the character components have two features:

(a) The number is small and random. A CAPTCHA image can be segmented to \(5 \sim 7\) character component images.
(b) There are mutual relations. Character components are created from an individual character, and there are inevitable links in structure and order.

According to the above features, we propose a minimum-nearest neighbor merging algorithm:

(a) Due to the small number of components and to avoid merging errors and confusion, only two components are merged each time. Therefore, the number of mergers can be calculated by:

\[
k = S_1 - S_0
\]

where \( k \) represents the number of mergers, \( S_1 \) represents the number of character components produced by segmenting a single image, and \( S_0 \) represents the actual number of characters in a single image.
(b) In general, the one character component with the minimum number of pixels is most likely to be merged. 

\[ C_1 = f(d_1, d_2, \ldots, d_s) \]

where \( d_1, d_2, \ldots, d_s \) are elements in the array \( d \), which used to store the number of pixels corresponding to the ordinal of characters, \( f(d_1, d_2, \ldots, d_s) \) returns the ordinal of the minimum number of pixels to \( C_1 \).

(c) During selecting the other character component to be merged, the inevitable connection among character components in structure and order should be fully considered in order to reduce the blindness of merging.

In terms of structure, the width of two character components together must be less than the maximum width of an individual character.

In terms of order, the character components are segmented from an individual character, so the left and right neighbors of the \( C_i \)th character component to be merged are prioritized as the other component to be merged. So, the ordinal of the other character component to be merged can be calculated by:

\[
C_2 = \begin{cases} 
2 & \text{if } C_1 = 1 \\
S_1 - 1 & \text{if } C_1 = S_1 \\
C_1 - 1 & \text{if } 1 < C_1 < S_1 \& \ w_r \\
C_1 + 1 & \text{if } 1 < C_1 < S_1 \& \ w_r \leq w_l \\
\leq w_l & \text{if } 1 < C_1 < S_1 \& \ w_r > w_l \\
f(d_{C_1-1}, d_{C_1+1}) & \text{if } 1 < C_1 < S_1 \& \ w_r \leq w_l \\
\leq w_l & \text{if } 1 < C_1 < S_1 \& \ w_r > w_l 
\end{cases}
\]

where \( C_2 \) respects the ordinal of another character component to be merged, \( w_l \) is the width of new character component consisted of the character component \( C_1 \) and its left neighbor \( C_1 - 1 \), \( w_r \) is the width of new character component consisted of the character component \( C_1 \) and its right neighbor \( C_1 + 1 \), \( w_l \) is the maximum width of a character component among the CAPTCHA image.

Finally, the proposed minimum-nearest neighbor merging algorithm is summarized in Algorithm 3.

**Algorithm 3: Minimum-nearest Neighbor Merging Algorithm**

**Input:** the multiple character component images segmented by a same CAPTCHA image;

**Output:** the character images to be recognized.

1. Calculate merging times: \( k = S_1 - S_0 \);
2. For \( t = 1 : k \)
3. \{ Find the one character component with the minimum number of pixels in all character components: \( C_1 = f(d_1, d_2, \ldots, d_s) \);
4. Find the other character component \( C_2 \);
5. \{ If \( C_1 = 1, C_2 \) is the right neighbor of \( C_1 \); \( C_2 = 2 \);
6. If \( C_1 = S_1, C_2 \) is the left neighbor of \( C_1 \); \( C_2 = S_1 - 1 \);
7. If \( 1 < C_1 < S_1, C_2 \) is the neighbor with suit width and pixels;
8. \{ If \( w_r > w_l \& w_r \leq w_l, C_2 \) is the left neighbor; \( C_2 = C_1 - 1 \);
9. If \( w_r > w_l \& w_r \leq w_l, C_2 \) is the right neighbor; \( C_2 = C_1 + 1 \);
10. If \( w_l \leq w_l \& w_r \leq w_l, C_2 \) is the neighbor with less pixels;
11. \}
12. \}
13. Merge the character components \( C_1 \) and \( C_2 \);
14. \( S_1 = S_1 - 1 \);
15. Reorder the character components;
16. \}

In order to help the readers to understand Algorithm 3, we describe the merging process in detail using an example. Take for example the character components in Figure 10(b) and the specific parameter values are shown in Table 1. In addition, let \( t \) is the loop times, \( n_{\text{min}} \) is the corresponding ordinal of \( d_{\text{min}} \) which is the minimum element of array \( d \), and \( w_l = 30 \).

