CAPTCHaStar! A novel CAPTCHA based on interactive shape discovery

Mauro Conti, Claudio Guarisco, and Riccardo Spolaor
University of Padua, Italy
{conti, claudio.guarisco, riccardo.spolaor}@math.unipd.it

Abstract—Over the last years, most websites where users can register (e.g., email providers and social networks) adopted CAPTCHAs (Completely Automated Public Turing test to tell Computers and Humans Apart) as a countermeasure for automated attacks. The battle of wits between designers and attackers of CAPTCHAs led to current ones being annoying and hard to resolve for users, while still being vulnerable to automated attacks.

In this paper, we propose CAPTCHaStar, a new image-based CAPTCHA that relies on user interaction. This novel CAPTCHA leverages the innate human ability to recognize shapes in a confused environment. We assess the effectiveness of our proposal for the two key aspects for CAPTCHAs, i.e., usability, and resiliency to automated attacks. In particular, we evaluated the usability, carrying out a thorough user study, and we tested the resiliency of our proposal against several types of automated attacks: traditional ones; designed ad-hoc for our proposal; and based on machine learning. Compared to the state of the art, our proposal is more user friendly (e.g., only some 35% of the users prefer current solutions, such as text-based CAPTCHAs) and more resilient to automated attacks.

I. INTRODUCTION

Many public services on the Internet are subject to automated attacks, i.e., an automated program can exploit a vulnerable on-line service, pretending to be a legitimate user. As an example, an attacker may create multiple accounts on an e-mail provider and use them to send spam messages. In the last years, an increasing number of websites adopted countermeasures against these malicious attacks. The most common method consists in allowing access to a service only to users able to solve a CAPTCHA (Completely Automated Public Turing Test to Tell Computers and Humans are Apart). The main purpose of a CAPTCHA is to distinguish a human user from a software robot (from now on also referred as “bot”) that runs automated tasks. In order to do that, researchers leverage the existing gap between human abilities and the current state of the art of software, including also Artificial Intelligence techniques [1]. A CAPTCHA is a program that generates a test, which has the property to be easily solvable by humans, but hardly solvable by a bot [2] (if not employing a significant amount of resources and time). As an example, a bot cannot easily understand the meaning of a sentence (or a picture), while humans can carry out this task with negligible effort.

The design of a good CAPTCHA is not a trivial task. Indeed, both usability to legitimate users and resiliency against automated attacks must be simultaneously satisfied. Attackers of CAPTCHA usually improve automated attacks over time. For this reason, designers use to improve their CAPTCHAs in order to reduce the success rate of novel attacks. Unfortunately, these improvements usually cause a dramatic decrease in usability [3]. Researchers put a significant effort in understanding the trade-off between usability and resiliency to attacks [4]. Also, in order to measure the effective usability of a CAPTCHA, Yan et al. [5] presented a set of metrics that we also consider in this paper: success rate, completion time and ease of understanding.

Contribution. The contribution of this paper is as follow:

• We present CAPTCHaStar, a novel CAPTCHA based on shape recognition and user interaction. CAPTCHaStar prompts the user with some “stars” inside a square. The position of these stars changes according to the position of the cursor. The user must move the cursor, until the stars aggregate in a recognizable shape. Our CAPTCHA leverages the innate human ability to recognize a shape in a confused environment. Indeed, a machine cannot easily emulate this ability [6]. This makes CAPTCHaStar easy solvable by humans while remaining difficult for bots.
• We assessed the usability of our proposal via a user study, considering an extensive set of parameters. The results show that CAPTCHaStar users achieve a success rate higher than 90% for the best combination of parameter values. Furthermore, humans can solve our CAPTCHA in less than 30 seconds (on average).
• We assess the security of our proposal. In particular, we first studied the resiliency of our CAPTCHaStar against traditional attacks (such as exhaustion and leak of the database). Then, we presented some possible ad-hoc attack strategies and discuss their effectiveness against our proposal. Finally, we also assessed the resiliency of CAPTCHaStar against attacks based on machine learning. In all these studies, our solution shown promising results, comparable or even better than state of the art solutions.
• We compare the features of CAPTCHaStar with other existing CAPTCHAs. In particular, we compare our proposal against some of the most famous image-based designs in the literature. For each of these designs, we discuss the protection that it offers against various attack strategies. The results of our comparison underline that our design improves the state of the art.

Our work suggests that CAPTCHaStar is promising for a

1A demo is available at http://captchastar.math.unipd.it/demo.php
practical wide adoption (particularly for mobile devices, where the use of keyboard is more difficult and error-prone [7]), as well as motivate further research along the same direction. 

Organization. The rest of this paper is organized as follows. In Section II we report an overview of the current state of the art. In Section III we describe in details CAPTCHAStar, our novel CAPTCHA. In Section IV we evaluate its usability features, while in Section V we assess its resiliency to automated attacks. Finally, in Section VI, we summarize the contributions of our research, we compare CAPTCHAStar with other image-based CAPTCHAs in the literature, and we discuss possible future work.

II. RELATED WORK

In this section, we discuss the main techniques in the literature to design CAPTCHAs, along with their pros and cons. This section is not intended to be a comprehensive review of the whole literature. Interested readers can refer to the work in [8] for an extensive survey of the state of the art. Henceforth, we refer to a single instance of a CAPTCHA test prompted to a user with the term challenge of the art. In the following sections, we divide CAPTCHAs in two main categories, according to the skill required to solve them: text-based (Section II-A), when they require text recognition, and image-based (Section II-B), when they challenge the user to recognize images. For each category, we briefly describe their usability, traditional attack strategies, and possible countermeasures. Recently, Google proposed noCaptcha, a system that uses an “advanced risk analysis backend that considers the engagement of the user” and prompts the user with either a text-based or an image-based challenge [9]. Unfortunately, there is not yet much technical information available (as well as research papers) to understand how exactly it works, nor to run a proper comparison. As far as we know, the actual CAPTCHA prompted to the user seems independent from the actual “risk assessment”, i.e., even CAPTCHAStar might be used!

