CAPTCHaStar! A Novel CAPTCHA Based on Interactive Shape Discovery

Mauro Conti, Claudio Guarisco and Riccardo Spolaor University of Padua, Padua, Italy {conti, rspolaor}@math.unipd.it, cguarisc@gmail.com

Abstract—Over the last years, most websites on which 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 against automated attacks. The battle of wits between designers and attackers of CAPTCHAs led to current ones being annoying and hard to solve 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 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 [23]. 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 [40] (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 [7]. Researchers put a significant effort in understanding the trade-off between usability and resiliency to attacks [5]. Also, in order to measure the effective usability of a CAPTCHA, Yan et al. [43] 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 CAPTCHaStar¹, 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 [17]. 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 present 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 showed 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.

¹A demo is available at http://captchastar.math.unipd.it/demo.php.

Our work suggests that CAPTCHaStar is promising for a practical wide adoption (particularly for mobile devices, where the use of keyboard is more difficult and error-prone [33]), 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. In Section VI, we compare CAPTCHaStar with other image-based CAPTCHAs in the literature, and we discuss limitatons and possible future work. Finally, in Section VII, we draw some conclusions summarizing the contributions of our research.

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 [37] 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*. In the following sections, we divide CAPTCHAs in two main categories, according to the skill required to solve them: textbased (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 back-end that considers the engagement of the user" and prompts the user with either a text-based or an image-based challenge [2]. 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. [3] proposed the first text-based CAPTCHA in 2002. After this first proposal, several other researchers worked on this kind of design. Several researchers focused on improving the resiliency against automated attacks [4], [12], [18]. Currently, text-based CAPTCHAs are the most widely used [8]. In the following, we report two examples of text-based CAPTCHAs that, as for CAPTCHaStar, do not require a keyboard to submit the answer: iCaptcha [39] and DDIM-CAPTCHA [44].

The authors of iCaptcha [39] focused their efforts on the prevention of relay attacks [26]: i.e., when a bot uses an external paid human to solve a CAPTCHA. This text-based CAPTCHA measures and analyzes the interactions a user

performs while solving the challenge. iCaptcha prompts the user with a classical obfuscated word. For each word, there is a sequence of obfuscated letters that the user has to use to compose her answer. iCaptcha verification operates on two fronts. First, the correctness of the answer discriminates a real human from a machine. Second, the interleaving time after the tap of each button discriminates a legitimate user from an external paid human. However, we consider this type of discrimination weak, because the latency of the network connection can heavily affect the measurement of the interleaving times. Moreover, iCaptcha presents the user a small set of candidate characters (i.e., the set of buttons) that composes the solution of the challenge. Unfortunately, while this feature improves usability, it also increases the success rate of attacks that leverage OCR (Optical Character Recognition) software.

The CAPTCHA design proposed by Ye et al. [44] (named DDIM-CAPTCHA) also leverages obfuscated words like traditional text-based schemes. The main difference with the previous design is in the way the user can solve a challenge: instead of asking the user to type the answer, this CAPTCHA asks to drag the correct letters from a pool into an "answer box". Letters in the pool overlap to each other, so the user has to constantly interact with the pool to pick the letter he wants.

a) 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 [13]. 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 [39]). 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 [31].

b) 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, CAPTCHAs 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 [9] 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 [26]. This approach has a really high success rate and it costs only one U.S. dollar per thousand CAPTCHAs [7].

Looking at the literature, the attack strategies against textbased 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 a 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 A02 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 [10], [6].

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 [14]. Indeed, image-based CAPTCHAs usually have a high success rate and they are less challenging than text-based ones [27]. 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 was Asirra [11], which was discontinued in fall 2014. 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 [35]: it requests to click on a specific picture, among six pictures randomly taken. Deep CAPTCHA [27] 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 [36] 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 [20] 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 [29] 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 [38] 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 [14] reprises the classical jigsaw puzzle. Indeed, the user needs to correctly rearrange the pieces of a jigsaw. Finally, PlayThru [1] asks the user to solve a randomly generated mini-game. These mini-games require to drag objects on their correct spots.