When \( t = 0 \), the character components are segmented and sorted. The number of pixels, left boundary, right boundary of each character component are obtained

| Loop time | Parameter          | 1      | 2      | 3      | 4      | 5      | 6      | 7      |
|-----------|--------------------|--------|--------|--------|--------|--------|--------|--------|
|          | Left boundary      | 14     | 42     | 47     | 63     | 86     | 87     | 95     |
|          | Right boundary     | 41     | 53     | 63     | 86     | 89     | 102    | 109    |
|          | Number of pixels   | 358    | 59     | 99     | 228    | 146    | 146    | 58     |
| 1        | Merged width       |        |        |        |        |        |        |        |
|          | Left boundary      | 14     | 42     | 47     | 63     | 86     | 95     |        |
|          | Right boundary     | 41     | 53     | 63     | 86     | 102    | 109    |        |
|          | Number of pixels   | 358    | 59     | 99     | 228    | 159    | 58     |        |
| 2        | Merged width       |        |        |        |        |        |        |        |
|          | Left boundary      | 14     | 42     | 47     | 63     | 86     |        |        |
|          | Right boundary     | 41     | 53     | 63     | 86     | 109    |        |        |
|          | Number of pixels   | 358    | 59     | 99     | 228    | 217    |        |        |
| 3        | Merged width       |        |        |        |        |        |        |        |
|          | Left boundary      | 14     | 42     | 47     | 63     |        |        |        |
|          | Right boundary     | 41     | 63     | 86     | 109    |        |        |        |
|          | Number of pixels   | 358    | 158    | 228    | 217    |        |        |        |
respectively. For example, the number of character components segmented from CAPTCHA image is 7, the number of individual characters in a CAPTCHA image is 4. So it needs 3 times of merging.

When \( t = 1 \), the 5th component is the smallest character component, then \( d_{\text{min}} = 13, n_{\text{min}} = 5 \). First, the minimum character component is in the middle, \( w_l = 26, w_r = 26 \), and they are less than the maximum width of character 30. Next, compare the number of pixels of left and right neighbor components, we found \( d_4 > d_6 \). Therefore, the 5th and 6th character components are merged into a new 5th character component. Accordingly, it needs to update the pixel number, left and right boundary and other related information (see Figure 11). Lastly, the 6th character component information is deleted, and all character components are reordered.

When \( t = 2 \), the 6th component is the smallest character component, \( d_{\text{min}} = 58, n_{\text{min}} = 6 \). The minimum character component is in the end, it and its left neighbor can be merged into the new 5th character component. (see Figure 12).

When \( t = 3 \), the 2nd component is the smallest character component, that is, \( d_{\text{min}} = 59, n_{\text{min}} = 2 \). First, the minimum character component is in the middle, \( w_l = 39, w_r = 21 \). Therefore, the 2nd and 3rd character components are merged into a new 2nd character component. (see Figure 13).

After 3 loops, the merging operation has been completed. Character components are merged into individual characters (show as Figure 14).

3.6 Recognition

After segmentation and merging, all CAPTCHA images are segmented to individual characters, and we can recognize them by some machine learning methods. As well as the method of [12], we also use CNN to recognize these individual characters for high accuracy.

CNN allows to directly inputting the individual character images to be recognized without feature extraction, and it has a certain degree of robustness in displacement, scale, and deformation. As a deep learning network, CNN has been widely used in the fields of CAPTCHA recognition [7, 12, 13], information hiding [18, 19, 20, 21], and so forth.