A. Text-based CAPTCHAs

A text-based CAPTCHA presents an obfuscated word in the form of an image, and asks the user to read and rewrite it, usually in a text box. Baird et al. [10] proposed the first text-based CAPTCHA in 2002. After this first proposal, several other researchers worked on this kind of design. The main researches focused on improving the resiliency against automated attacks [11, 12, 13]. Currently, text-based CAPTCHAs are the most widely used [14].

Usability features: The first implementations of text-based CAPTCHAs had a very short completion time and high success rate for legitimate users. Unfortunately, the introduction of countermeasures to new automated attacks have dramatically lowered these usability features, highlighting the need for new designs [15]. The instructions to solve text-based CAPTCHAs are really easy to understand. Indeed, they usually do not need any a-priori knowledge from the users, except for the ability to read. Users need to type the answer using a keyboard, except for particular designs (e.g., iCaptcha [16]). Unfortunately, inputting the answer with a keyboard undermine the usability of a CAPTCHA on smartphone or tablet. Indeed, in such devices, a single-handed touch-based interaction style is dominant [17].

Attacks and countermeasures: The most common way to automatically solve text-based CAPTCHAs is to use an OCR (Optical Character Recognition) software. In the past few years, CAPTCHA designers and attackers took part in a battle of wits. This battle led to an improvement of OCR software, hence making OCR a very effective threat [18] to text-based CAPTCHAs. Another effective approach to solve CAPTCHA is the so-called relay attack: some companies sells real-time human labor to solve CAPTCHAs [19]. This approach has a really high success rate and it costs only one U.S. dollar per thousand CAPTCHAs [3].

Looking at the literature, the attack strategies against text-based CAPTCHAs can be classified as follows:

A01) Forward the challenge to paid or unaware humans that solve it (i.e., relay attack).
A02) In case the answer is a word of sense, use OCR technology combined with a dictionary.
A03) Use OCR software on a single character separately.
A04) Segment the word, in order to obtain a single image for every character.
A05) Remove smaller lines that can be added as an obstacle to the segmentation process.
A06) Fill hollow spaces inside each character, to improve OCR effectiveness.
A07) Repair characters outline by fixing broken lines. This method leverages on analyzing the distance between pixels.

Attackers may combine two or more of these attack strategies in order to achieve an higher success rate.

CAPTCHA designers reacted to these attacks proposing several improvements to mitigate their effectiveness. Some examples follow (between parenthesis we indicate the attack for which the mitigation strategy could be effective):

- Add more layers of interaction between user and CAPTCHA (could be effective for threat A01 above).
- Add more distortion to the letters, e.g., warping, scaling, rotating (against A03 and A06).
- Use of English-like words (for the sake of usability) or totally random words (against A03).
- Add more pollution to the image, e.g., ticker lines over the letters, confusing background (against A04 and A05).
- Increment noise, e.g., degrading the quality of the resulting image (against A07).

Unfortunately, some of these mitigation strategies have been shown to be ineffective [20, 21].

B. Image-based CAPTCHAs

Image-based CAPTCHAs usually ask the user to recognize an image or to interact with on-screen objects to find a solution. Unlike text-based CAPTCHAs, every image-based
design is substantially different from each other. For this reason, a user who faces a CAPTCHA design for the first time needs a little more effort to understand how it works. Studies suggest that image-based CAPTCHAs are more appreciated by users [22]. Indeed, image-based CAPTCHAs usually have a high success rate and they are less challenging than text-based ones [23]. In the following, we report some examples of image-based CAPTCHA that we could group in three subcategories: static, motion, and interactive.

One of the representative static image-based CAPTCHAs is Asirra [24]. Asirra asks the user to distinguish between cats and dogs, on twelve different photos randomly taken from an external website. Another static image-based CAPTCHA is Collage [25]: it requests to click on a specific picture, among six pictures randomly taken. Deep CAPTCHA [26] prompts the user with six 3D models of real world objects and it asks to sort them by their size.

Some designers focus on CAPTCHA that requires video recognition rather than static image recognition. For example, Motion CAPTCHA [29] shows the user a randomly chosen video from a database, then it asks the user to identify the action performed by the person in the video. Similarly, YouTube Videos CAPTCHA [27] leverages on real video in YouTube service, and it asks the user to write three tags related to the content of the video.

Interactive CAPTCHAs mitigate the relay attack threat. For example, Noise CAPTCHA [28] presents a transparent noisy image overlapped to a noisy background. The user needs to drag this image until he can recognize a well formed text. Cursor CAPTCHA [29] changes the appearance of mouse cursor into another random object. The user needs to overlap the cursor on the identical object placed in a random generated image. Jigsaw CAPTCHA [22] reprises the classical jigsaw puzzle. Indeed, the user needs to correctly rearrange the pieces of a jigsaw. Finally, PlayThru [30] asks the user to solve a randomly generated mini-game. These mini-games require to drag objects on their correct spots.

**Usability features:** Since image-based CAPTCHAs are different from each other, the usability may change depending on the considered design. Usually, image-based CAPTCHAs do not require to type on a keyboard. For this reason, smartphone and tablet users prefer image-based CAPTCHAs over text-based ones [7]. The instructions for each different CAPTCHA design are usually short and intuitive. Finally, on the server-side, resources required and setup time should be as small as possible. However, some image-based CAPTCHAs need many external libraries and may require a large amount of computational power (for example, the design proposed in [31] requires more than two minutes to generate a single challenge).