- c) 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 [33]. 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 [45] requires more than two minutes to generate a single challenge).
- d) 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. Henceforth, an attacker might run what we call *indirect attack*: get the solution from the client-side (e.g., via reverse engineering of the client application).
- 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 Turk²).
- 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.

²https://www.mturk.com/.

- 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 a human solver, and then relay back the input of the user to the server. This strategy is defined as *stream relay attack* [25].

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 unfeasible 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 [24] (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 her cursor, until she 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 she 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 she is confident that the resulting shape is detailed enough (see Figure 1d). Finally, the user confirms the current cursor position as her final answer. The system compares the solution with the final answer (allowing a small margin 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, and performs the check only when the user confirms her answer, that is considered as final and irrevocable. For the sake of usability, CAPTCHaStar considers as a valid answer also a pair of coordinates that is close enough to the actual solution (more details in Section III-B).

The generation phase of a challenge involves some parameters to tune usability and security:

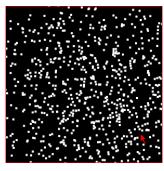
- *Noise* (ψ) : the percentage of noisy stars added to the scheme, with respect to the number of original stars.
- Sensitivity (δ) : the relationship between the amount of displacement of the cursor (in pixel) and the movement of each star (more details in Section III-B).
- *NSol*: the number of possible solutions (i.e., secret positions) of the challenge. Each solution corresponds to a different shape.
- *PicSize*: the maximum value between width and height on the sampled picture, expressed in number of pixels.
- *Rotation*: Boolean value that indicates whether the picture is rotated by a random degree.

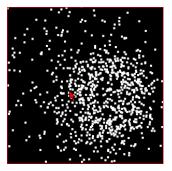
B. Prototype implementation

To assess the feasibility and effectiveness of our solution, we did a complete implementation. In particular, we aimed at providing an implementation that could be widely deployed. Since *PHP* is the most supported programming language by web servers [41], we implemented the server-side part of our design using this language. We implemented the client-side part using *HTML5 Canvas*, because it has the support for majority of commercial browsers [15]. We manually retrieved more than 5000 pictures with two colors, by searching free-to-use collections³ of vector icons in .png format (this step could be automated, e.g., with web crawlers). We collected

³Free vector icons, http://www.flaticon.com/.









(a) A random starting picture.

(b) A sample unsolved challenge. (c) An almost solved challenge. (d) A correctly solved challenge.

Fig. 1: The process of solving a CAPTCHaStar challenge.

all these pictures in a pool. For a real life deployment of that system, we recommend using a pool as large as possible. In the following, we first describe how a challenge is generated on the server-side, then we describe how it is presented to the user on the client-side.

1) Generation of a challenge: Each challenge is composed by original stars (generated from the base shape) and noisy stars (generated randomly). The steps to generate a challenge are as follows: i) Picture selection and pre-process; ii) Picture decomposition; iii) Trajectory computation. Our system repeats these steps for a number of times equal to the value of the parameter NSol.

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: (i) 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; (ii) 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_x^i, P_y^i) as the coordinates of the position that the star i takes when the cursor is in coordinates (sol_x, sol_y) . For each star i, our system randomly generates four coefficients $(m_{x,x}^i, m_{x,y}^i, m_{y,x}^i, m_{y,y}^i)$, that relates the coordinates of the star with the coordinates of the cursor: m_{ab}^i 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 $\left[-\frac{\delta}{10}, \frac{\delta}{10}\right]$ (we remind that δ is the *sensitivity* value). Our system computes a pair of constants, (C_x^i, C_y^i) , for each original star i as follows:

$$\begin{split} C_x^i &= P_x^i - sol_y \cdot m_{x,y}^i - sol_x \cdot m_{x,x}^i, \\ C_y^i &= P_y^i - sol_y \cdot m_{y,y}^i - sol_x \cdot m_{y,x}^i. \end{split}$$