4. ANALYSIS OF AF&NM METHOD

Compared with the methods of [12] and [13], proposed AF&NM method has some difference. In the preprocessing stage, the thinning operation has been used to reduce redundant breakpoints. In the filling stage, our attack removes noise blocks with inner-outer contour quickly. Next, we acquire candidate characters without redundancy by merging rather than combination. So, our attack has no search stage.

(a) High accuracy in repairing contours

In the method of [12], there are redundant breakpoints, which might produce noise blocks.

In the AF&NM method, the character contour is thinned into single-pixel-width lines. Thus, redundant breakpoints can be correspondingly reduced, and the right breakpoints will be located more accurately.

As shown in Table 2, in the original binary image, redundant breakpoints account for 46.67% of the total number of marked breakpoints, and the redundant breakpoints are only 17.24% in the thinned images. The processing times of nonthinning image and thinned image are 14.08 and 8.64 ms, respectively. So with the reduction of the pixels in thinned contour, the redundant breakpoints are reduced by nearly 30% and the processing time is shortened by approximately 40%. The results show that this method is more effective in improving accuracy and shortening time.

(b) Fast denoising

In [12], all nonblack pixels components are filled; and black contour is removed by dilation operation. If \( m \times m \)
It’s assumed that the analysis of the algorithm is as follows.

A large number of characters to be recognized are produced by the combination method of [12], and the analysis of algorithm is as follows.

It’s assumed that \( S_1 \) represents the number of character components produced by segmenting a single image, \( m \) represents the number of components in character components, \( j \) represents the number of the starting component in a character component, \( num(j) \) represents the number of character components whose starting component is \( j \), \( sum \) represents the number of character components that are merged by components. In addition, the actual number of characters in a single image is \( S_0 \), and for the width limit of the character component, the maximum number of components per character component is \( m' = \lceil S_1/(S_0 - 1) \rceil \). Then the following analysis of the combination methods of [12] and [13] can be known.

When \( j = 1 \), that is, the first component is the start of the combination. Due to the limit of \( m' \), the number of character components produced by combining \( m \) character components is \( n_m \) \((m = 1, 2, \ldots, m')\), thus

\[
\begin{align*}
  n_1 &= 1 \quad m = 1 \\
  n_2 &= S_1 - 1 \quad m = 2 \\
  n_3 &= C^2_{S_1 - 1} \quad m = 3 \\
  &\vdots \\
  n_{m'} &= C^{m'-1}_{S_1 - 1} \quad m = m'
\end{align*}
\]

(7)

So, the total number of character components which start from the \( 1^{\text{st}} \) character component is the sum of the above numbers.

\[
\sum_{m=1}^{m'} n_m = \sum_{m=1}^{m'} C^{m-1}_{S_1 - 1}
\]

Likewise, when \( j = 2 \), when the combination starts from the \( 2^{\text{nd}} \) character component, the total number of character components is \( \sum_{m=1}^{m'} C^{m-1}_{S_1 - 2} \).

When \( j = n - m' + 1 \), when the combination starts from the \((n - m' + 1)^{\text{th}}\) character component, the total number of character components is \( \sum_{m=1}^{m'} C^{m-1}_{S_1 - j} \).

When \( j = S_1 - m' + 2 \), since the remaining character components have been less than \( m' \), so when the combination starts from the \((S_1 - m' + 2)^{\text{th}}\) character component, the total number of character components is \( \sum_{m=1}^{m'} C^{m-1}_{S_1 - j} \).

To sum up the above two cases, we can get the formula for calculating the number of combined characters components in a single image:

\[
num(j) = \begin{cases} 
  \sum_{m=1}^{m'} C^{m-1}_{S_1 - j} & j \leq S_1 - m' + 1 \\
  \sum_{m=1}^{S_1-j+1} C^{m-1}_{S_1 - j} & j > S_1 - m' + 1 
\end{cases}
\]

(9)

\[
sum = 1 + \sum_{j=1}^{S_1-1} num(j)
\]

(10)

Therefore, \( sum \) is the number of candidate characters for a single image. It is more than the actual number of characters \( S_0 \).