**Attacks and countermeasures:** The attacks designed to automatically solve image-based CAPTCHAs are usually very specific, i.e., the attacker has to exploit weak points of each specific CAPTCHA design. The main attack strategies used against image-based CAPTCHAs are the following (to avoid confusion and have a unique numbering for attack strategies—also considering the ones for text-based CAPTCHAs—we continue from A08):

A08) Some CAPTCHAs (especially the ones based on games) hide the solution on client-side. An attacker that performs an indirect attack is able to retrieve the solution.
A09) Some CAPTCHAs rely on a pool of pre-computed challenges, stored in a database. A malicious attacker can perform the exhaustion of the database using real humans (e.g., via Amazon Mechanical Turk [7]).
A10) Similarly, an attacker can make queries to a leaked database to identify the solution of a challenge.
A11) An attacker can use machine learning techniques (e.g., Support Vector Machine) to recognize the objects that compose a challenge and solve it.
A12) In case of a limited number of possible answers, an attacker could simply rely on a random chance obtaining a decent success rate.
A13) CAPTCHAs solvable with a single interaction are prone to pure relay attacks. Indeed, attackers can simply send a screenshot of the challenge to an external paid human.
A14) Given a heavily interactive CAPTCHA, a bot can synchronously relay the data stream from the server over to an human solver, and then relay back the input of the user to the server. This strategy is defined as stream relay attack [32].

Several improvements are possible to mitigate the previous weaknesses. Some examples follow:

- Use code obfuscation or encryption (against A08).
- Use Web crawlers to have a self-growing database (against A09).
- Process objects stored in the database before presenting them in the challenge. This makes it infeasible to match the original object with the one presented in the challenge (against A09 and A10).
- Enlarge the search space in order to increase the computational cost to find a solution (against A11).
- Increase the number of possible answers (against A12).
- Analyze the behavioral features, identifying suspicious pattern of movement [33] (against A13 and A14).

### III. OUR PROPOSAL: CAPTCHASTAR

In this section, we present CAPTCHAStar, a novel image-based CAPTCHA. The aim of our proposal is to provide a high level of usability, while improving security. In the following, we first provide a high level overview of the system (Section III-A), then we discuss the actual implementation of the prototype (Section III-B).

#### A. CAPTCHAStar overview

Our CAPTCHA prompts the user with several small white squares, randomly placed inside a squared black space. From now on, we refer to a single white square as a **star**, and to the squared black space as the **drawable space**. The position of each star changes according to the current coordinates of the cursor, inside the drawable space. Given a challenge, we
define as state a snapshot of the stars location on the drawable space, relative to a specific cursor position. The challenge asks the user (who wants to be recognized as a human) to change the position of the stars, by moving his cursor, until he is able to recognize a shape (which is not predictable). In particular, CAPTCHAStar creates such a shape starting from a picture randomly chosen among a huge set of pictures. Figure 1a illustrates an example of a picture with ideal features: two colors and a limited number of small details.

Our system decomposes the selected picture in several stars using a sampling algorithm (described later in Section III-B). For each star, the system sets its movement pattern, in a way such that the stars can aggregate together, forming the shape of the sampled picture. This happens only when the cursor is on a secret position. We refer to that position as the solution of the challenge. In general, a single CAPTCHAStar challenge can include more than one shape, each of them having its own solution (i.e., secret position of the cursor), at which becomes visible.

When the position of the cursor is far from the solution, the stars appear randomly scattered on the black space. Figure 1b shows an example, obtained from the stars that compose the picture in Figure 1a. The user has to move the cursor inside the drawable space until he recognizes a meaningful shape. As the distance between the cursor and the solution decreases significantly, the stars aggregate together in a more and more detailed shape (see Figure 1c). The user needs to adjust the position of the cursor, until he is confident that the resulting shape is detailed enough (see Figure 1d). Finally, the user confirms the current cursor position as his final answer. The system compares the solution with the final answer (allowing a small distance of error), eventually assessing whether the interaction was made by human.

To make the solution of the CAPTCHA more difficult for a bot, in addition to the stars forming the original shape (original stars), we add also noisy stars: i.e., stars that will be in random position when the shape is complete. The number of the noisy stars can be tuned according to a specific parameter.

The system stores on server-side the solution of the challenge as the pair of coordinates (sol_x, sol_y). Our system repeats these steps for a number of times equal to the value of the parameter NSol.

1) Generation of a challenge: The steps to generate a challenge are as follows:
   i) Picture selection and pre-process.
   ii) Picture decomposition.
   iii) Trajectory computation.

Picture selection and pre-process. Our system randomly chooses one of the pictures from the pool, and resizes it according to the value of the parameter PicSize. If the Rotation parameter is enabled, CAPTCHAStar rotates the picture by a random degree. At this point, our system converts the picture in black and white (i.e., binarization).

Picture decomposition. The sampling algorithm first divides the picture in 5x5 pixel tiles, then it counts the number of black pixels inside each tile. A tile will result in an original star when it matches one of the following conditions:
   • If the tile is filled with black pixels (i.e., having 5x5 = 25 black pixels), our system generates an original star and places it at the center of the tile.
   • If the tile has a number of black pixels between 9 and 24, our system generates an original star and places it in a position that is shifted from the center of the tile, toward the position where there are the majority of black pixels.

Our system places the final shape composed by stars inside the drawable space, in a random position (such that all the original stars lie inside).