CAPTCHaStar generates the noisy stars in a similar way, but their coordinates (P_x, P_y) having random values. The number of noisy stars is equal to the percentage ψ of the number of original stars. Henceforth, we define as trajectories parameters of star i, the following set of parameters: $m_{x,x}^i$ $m_{x,y}^i, C_x^i, m_{y,x}^i, m_{y,y}^i, C_y^i$. The only information that the client needs from the server in order to calculate the position of the stars, whenever the user moves her 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:

$$\begin{aligned} x^i &= m_{x,y}^i \cdot cur_y + m_{x,x}^i \cdot cur_x + C_x^i, \\ y^i &= m_{y,x}^i \cdot cur_x + m_{y,y}^i \cdot cur_y + C_y^i. \end{aligned}$$

When the user confirms her 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.

Our server-side script calculates Δ 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 Δ 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 (more details in Section IV-A). 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 [28] of the stars' behavior, hence moving the cursor toward the position that is closer to a real shape.

C. System performance

In this section, we discuss the performance of our proposal in terms of computational time and bandwidth. The performance is directly related to the number of stars in a challenge. In Figure 2, we report the probability density of the number of stars in a challenge, computed from our database with 4033 challenges. This probability density results in a Normal Density Curve with mean $\mu = 542.7$ and standard deviation $\sigma = 314.4$. According with this finding, for a CAPTCHaStar challenge with an average number of stars (i.e., 543 stars), the server sends to the client 12.7 kB (i.e., 4 bytes for each one of the six parameters). In Figure 3, we report the distribution of network overheads caused by the request of a CAPTCHaStar challenge. We can observe that more of the 75% of the challenges are smaller than 17 kB. We underline that in out current prototype implementation, the parameters are passed without any compression, while a compression could reduce the amount of Bytes transmitted. As a comparison with the existing CAPTCHAs, we point out that a text-based reCaptcha challenge sends some 3 kB, while an image-based reCaptcha challenge sends some 35 kB. As an additional example, a single Asirra challenge needs more than 360 kB.

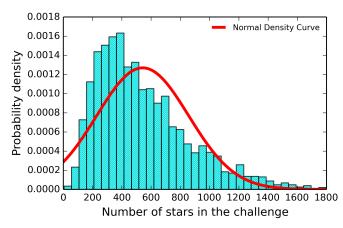


Fig. 2: Probability distribution of the number of stars.

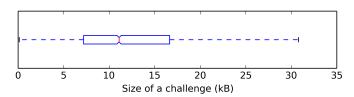


Fig. 3: Statistical distribution of the sizes of the challenges in kilobytes. The notch of the box represents the median value. First and third quartile are represented as the left and right side of the notched box. Lines that extend horizontally from the boxes indicate the 2^{nd} percentile (left) and the 98^{th} percentile (right).

The server we use to generate the challenges is a PC with a 3.0 GHz AMD Athlon Dual Core Processor and 1 GB of RAM. In Figure 4, we report the average time required by each phase of our solution to generate a challenge described above, depending on the number of stars. As we can see from Figure 4, the most costly phase of the process is the picture decomposition phase, i.e., the generation of the original stars by sampling the starting image. This phase takes some 76% of the overall time to generate a challenge. The picture selection and pre-process phase (i.e., loading, rotating and resizing the image) requires around 21% of the overall time. Only some 3\% of the overall challenge generation time is due to the trajectory computation phase. Considering the average challenge (i.e., 542 stars), the overall time required to generate such challenge is about 0.75 seconds. We underline that even in the worst case scenario, the overall time is always lower than two seconds. This suggests that our prototype can handle efficiently a wide number of requests, even with low cost hardware resources. Moreover, we believe the generation time would in no way be a showstopper for CAPTCHAs (and for our proposal in particular), since a possible solution (indeed applicable to several CAPTCHAs) would be either to generate the challenge while the user is doing other operations, or to maintain a pool of pre-generated challenges, and randomly pick one when needed.