In AF&NM method, no redundant characters to be recognized are generated. Let \( n_i \) is the number of images, \( n_0 \) is the number of images to be merged, \( m_i \) is the number of components in the current image, thus the total merging times:

\[
com = \sum_{i=1}^{n_0} (m_i - S_0)
\]

(11)

where if \( n_0 = 0 \), all images are divided into individual characters directly without merging, and the minimum of merging times is 0. If \( n_0 = n_s \), all images are divided into the maximum \( m_1 \) character components, and the maximum of merging times is \( (m_1 - S_0) \times n_s \). Moreover, no matter how many times of merging, the number of characters to be recognized is always the same as \( S_0 \times n_s \), hence the number of the actual individual characters without redundancy.
5. EXPERIMENTAL RESULTS

In order to verify the performance of proposed AF&NM method, we performed experiments on real CAPTCHA data sets. We respectively test the success rate of AF&NM method and its attack speed, and complete contrast experiments with methods of [12] and [13].

5.1 Experimental Environment and Settings

We implemented experiment in MATLAB 2015a, and tested it on a desktop computer with a 3.10 GHz Intel Core i5-2400 CPU, 4 GB RAM, and Windows 10 professional x64.

We have implemented our attack and tested it on two hollow CAPTCHA schemes: Tencent and BotDetect. Tencent is the largest Internet platforms in China. BotDetect was the world’s first commercially available CAPTCHA component in 2004. Therefore, the two representative CAPTCHA schemes with applicability and professional are used to test the performance of our attack method.

For each scheme, we collected from the corresponding website 800 random CAPTCHA images as a training set, and another 200 as a test set. The CAPTCHA samples possess the following features (see Figure 15).

- Every CAPTCHA image consists of 4 capital hollow letters with rotation;
- CAPTCHA images have light background without interference lines and noise blocks.

We ran AF&NM method and the methods of [12] and [13] on Tencent scheme and BotDetect scheme. In [22], Gao et al. pointed out that “We have converted hollow characters into solid characters in advance.” In this way, the method of [13] can be more specific for attack hollow CAPTCHAs.

We implement a CNN as our recognition engine. The CNN’s classical structure adapted from LeNet-5 [23], which has 6 layers in total (see Figure 16).

5.2 Experiments of Attack Success Rate

The success rate is an important indicator to evaluate the effectiveness of the attack method. In experiments, we count the success rate of individual characters and the success rate of single images. The total of images and characters were respectively 200, 800.

For Tencent scheme, the success rate of each phase in AF&NM method shows on Table 4. Considering the mistakes accumulated in all stages, the success rates for individual characters of each phase of AF&NM are 99.25%, 84.00%, 98.50% and 97.25%. Accordingly, the success

| Table 4: Success rates of AF&NM method on Tencent scheme (%) |
|---------------------------------------------------------------|
|                        Preprocessing and filling | Segmentation | Merging | Recognition |
| Success rate of individual characters                          | 99.25        | 84.00   | 98.50       | 97.25       |
| Success rate of single images                                  | 97.00        | 56.00   | 97.00       | 91.00       |

For Tencent scheme, the CAPTCHA image consists of characters with adhesion; For BotDetect scheme, the CAPTCHA image consists of characters with adhesion and overlap.

We ran AF&NM method and the methods of [12] and [13] on Tencent scheme and BotDetect scheme. In [22], Gao et al. pointed out that “We have converted hollow characters into solid characters in advance.” In this way, the method of [13] can be more specific for attack hollow CAPTCHAs.