Trajectory computation. We define the solution sol of the challenge as the pair of coordinates (sol_x, sol_y). Our system generates sol_x and sol_y at random, within the range of [5, 295]. We adopted such range for the sake of usability. In particular, this guarantees that the solution will not appear on the edges of the drawable area (which is 300x300 pixel). For each original star i, our system also defines \( (P_{xi}, P_{yi}) \) as the coordinates of the position that the star \( i \) takes when the cursor is in

\[ \text{Number of possible solutions (i.e., secret positions) of the challenge}. \]
coordinates \((sol_x, sol_y)\). For each star \(i\), our system randomly generates four coefficients \((m_{ix,x}, m_{ix,y}, m_{iy,x}, m_{iy,y})\), that relates the coordinates of the star with the coordinates of the cursor: \(m_{ib}\), associates the coordinate of the star \(i\) in axis \(a\), with the coordinate of the cursor in axis \(b\). The values of these coefficients are picked in the range \([-\frac{\delta}{\pi}, \frac{\delta}{\pi}]\) (we remind that \(\delta\) is the sensitivity value). Our system computes a pair of constants, \((C^*_x, C^*_y)\), for each original star \(i\) as follows:

\[
C^*_x = P_x^i - sol_y \cdot m_{ix,y} - sol_x \cdot m_{ix,x}, \\
C^*_y = P_y^i - sol_y \cdot m_{iy,y} - sol_x \cdot m_{iy,x}.
\]

CAPTCHAStar generates the noisy stars in a similar way, but their coordinates \((P_x^i, P_y^i)\) having random values. The number of noisy stars is equal to the percentage \(\psi\) of original stars. Henceforth, we define as trajectories parameters of star \(i\), the following set of parameters: \(m_{ix,x}, m_{ix,y}, C^*_x, m_{iy,x}, m_{iy,y}, C^*_y\).

All the information that the client needs from the server in order to calculate the position of the stars, whenever the user moves his cursor, is the trajectories parameters. We underline that noisy and original stars are mixed together, i.e., they are indistinguishable from client side.

2) Presentation of a challenge: Whenever the user moves the cursor, our system uses the cursor coordinates \(cur = (cur_x, cur_y)\) to compute the new coordinates of each star \(i\), as follows:

\[
x^i = m_{ix,y} \cdot cur_y + m_{ix,x} \cdot cur_x + C^*_x, \\
y^i = m_{iy,x} \cdot cur_x + m_{iy,y} \cdot cur_y + C^*_y.
\]

When the user confirms his answer (e.g., with a mouse click), the client passes \(cur\) to a simple server-side script, via HTTP GET parameter. For the sake of usability, on mobile devices the submission of the answer is performed by tapping on a button, which is external to the drawable space.

That script calculates \(\Delta\) as the euclidean distance between \(sol\) and \(cur\). We define usability tolerance as a threshold, in terms of euclidean distance from \(sol\). When the value \(\Delta\) is below the usability tolerance, the system considers the test as passed (failed otherwise). From our experiments, we found that a reasonable value for usability tolerance is close to five. We highlight that the position of each star varies linearly with the movement of the cursor. For this reason, humans can easily build a mental map of the stars behavior, hence moving the cursor toward the position that is closer to a real shape.

The average time for generating a challenge (following the process as described above) is less than two seconds, using a PC with 3.0 GHz AMD Athlon Dual Core Processor and 1 GB memory. This suggests that our prototype can handle efficiently a wide number of requests, even with low cost hardware resources.

IV. Usability

In order to evaluate the usability of CAPTCHAStar, we ran a preliminary user study considering the metrics proposed in \[5\], and an exhaustive set of parameter combination. We built a survey composed by 8 different tests: 6 CAPTCHAStar challenges (named from T1 to T6) and 2 text-based ones (T7 and T8). Table \[6\] reports the value of parameters for the tests from T1 to T6.

| Test | T1 | T2 | T3 | T4 | T5 | T6 |
|------|----|----|----|----|----|----|
| \(\psi\) | 0% | 70% | 70% | 10% | 0% | 250% |
| \(\delta\) | 5 | 5 | 5 | 5 | 5 | 5 |
| NSol | 1 | 1 | 1 | 2 | 3 | 1 |
| Rotation | Off | Off | On | Off | Off | Off |

TABLE I: Values of parameters \(\psi\), \(\delta\), NSol and Rotation.

The last two tests are text-based CAPTCHAs from re-Captcha, with one and two words (i.e., T7 and T8, respectively). In order to minimize the learning effect \[37\], we prompt the user with the eight tests selected in a random order. At the beginning of the survey, we asked some demographic information: age, gender, nationality, level of education, years passed using Internet, and frequency of internet use. At the end of the eighth test, we asked the participants to: (i) rate the ease of understanding; (ii) indicate if they prefer our proposal or text-based CAPTCHAs; (iii) leave us any suggestion. We recruited the participants with an invitation (including a public link to the survey) that we broadcast on mailing lists and on social networks (i.e., Facebook, Google+, Twitter, and LinkedIn). We did not give any reward for the participation. All the participants took the survey unsupervised using their own devices, in order to recreate the natural conditions of use of CAPTCHAStar.

Results and discussion: More than 230 users took part in our survey (81% male and 19% female), with average age of 25.5 and an education level distributed as follows: 32% high school diploma, 29% bachelor degree, 26% master degree, 9% PhD, and 4% none of the previous ones. The totality
of participants uses internet daily, 49% from 5 to 10 years, 33% for more than 10 to 15 years, 28% for more than 15 years. The majority (90%) of participants are Italians. We asked users to rate the ease of understanding on a scale from 1 (very simple) to 10 (very difficult), and the results show an average value of 4.53, with standard deviation of 2.53. Among all the participants, only 35% of them preferred traditional CAPTCHAs rather than CAPTCHAStar. Table II reports the success rate and the average solving time for each of the eight challenge described above.

| Test | T1 | T2 | T3 | T4 | T5 | T6 | T7 | T8 |
|------|----|----|----|----|----|----|----|----|
| Succ. Rate (%) | 78.7 | 90.2 | 90.6 | 48.1 | 85.1 | 76.6 | 62.7 | 48.9 |
| Difficulty | 1.9 | 2.4 | 2.6 | 2.9 | 3.1 | 2.4 | 2.7 |
| Succ. Std | 14.4 | 17.5 | 22.2 | 54.1 | 30.2 | 28.5 | 11.0 | 14.9 |
| Fail Std | 9.8 | 9.3 | 15.8 | 33.5 | 20.2 | 19.7 | 5.4 | 6.1 |

TABLE II: Survey results.