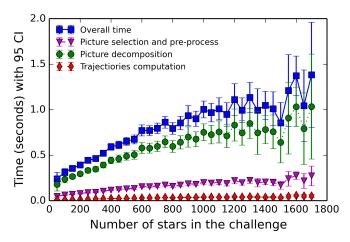


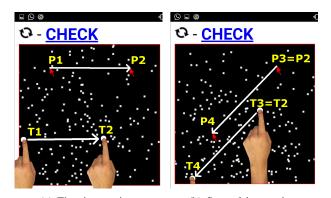
Fig. 4: Challenge generation time.

D. User interaction

For the sake of usability of CAPTCHaStar, we consider different cursors according to the device it runs on. In particular, considering a browser on a personal computer, the cursor coincides with the default mouse pointer, and the user can submit her final answer by clicking the left mouse button. On the other hand, on smartphones or other touch-enabled devices, the cursor position is usually not represented with a graphical object, such as the mouse pointer on personal computers. For this reason, we represent the cursor position as a red arrow inside the drawable area (as shown in Figure 5). The user can move the cursor by swiping her finger on the drawable area (note that only the starting point of the swipe must be inside

the drawable area). In Figure 5a, we report a first example of a user moving the cursor from position P1 to P2, by swiping the finger from T1 to T2 on the touchscreen. In Figure 5b, we show a second example, where the starting position P3 of the cursor corresponds to the ending position P2 in Figure 5a. In the second example, a user moves the cursor from position P3 to P4, with a swipe from T3 to T4. We highlight that the path of the cursor is mapped directly to the path of the swipe, regardless of the starting point of that swipe. Moreover, the position of the cursor at the end of the swipe in Figure 5a (i.e., T2) remains in position P2.

In order to submit her final answer on a mobile browser, the user has to tap on a "CHECK" link placed outside of the drawable area, as shown at the top of figures 5a and 5b.



(a) First interaction. (b) Second interaction. Fig. 5: User interaction examples on touchscreen devices.

IV. USER STUDY

In order to evaluate our proposal, we ran a user study according to the usability metrics proposed in [43], and an exhaustive set of parameter combinations. In particular, we compare our solution with text-based CAPTCHAs taken from reCaptcha [32]. In the following, we describe in detail how we ran the user study and discuss the obtained results.

A. Parameters selection

In order to notify the user whether she passes or fails a challenge, we need to set the value of the *usability tolerance* parameter (introduced in Section III-B). On one hand, increasing the value of this parameter makes CAPTCHaStar more permissive. On the other hand, it also increases the success rate of random guess attacks with a quadratic growth. For this reason, it was crucial to identify an optimal value for the *usability tolerance* parameter, in order to obtain a good tradeoff between usability and security.

We ran this preliminary study on a set of 35 participants (volunteers and without any reward) under our supervision. A session of this study consisted in solving three CAPTCHaStar tests (named P1, P2 and P3) for at least two times (i.e., six challenges in total). CAPTCHaStar tests are randomly generated using the parameter values reported in Table I.

Figure 6 reports the success rate of the participants and success rate of random guess attacks, as the *usability tolerance*

Test	P1	P1 P2	
ψ	0%	70%	70%
δ	5	7	7
NSol	1	1	1
Rotation	Off	Off	On

TABLE I: Values of parameters ψ , δ , *NSol* and *Rotation* for the preliminary study.

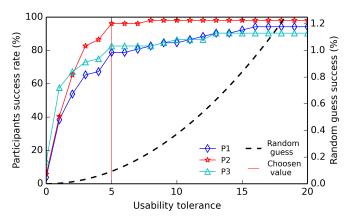


Fig. 6: Success rate of the participants of the preliminary study (scale on the left-hand side of the graph) and success rate of random guess attack (scale on the right-hand side of the graph), varying the value of *usability tolerance*.