Figure 15: The CAPTCHA samples used in experiments. (a) Tencent CAPTCHA examples. (b) BotDetect CAPTCHA examples

Figure 16: Network structure of CNN [23]
Table 5: Success rates contrast on Tencent scheme

| Scheme     | Method     | Number of successful characters | Success rate of individual characters (%) | Number of successful images | Success rate of single images (%) |
|------------|------------|----------------------------------|-------------------------------------------|-----------------------------|-----------------------------------|
| Tencent    | Ref. [12]  | 755                              | 94.38                                     | 166                         | 83.00                             |
|            | Ref. [13]  | 756                              | 94.50                                     | 164                         | 82.00                             |
|            | Proposed   | 778                              | 97.25                                     | 182                         | 91.00                             |

Table 6: Success rates of AF&NM method on BotDetect scheme (%)

| Preprocessing and filling | Segmentation | Merging | Recognition |
|---------------------------|--------------|---------|-------------|
| Success rate of individual characters | 80.88 | 66.13 | 80.75 | 89.00 |
| Success rate of single images | 62.00 | 36.50 | 71.50 | 75.50 |

success rate of our attack is still higher than the ones of the methods of [12] and [13].

From Tables 5 and 7, we also can see that the success rate of Tencent scheme is higher than that of BotDetect scheme. Compared to Tencent CAPTCHA, BotDetect CAPTCHA images have more characters with adhesion and overlap, which will have an impact on success rate. This is because overlapping characters produce a large number of character components, which adds to the difficulties for subsequent combinations.

Bursstein et al. [7] suggested that a CAPTCHA scheme is broken when the attacker achieves an accuracy rate of at least 1%. According to the criterion, two hollow schemes are successfully broken by our attack. Thus, the AF&NM method is applicable to the hollow and adherent CAPTCHA and has some effect on the hollow and overlap CAPTCHA. The experimental data show that our method is superior to the other two methods in success rate.

5.3 Experiments of Attack Speed

The attack speed is an important indicator to measure an attack method. The shorter attack time indicates that the security of a CAPTCHA scheme is worse. In addition, improving the attack speed of CAPTCHAAs can reduce the time spent in the massive CAPTCHAAs attack test.

For Tencent scheme, our attack speed of each stage and the proportion are as shown in Table 8. The average time cost for each image is only 48.25ms. The average time cost of segmentation and merging are 2.55ms and 6.96ms. The sum of two average time cost makes up less than 20% of the total time. And nearly half time is consumed in the preprocessing stage, in which thinning and closing contours respectively take up 17.24% and 17.90%. The filling stage, including 3 steps, is easy to implement.

Table 7: Success rates contrast on BotDetect scheme

| Scheme   | Method     | Number of successful characters | Success rate of individual characters (%) | Number of successful images | Success rate of single images (%) |
|----------|------------|----------------------------------|-------------------------------------------|-----------------------------|-----------------------------------|
| Bot-Detect | Ref. [12]  | 543                              | 67.88                                     | 128                         | 64.00                             |
|          | Ref. [13]  | 462                              | 57.75                                     | 106                         | 53.00                             |
|          | Proposed   | 712                              | 89.00                                     | 151                         | 75.50                             |
and accounts for only 8.32%. Besides, because the CNN can automatically extract features and classify, recognition costs 13.78ms and accounts for 28.58% of the total time.

To evaluating the attack speed of AF&NM method, contrast experiments have been conducted with methods of [12] and [13]. In the attack process of Tencent and BotDetect CAPTCHAs, the time cost of three methods is as shown in Figures 17 and 18. For Tencent scheme, the attack speeds with [12] and [13] are respectively 139.29ms and 134.63ms, our attack speed is 48.25ms. For BotDetect scheme, the attack speeds with [12] and [13] are respectively 1032.93ms and 979.94ms, our attack speed is 177.54ms.

According to contrast results, the average attack speed of [12] and [13] are very close to each other and both far slower than our attack speed. Meanwhile, we can see that the attack speed of Tencent scheme is faster than that of BotDetect scheme because the latter has more character components. Comprehensively, AF&NM method is superior to the methods of [12] and [13] in terms of the attack speed.