In most cases, when considering failed tests, the average completion time is higher than successfully passed ones. In general, the standard deviation of these completion times is quite high (more than 25 for most of the tests): this is due to the different skills of the users. We highlight that all the CAPTCHAStar tests (i.e., T1 to T6) have a success rate higher than the one of T8 (i.e., text-based with two words), and only for T4 the success rate is lower than the one of T7 (i.e., text-based with one word). In particular T2 shows a success rate that is some 90%, which is higher than the 84% for text-based CAPTCHAs reported in [3]. We underline that in our text-based CAPTCHAs T7 and T8 (where we used current reCaptha used by Google), we observed a success rate of 62.7% (for the simpler test with only one word). In figures 2a and 2b we report in the domain of time the percentages of participants that solved and failed a challenge, respectively.

We can observe that these users improve their performance to solve CAPTCHAStar challenges as they repeat the survey (i.e., increasing the success rate and decreasing of the completion time), while their performance on text-based challenges (T7 and T8) remains quite the same. These preliminary results indicate that as users gain more confidence with CAPTCHAStar, the completion time significantly decreases.

V. RESILIENCY TO AUTOMATED ATTACKS

An important feature of a good CAPTCHA is the resiliency to automated attacks. In the following, we investigate the resiliency of our proposal against several attacks, such as traditional attacks (Section V-A); automated attacks using ad-hoc heuristics (Section V-B) and attacks based on machine learning (Section V-C).

A. Traditional attacks

In this section, we discuss how CAPTCHAStar withstands traditional attack strategies for CAPTCHAs (we listed those strategies in Section II-B).

- **Indirect Attack** (A08): An indirect attack is not feasible, since all the information about the solution are not available on the client-side. CAPTCHAStar generates randomly the challenge on the server-side, and passes to the client only the description of the behavior of each star with respect to the current cursor position. We remind that the coordinates \((sol_x, sol_y)\), corresponding to the solution of the challenge, are never revealed to the client. Our system checks the correctness of the final answer on the server-side, only after the user confirms it.

- **Exhaustion of Database** (A09): Our system generates a challenge starting from a .png picture, randomly chosen among more than five thousand candidates. Moreover, this database can be automatically enriched with the help of a web crawler, but we consider this as a future work.

- **Leak of Database** (A10): An attacker who tries to match a challenge with its original picture faces a more complex...
problem than actually solving the challenge. Indeed, the attacker has to solve the challenge in order to input the complete shape to a matching algorithm. Moreover, we highlight that during the generation phase the system applies some transformations to the original picture.

- **Machine Learning** (A11): In order to understand the feasibility of this attack, we actually trained a classifier to beat our CAPTCHA. Results suggest that this approach could be a serious threat, but it needs an unpractical amount of time and resources to be performed. We provide more detailed study about this specific attack in Section IV-C.

- **Random Choice** (A12): For the sake of usability, CAPTCHAStar accepts as a correct answer also the neighborhood of the solution (according to the value of usability tolerance parameter). Nevertheless, the probability of random guess is some 0.09% with usability tolerance equal to 5.

- **Pure Relay Attack** (A13): The solution discovery requires constant interaction with the CAPTCHA. For this reason, a single screenshot sent to a third party is surely not enough to put in practice a relay attack.

- **Stream Relay Attack** (A14): As we introduced in Section II-B, a stream relay attack needs to synchronously stream the current state to a human third party. CAPTCHAStar needs a constant and immediate feedback system on each cursor movement. Streaming a large number of frames over a (usually) slow connection between the bot and the solvers machine may reduce solving accuracy and increase the response time. Unfortunately, this attack strategy remains the most effective against CAPTCHAs (including our proposal).

### B. Automated attacks using ad-hoc heuristics

In this section, we describe the design of a CAPTCHAStar automatic solver, in order to deeply test the reliability of our design. While retrieving all the possible states of a challenge is a trivial task (an attacker can simply take a snapshot for each cursor position), identifying the specific state corresponding to the solution is not simple. Indeed, the core task of an automatic solver is to recognize the presence of a shape in a given state. In the following, we report some ad-hoc heuristics we came up with to perform this task (of course, we cannot exclude better solutions that could be proposed in the future).

We created a program capable of generating every possible state, and assign a score to each state using an heuristic. Given a state, the aim of the heuristic is to quantify the dispersion of the stars. We consider as a candidate solution the state that minimize the score given by the applied heuristic. The total number of states that the automatic solver has to evaluate is equal to 84100 (i.e., 2902). The computational cost of the attack can be really high, according to the implemented heuristic. We implemented the automatic solver and the heuristics described below in C programming language. For each heuristic, we evaluate the automatic solver in terms of success rate and average execution time for at least 250 challenges.

For this evaluation, we use the same value of parameters as in test T2 in the usability survey in Section IV-V (we chose these parameters since test T2 was the test with the highest success rate). In this evaluation, we used a PC with 2.3 GHz Intel Pentium B970 and 4 GB memory.

1) **Minimize height/width of stars (MinSize):** We denote by \(S^k\) the challenge state generated when the cursor is in position \(k\), in coordinates \((x_k, y_k)\). We also consider \(x_s\) and \(y_s\) the \(x\) and \(y\) coordinates, respectively, of star \(s\). The heuristic is defined as follows:

\[
\text{MinSize}(k) = (\max_{s \in S^k} x_s - \min_{s \in S^k} x_s) + (\max_{s \in S^k} y_s - \min_{s \in S^k} y_s).
\]

When \(\psi = 0\%), this heuristic has more than 90% of success rate. The addition of a few noisy stars to the challenge completely nullify the effectiveness of this heuristic (i.e., success rate of 0% with only two noisy stars). The algorithm has a very low computational cost. We recorded an average execution time of 10 seconds.