For Tencent scheme, we compare the attack speed of method of [12] and AF&NM method in detail. From Figure 17, the time cost of filling and segmentation stages of the AF&NM method is 6.56ms per piece, and that of method in [12] is 63.85ms which is 10 times more than AF&NM method. First of all, the time of filling all of the connected components in [12] is 4 times that of only filling characters components regions in AF&NM method. Moreover, the method of [12] performs the dilation algorithm on each stroke to remove the black contour. Its time cost is very large. In addition, the combined time in [12] is nearly 5 times the merging time in AF&NM method. The main reason is that there are a large number of redundant characters in [12] by combining.

As we can see from Table 9, it takes a little more time in the preprocessing stage in AF&NM method, for a thinning operation is extra introduced. However, more preparation will only quicken the speed of doing work. In all stages, the proposed method saves approximately 65.35% of the time than in the method of [12]. Especially in the filling & segmentation stages and combination/merging stages, the attack time is approximately reduced by 89.73% and 80.88% respectively. From

### Table 8: Attack speed per image of each phase of AF&NM method on Tencent scheme

| Breaking time (milliseconds per piece) | Proportion (%) |
|----------------------------------------|---------------|
| Preprocessing Filling                  |               |
| Binary-ization Thinning                |               |
| Closing contour                        |               |
| Removing outer contour                 |               |
| Closing inner contour                  |               |
| Removing noise block                   |               |
| Segmentation                           |               |
| Merging                                |               |
| Recognition                            |               |
| Total                                  |               |

|                | 3.99 | 8.32 | 8.64 | 1.82 | 0.82 | 1.37 | 2.55 | 6.96 | 13.78 | 48.25 |
|----------------|------|------|------|------|------|------|------|------|-------|-------|
| Proportion (%) | 8.27 | 17.24| 17.90| 3.78 | 1.69 | 2.85 | 5.28 | 14.41| 28.58 | 100   |
another perspective, this also fully validates the optimization that thinning has brought to the whole attack.

In short, from the experimental results, we can see the proposed AF&NM method can increase the attack accuracy for images by approximately 8% ~ 22.5% and decrease the attack time by approximately 2/3 ~ 4/5 in average compared with the methods of [12] and [13].

6. CONCLUSIONS

To improve the attack accuracy and reduce the attack time, we gave a simple and novel framework for hollow CAPTCHA attack in this paper. Different from the existing frameworks, our framework supports the classification of character components and individual characters produced by segmentation. In addition, learned from these state-of-the-art attacks, we proposed an attack on hollow CAPTCHA using accurate filling and non-redundant merging. Lastly, the experimental results show that compared with existing typical attack methods on hollow CAPTCHAs, the proposed method has a higher success rate and a superior efficiency in attack. Next, we will further study on seriously distorted CAPTCHA characters.

DISCLOSURE STATEMENT

No potential conflict of interest was reported by the authors.

FUNDING

This work was supported by National Natural Science Foundation of China [grant numbers 61572052, 61602508, 61772549, U1636219, U1736214]; National Key R&D Program of China [grant number 2016QY01W0105, 2016YFB0801303]; Plan for Scientific Innovation Talent of Henan Province [grant number 2018JR0018].

NOTES

1.  https://www.qq.com/
2.  https://captcha.com/

ORCID

Jun Chen  http://orcid.org/0000-0002-5526-6952
Xiangyang Luo  http://orcid.org/0000-0003-3225-4649

REFERENCES

1.  L. V. Ahn, M. Blum, and J. Langford, “Telling humans and computers apart automatically,” Commun. ACM, Vol. 47, no. 2, pp. 57–60, 2004.

2.  J. Yan and A. S. E. Ahmad, “A low-cost attack on a microsoft CAPTCHA,” in Proceedings of ACM Conference on Computer and Communications Security (CCS), Alexandria, USA, Oct. 27–30 2008, pp. 543–54.

3.  P. Sahare and S. B. Dhok, “Robust character segmentation and recognition schemes for multilingual Indian document images,” IETE Tech. Rev., DOI: 10.1080/02564602.2018.1450649, 2018.

4.  B. D. Sarma and S. R. M. Prasanna, “Acoustic-phonetic analysis for speech recognition: a review,” IEEE Tech. Rev. DOI: 10.1080/02564602.2017.1293570, 2017.

5.  W. Liu, X. Luo, Y. Liu, J. Liu, M. Liu, and Y. Q. Shi, “Localization algorithm of indoor Wi-Fi access points based on signal strength relative relationship and region division,” Comput. Mater. Continua, Vol. 55, no. 1, pp. 71–93, 2018.

6.  G. Mori and J. Malik, “Recognizing objects in adversarial clutter: breaking a visual CAPTCHA,” in Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR), Madison, Wisconsin, June 18–20, 2003, Vol. 1, pp. 134–41.

7.  E. Bursztein, M. Martin, and J. C. Mitchell. “Text-based CAPTCHA strengths and weaknesses,” in Proceedings of ACM Conference on Computer and Communications Security (CCS), Chicago, USA, Oct. 17–21, 2011, pp. 125–37.

8.  O. Starostenko, C. Cruz-Perez, F. Uceda-Ponga, and V. Alarcon-Aquino. “Breaking text-based CAPTCHAs with variable word and character orientation,” Pattern Recognit., Vol. 48, no. 4, pp. 1101–12, 2015.

9.  A. S. E. Ahmad, J. Yan, and M. Tayara. “The robustness of google CAPTCHAs,” Computing Science Technical Report CS-TR-1278, Newcastle University, UK, 2011.

10.  G. Moy, N. Jones, C. Harkless, R. Potter. “Distortion estimation techniques in solving visual CAPTCHAs,” in Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR), Washington, DC USA, 27 June–2 July, 2004, Vol. 2, pp. 23–8.

11.  R. A. Nachar, E. Inaty, P. J. Bonnin, and Y. Alayli. “Breaking down CAPTCHA using edge corners and fuzzy logic segmentation/recognition technique,” Secur. Commun. Netw., Vol. 8, pp. 3995–4012, 2015.

Table 9: Attack speed difference between methods of [12] and AF&NM

| Speed difference | Pre-processing | Filling+ Segmentation | Combination/Merging | Recognition | Search | Total |
|------------------|----------------|-----------------------|---------------------|-------------|--------|-------|
| Time (ms)        | 3.49           | -57.29                | -29.45              | -5.57       | -2.22  | -91.04|
| Rate(%)          | 19.99          | -89.73                | -80.88              | -28.79      | -100   | -65.35|

Note: + represents increase; − represents decrease.
12. H. Gao, W. Wang, J. Qi, X. Wang, X. Liu, and J. Yan. “The robustness of hollow CAPTCHAs,” in Proceedings of ACM Conference on Computer and Communications Security (CCS), Berlin, Germany, Nov. 4–8, 2013, pp. 1075–86.

13. H. Gao, J. Yan, F. Cao, Z. Zhang, L. Lei, M. Tang, et al. “A simple generic attack on text CAPTCHAs,” in Proceedings of Network and Distributed System Security Symposium (NDSS), San Diego, California, USA, Feb. 21–24, 2016, DOI:10.14722/Ndss.2016.23154.

14. N. Otsu. “A threshold selection method from gray-level histograms,” *IEEE Transact. Syst. Man Cybernet.* Vol. 9, no. 1, pp. 62–6, 1979.

15. L. Lam, S. Lee, and C. Y. Suen. “Thinning methodologies – a comprehensive survey,” *IEEE Trans. Pattern Anal. Mach. Intell.* Vol. 14, no. 9, pp. 869–85, 1992.