2) **Minimize the distribution (MinDistribution):** The main idea under this heuristic consists in dividing the drawable space in tiles. Indeed, this heuristic evaluates the stars dispersion on each tile singularly. Henceforth, we define a matrix \(M^k\) as the matrix of pixels in the drawable area, after the drawing process of the state \(S^k\). Each cell is defined as follows:

\[
M_{i,j}^k = \begin{cases} 
1 & \text{if pixel } (i,j) \text{ is white;} \\
0 & \text{otherwise.}
\end{cases}
\]

We divide \(M^k\) in a set \(T^k\) of 144 squared tiles (i.e., \(t \in T^k\) is a sub-matrix of \(M^k\)), each with a side of 25 pixels. We define the score of a single tile \(t \in T^k\) as:

\[
f_{\text{score}}(t) = \left| 2 \sum_{i=1}^{25} \sum_{j=1}^{25} t_{ij} - 25^2 \right|.
\]

The heuristic is defined as:

\[
\text{MinDistr}(k) = \sum_{t \in T^k} f_{\text{score}}(t).
\]

The value of sensitivity parameter (\(\delta\)) heavily affects the effectiveness of this heuristic. Indeed, when \(\psi = 70\%\), the attack that uses this heuristic achieves a success rate of 2.7\%, with \(\delta = 5\). On the other hand, the success rate significantly decreases to 0.07\%, when we increase the value of \(\delta\) to 7. The computational cost of this heuristic is slightly higher than the previously discussed MinSize. The average time is 65 seconds.

3) **Minimize the sum of distances (MinSumDist):** This heuristic aims at recognizing when stars are clustered together, even in different groups. We define \(d(s_1, s_2)\) as the euclidean distance between the stars \(s_1\) and \(s_2\). The heuristic is defined as follows:

\[
\text{MinSumDist}(k) = \sum_{s \in S^k} \min_{r \in S^k} d(s, r).
\]

When \(\psi = 70\%\) and \(\delta = 7\), the success rate of this strategy is 0.56\%. The computational cost of this heuristic is higher.
than MinDistribution: we observed an average execution time of 12 minutes and 45 seconds.

4) Minimize the sum of all distances (AllSumDist): We modify the previously discussed heuristic in order to consider all distances. The heuristic is defined as:

\[ \text{AllSumDist}(k) = \sum_{s \in S^k} \sum_{r \in S^k} d(s, r). \]

This heuristic is the most effective, with a success rate of 1.92% on \( \psi = 70\% \) and \( \delta = 7 \). However, this heuristic has a very high computational cost. We recorded an average execution time of more than 25 minutes.

In Figure 4a and Figure 4b we report behavior of the success rate while varying of \( \delta \) and \( \psi \), respectively, for the heuristics described above. From Figure 4a we observe that for \( \psi = 70\% \), the success rate is always smaller than 3%. From Figure 4b we observe that with a small level of noise the success rate would be significant (i.e., 40\% for \( \psi = 10 \) and \( \delta = 7 \)). However, increasing the noise level effectively mitigates this problem (i.e., for \( \delta = 7 \) and \( \psi > 50\% \), the success rate is always smaller than 5%).

![Comparison of success rates on variations of \( \delta \) and \( \psi \).](image)

(a) \( \psi = 70\% \), varying \( \delta \).
(b) \( \delta = 7 \), varying \( \psi \).

Fig. 4: Comparison of success rates on variations of \( \delta \) and \( \psi \).

From Table III we observe that even if the variation of execution time is very high (from 10 seconds of MinSize, to 1500 of AllSumDist ones), the success rate is always smaller than 2%.

| Strategy | MinSize | MinDistribution | MinSumDist | AllSumDist |
|----------|---------|-----------------|------------|------------|
| Time (s) | 10      | 65              | 765        | 1500       |
| Succ. Rate | 0.00%  | 0.07%           | 0.50%      | 1.92%      |

TABLE III: Execution time and Success (\( \psi = 70\%; \delta = 7 \)).

C. Attacks based on machine learning

In order to assess the resiliency of CAPTCHAStar against machine learning-based attacks, we designed a tool that tries to find the solution of a challenge. In the following, we report in details how we built such a tool. In particular, in Section V-C1 we introduce the methodology we followed to extract features from a challenge state. In Section V-C2 we explain the training phase of the classifiers. In Section V-C3 we describe the actual attack and we evaluate its performance. We implemented this tool using scikit-learn libraries.

![Scikit-learn: machine learning in Python,](http://scikit-learn.org/)

1) Features extraction: We recall that given a state \( S^k \), we obtain its Boolean matrix \( M^k \), as defined in Section V-B. A classifier is a supervised learning algorithm \([38]\) that requires a training set. The examples in the training set are labeled with the class they belong to. After the training phase, a classifier should be able to identify to which class belongs a new unlabeled example. All the examples must have a fixed number of features. Therefore, we need to represent a state of a challenge with a vector of \( n \) features. The methodology we follow for features extraction derives from the procedure described in [39], but with a significant difference in the computation of features values. Indeed, we need to represent Boolean matrices (i.e., black and white) instead of gray-scale matrices. The idea consists of dividing a matrix \( M^k \) into a set \( T^k_\omega \) of squared tiles. The parameter \( \omega \) is the amount of pixels in a tile side. For each considered value of \( \omega \), we build a vector \( F_\omega = < f_1, ..., f_n > \) of reference tiles. In particular, we empirically select \( n = 3\omega \). From now on, we refer to \( h(t_1, t_2) \) as the Hamming distance between two Boolean matrices \( t_1 \) and \( t_2 \) (i.e., two tiles). The tiles in the vector \( F_\omega \) must be different from each other. For this reason, we apply k-mean clustering method on a training set of candidate tiles with side \( \omega \), using \( K = n \) and \( h \) as similarity metric. At the end of clustering procedure, we obtain a vector \( F_\omega \), where \( f_i \in F_\omega \) is a tile that represents the centroid of the \( i \)-th cluster. We compute the values of a vector \( D^k_\omega = < d^k_1, ..., d^k_n > \), where each value is defined as follows:

\[ d^k_i = |\{ t \in T^k_\omega : f_i = \text{argmin}_{l \in F_\omega} h(l, t) \}|, \ \forall i = 1, ..., n. \]