16. R. C. Gonzalez and R. E. Woods. *Digital Image Processing.* Saddle River, NJ, USA: Prentice Hall, August 2007, 3rd ed.

17. R. M. Haralick and L. G. Shapiro. *Computer and Robot Vision.* Middlesex County, MA: Addison-Wesley Longman, 1991, 1st ed.

18. Y. Ma, X. Luo, X. Li, Z. Bao, and Y. Zhang. “Selection of rich model steganalysis features based on decision rough set α-positive region reduction,” *IEEE Trans. Circuits Syst. Video Technol.*, 2018. DOI 10.1109/TCSVT.2018.2799243.

19. Y. Zhang, C. Qin, W. Zhang, F. Liu, and X. Luo. “On the fault-tolerant performance for a class of robust image steganography,” *Signal Process.*, Vol. 146, pp. 99–111, 2018.

20. X. Luo, X. Song, X. Li, W. Zhang, J. Lu; C. Yang, and F. Liu. “Steganalysis of HUGO steganography based on parameter recognition of syndrome-trellis-codes,” *Multimed. Tools Appl.*, Vol. 75, no. 21, pp. 13557–13583, 2016.

21. L. Xiong and Y. Q. Shi, “On the privacy-preserving outsourcing scheme of reversible data hiding over encrypted image data in cloud computing,” *Comput. Mater. Continua*, Vol. 55, no. 3, pp: 523–39, 2018.

22. H. Gao, M. Tang, Y. Liu, P. Zhang, and X. Liu. “Research on the security of Microsoft’s two-layer CAPTCHA,” *IEEE Trans. Inf. Forensics Secur.*, Vol. 12, no. 7, pp. 1671–85, 2017.

23. Y. LeCun, et al. “Handwritten digit recognition: applications of neural network chips and automatic learning,” *IEEE Commun. Mag.*, Vol. 27, no. 11, pp. 41–6, 1989.

**Authors**

Jun Chen received the MS degree in computer science from Guangxi University, Nanning, Guangxi, China, in 2008. She is a PhD candidate at State Key Laboratory of Mathematical Engineering and Advanced Computing. Her research interests include multimedia security and machine learning.

E-mail: chenjun626@gmail.com

Xiangyang Luo received the MS degree and the PhD degree from Zhengzhou Information Science and Technology Institute, Zhengzhou, China, in 2004 and 2010, respectively. He is currently an associate professor and PhD supervisor at State Key Laboratory of Mathematical Engineering and Advanced Computing. In addition, he also recently served as guest editor of some special issues of International Journal *Multimedia Tools and Applications* and *International Journal of Internet*. His research interests include multimedia security.

Corresponding author. E-mail: luoxy_iwu@sina.com

Jianwei Hu received the MS degree in computer science and technology from Zhengzhou Information Science and Technology Institute, Zhengzhou, China. He is currently a lecturer at State Key Laboratory of Mathematical Engineering and Advanced Computing. His research interests include multimedia security.

E-mail: 13592680828@139.com

Xingbo Chen received the MS degree in computer science and technology from Guangxi University, Nanning, Guangxi, China, in 2005. He is currently a professor at State Key Laboratory of Mathematical Engineering and Advanced Computing. His research interests include multimedia security.

E-mail: chenjun626@gmail.com

Dengpan Ye received PhD degree at Nanjing University of Science and Technology in 2005 respectively. He worked as a Post-Doctoral Fellow in Information System School of Singapore Management University. And since 2012 he has been a professor in the School of Computer Science at Wuhan University. His research interests include information hiding and multimedia security.

E-mail: yedp@whu.edu.cn

Daofu Gong received the PhD degree in computer application technology from Zhengzhou Information Science and Technology Institute, Zhengzhou, Henan, China, in 2013. Currently, he is a lecturer at State Key Laboratory of Mathematical Engineering and Advanced Computing. His research interests include multimedia security.

E-mail: gongdf@aliyun.com