The values in the features vectors \( D^k_\omega \) are normalized, from 0 to 1. In practical terms, for a fixed \( \omega \), this procedure produces a vector of features \( D^k_\omega \), starting from a cursor position \( z \) that corresponds to the state \( S^k \).

2) Classifiers training: We train Random forest (RF) and Support Vector Machine (SVM) classifiers with 4000 random challenges (with \( \psi = 70\% \) and \( \delta = 7 \)). For implementation of the classifiers, we use RF classifier with 60 Decision trees estimators, and we use SVM classifier with Radial Basis Function (RBF) as kernel function. We use these classifiers to perform a binary classification, i.e., they recognize examples of two classes: solution and non-solution. For each challenge, we generate 400 states (this means a training set of \( 1.6 \cdot 10^6 \) examples). We train a classifier for each value of \( \omega \).

We underline that an attacker have to build this training set manually, i.e., we have access to the exact solution of a challenge, while an adversary can retrieve this information only by solving the challenge legitimately. Moreover, we also know the value of usability tolerance parameter, providing to the classifier also the neighbors of the solution.

3) Attacks design and performance: In the following, we discuss the design of two attacks that use the classifiers trained in the previous phase. We evaluated the attacks varying the parameter \( \omega \). For the sake of attack feasibility (in terms of both memory and time), we limited the research space to a
subset of $K$ possible cursor positions coordinates, defined as:
$$K_\lambda = \{(\lambda x, \lambda y) : \forall x, y \in \mathbb{N} \cap [0, 300/\lambda]\}.$$

We set the parameter $\lambda = 5$ pixels (i.e., the same value for the usability tolerance), in order to ensure that we have at least one solution among all the states. After this procedure, we obtain a set of $K_\lambda$ cursor positions. For each classifier with a specific $\omega$, we define $C_\omega$ as the function that evaluates the probability that a given state $S^k$ belongs to the class solution. We recall that a challenge admits only one answer, and it is final and irrevocable. We observed experimentally that the distribution of values for function $C_\omega$ often presents multiple local maximum and large plateau. For this reason, an attacker must find the cursor position $k_{sol}$ that corresponds to a global maximum for the function $C_\omega$:
$$k_{sol} = \arg\max_{k \in K_\lambda} C_\omega(S^k).$$

In this evaluation, we ran the attacks on a test set of 200 challenges (with $\psi = 70$ and $\delta = 7$), for each considered value of $\omega$. We executed the attacks on a high end PC with a 3.16 GHz Intel Xeon X5460 and 32 GB of RAM. Figure 5a and Figure 5b report the success rate and the average execution time to perform these attacks, respectively. The attack with the best success rate uses the SVM classifier with $\omega = 15$, and it achieves a success rate of 78.1% (as reported in Figure 5a). The time required to build the features vectors $D^k_\omega$, $\forall k \in K_\lambda$, remains stable at around 340 seconds. On one hand, the time required to compute the probability values $C_\omega(S^k)$, $\forall k \in K_\lambda$, increases linearly using SVM classifier, according to the value of $\omega$. On the other hand, this time remains under two seconds using RF. This means that an attack on a single challenge will have some 78% of success rate, but it will require 421 seconds to be performed. We recall that a human user can solve a challenge with more than 90% of success rate in an average time of 27 seconds (56 seconds in the worst case). Therefore, the problem for a bot of automatically recognizing a solution state of a challenge of CAPTCHAStar is hard to treat in a limited amount of time and resources. We underline that, as recently reported in [21], machine learning based attacks achieve some 50% in only two seconds against Baidu and eBay CAPTCHAs.

VI. CONCLUSIONS AND FUTURE WORKS

In this paper, we proposed CAPTCHAStar, a novel image-based CAPTCHA that leverages the innate human ability to recognize shapes in a confused environment [40]. Our preliminary studies demonstrate that our proposal meets both security and usability requirements for a good CAPTCHA design.

We described in detail our prototype and introduced the parameters involved in challenge generation. Our practical security assessment suggests that the optimal value for parameters are $\psi = 70\%$ and $\delta = 7$. Data collected from our user study confirmed the usability of our proposal. Indeed, users were able to obtain a success rate of higher than 90%, which is better than the success rate of CAPTCHAs currently used in websites [3] such as mail.ru and Microsoft. Finally, the majority of the users who participated in our survey preferred CAPTCHAStar over classical text-based CAPTCHAs. These results motivate further research in this direction.

In this paper, we also assessed the resiliency of CAPTCHAStar against traditional and automated ad-hoc attacks. Indeed, these attacks were shown to be ineffective adopting parameters $\psi = 70\%$ and $\delta = 7$. We also performed an attack leveraging a machine learning classifier, that we optimized by reducing as much as possible the research space. Despite this optimization of the attack and its execution on a high end PC, the resulting average execution time is still unacceptable, i.e., more than six minutes to find the solution for a single challenge (with a success probability of 79%). We recall that users are able to complete CAPTCHAStar challenges in an average time of less than 27 seconds (with a success probability of some 90%). Furthermore we recall that attacks to the state of the art CAPTCHAs take only two seconds [21] (with a probability of some 50%).

Comparing our solution with other image-based CAPTCHAs (presented in Section II-B), our proposal results to be more resilient against attacks. In particular, Table IV reports the comparison considering the common weaknesses of image-based CAPTCHAs, previously discussed in Section II-B. In the table we indicate whether the design is protected against the following attacks: indirect attack, exhaustion of DB, leak of DB, and pure relay attack. In addition, for stream relay attack and machine learning based attacks, we report the cost to perform such attack, in terms of computational time and resources. We notice that most of the designs in the literature limit their focus to a specific threat, but they offer less protection against others. Nevertheless, our proposal is designed to resist to all of them, while maintaining a high usability level.

As a future work, we plan to increase the resiliency of CAPTCHAStar by analyzing the pattern of mouse movements during the resolution of a challenge. We believe this analysis will be meaningful in order to better discriminate human users and automatic programs. Moreover, we intend to study the impact that the size of drawable space has on the usability. Indeed, increasing the size also enlarges the space of research and further reduce the random choice probability. Finally,
we plan to investigate the possibility to leverage additional gaps between human abilities and automatic programs. For example, we intend to involve in a challenge the semantic meaning of the final shape. This means to rely on the innate human ability to relate objects with their semantic. In fact, nowadays this ability is hardly imitable by a machine [31]. We strongly believe that improving CAPTCHAStar challenges in this way will increase further the resiliency of our proposal against machine-learning-based attacks.

**TABLE IV: Protection against the threats in Section II-B**

| CAPTCHA design | Indirect attack | Evasion of the DB | Leak of the DB | Pure relay attack | Stream relay attack | Machine learning | Random chance |
|-----------------|-----------------|-------------------|---------------|------------------|-------------------|------------------|---------------|
| Asira [24]      | X               | X                 | X             | X                | low               | low              | 0.02%         |
| Collage [25]    | X               | X                 | X             | X                | low               | high             | 16.60%        |
| Deep [26]       | X               | X                 | X             | X                | low               | high             | 0.20%         |
| Motion [27]     | X               | X                 | X             | X                | low               | low              | 25.00%        |
| Video [28]      | X               | X                 | X             | X                | low               | high             | 0.30%         |
| Noise [29]      | X               | X                 | X             | X                | mid               | mid              | ~0.00%        |
| Cursor [30]     | X               | X                 | X             | X                | low               | low              | ~0.00%        |
| Jigsaw [31]     | X               | X                 | X             | X                | low               | mid              | ~6.66%        |
| PlayThru [32]   | X               | X                 | X             | X                | high              | high             | ~0.00%        |
| CAPTCHAStar     | X               | X                 | X             | X                | high              | high             | 0.09%         |

**REFERENCES**

[1] D. Lopresti, “Leveraging the captcha problem,” in *Human Interactive Proofs*. Springer, 2005.

[2] L. Von Ahn, M. Blum, and J. Langford, “Telling humans and computers apart automatically,” *Communications of the ACM*, vol. 47, 2004.

[3] E. Bursztein, S. Bethard, C. Fabry, J. C. Mitchell, and D. Jurafsky, “How good are humans at solving captchas? a large scale evaluation,” in *IEEE S&P*, 2010.

[4] N. Ben-Asher, J. Meyer, S. Moller, and R. Englert, “An experimental system for studying the tradeoff between usability and security,” in *IEEE ARES*, 2009.

[5] J. Yan and A. S. El Ahmad, “Usability of captchas or usability issues in captcha design,” in *SOPUS*, 2008.

[6] G. E. Hinton, “To recognize shapes, first learn to generate images,” *Progress in brain research*, vol. 165, 2007.

[7] G. Reynaga and S. Chiasson, “The usability of captchas on smartphones.” in *IEEE CS*, 2013.

[8] M. Shirali-Shahreza and S. Shirali-Shahreza, “Advanced collage captcha,” in *IEEE ITING*, 2008.

[9] M. Shirali-Shahreza and M. Shirali-Shahreza, “Motion captcha,” in *IEEE HCI*, 2008.

[10] S. Shirali-Shahreza and M. Shirali-Shahreza, “Motion captcha,” in *IEEE HCI*, 2008.

[11] K. A. Kluever and R. Zanibbi, “Balancing usability and security in a video captcha,” in *SOPUS*, 2009.

[12] M. Okada and S. Matsuyama, “New captcha for smartphones and tablet pc,” in *IEEE CCE*, 2012.

[13] V. A. Thomas and K. Kaur, “Cursor captcha-implementing captcha using mouse cursor,” in *IEEE WOON*, 2013.

[14] “Are you a robot? introducing "no captcha recaptcha”,” http://areyouahuman.com, Jun. 2013.

[15] B. B. Zhu, J. Yan, Q. Li, C. Yang, J. Liiu, N. Xu, M. Yi, and K. Cai, “Attacks and design of image recognition captchas,” in *IEEE CCS*, 2010.

[16] M. Mohamed, N. Sachdeva, M. Georgescu, S. Gao, N. Saxena, C. Zhang, P. Kumaraguru, P. C. van Oorschot, and W.-B. Chen, “A three-way investigation of a game-captcha: Automated attacks, relay attacks and usability,” in *ACM ASIA CCS*, 2014.

[17] M. Mohamed, S. Gao, N. Saxena, and C. Zhang, “Dynamic cognitive game captcha usability and detection of streaming-based farming,” in *USEC*, Internet Society, 2014.

[18] "Are you a robot? introducing "no captcha recaptcha”,” http://areyouahuman.com, Jun. 2013.

[19] "Are you a robot? introducing "no captcha recaptcha”,” http://areyouahuman.com, Jun. 2013.