Test	T1	T2	T3	T4	T5	
$oldsymbol{\psi}$	0%	70%	70%	10%	0%	250%
δ	5	7	7	7	10	5
NSol	1	1	1	2	3	1
Rotation	Off	Off	On	Off	Off	Off
Usability tolerance	5	5	5	5	5	5

TABLE II: Values of parameters ψ , δ , NSol, Rotation and Usability tolerance for the survey.

varies. In particular, we noticed that the participants success rate grows rapidly until *usability tolerance* is equal to five, then it plateaus. Since the random guess attack success rate grows very fast (i.e., has a quadratic growth), and the user success rate does not improve significantly with a *usability tolerance* greater than five, for our user study (described in the following) we set the *usability tolerance* equal to five.

B. Survey design and implementation

We designed a web-based survey page, in order to collect data from a large number of participants. We built a survey composed of eight different tests: six CAPTCHaStar challenges (named from T1 to T6) and two text-based ones (T7 and T8). Tests from T1 to T6 are randomly generated (i.e., starting from a random image) using the value of parameters reported in Table II. Tests T4 and T5 have more than one solutions, i.e., two and three, respectively. Test T4 requires the user to find both of its solutions, while for T5, it is enough to find only one of the three existing solutions. In Figure 7, we present some examples of solved challenges for the settings from T1 to T6.

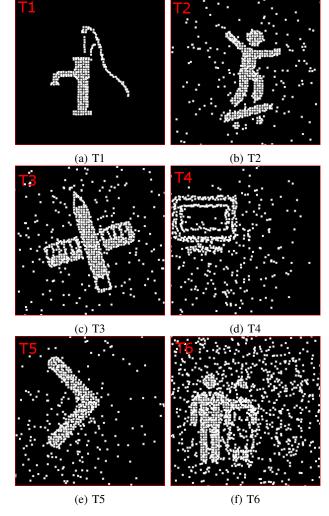


Fig. 7: Screenshots of examples of solved challenges, generated randomly using the settings from T1 to T6.

The last two tests are random text-based CAPTCHAs from reCaptcha, with one and two words (i.e., T7 and T8, respectively). In order to minimize the learning effect [21], we prompt the user with the eight tests selected in a random order. At the beginning of the survey, we prompt users with a description of our proposal and a simple demo. Then, we ask the participants to fill out a form with their demographic information: age, gender, nationality, level of education, years passed using Internet, and frequency of Internet use. We gather this data in order to understand whether factors like the experience of the user affects the performances in solving CAPTCHaStar challenges. In the same page, we also ask the participants to read and accept an informed consent statement, where we declare how we intend to use the collected data and that we do not intend to disclose private information to third parties. For each test in the survey, we ask the user to rate the perceived difficulty of that test on a scale from 1 to 5. 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 design this survey in a way that each session should last less than 10 minutes.

C. Participants

All the participants took the survey unsupervised using their own devices, in order to recreate the natural conditions of use of CAPTCHaStar. 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), in order to collect usage data for a large number of participants. We did not give any reward for the participation. More than 250 users took part in our survey (258 users, 81% male and 19% female). We made sure that none of the 35 participants of the preliminary study took part to this user study. The average age of the participants was 25.5. The education level was 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 the participants used Internet daily, 49% from 5 to 10 years, 33% for more than 10 to 15 years, 28% for more than 15 years. The majority (some 90%) of the participants were Italians. However, we did not notice significant differences in the performance of users of different nationalities. Finally, only a few participants used mobile devices (16 users), with performances similar to desktop users.

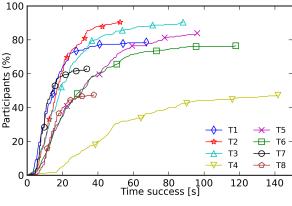
D. Results and discussion

Among all the participants, only 35% of them preferred traditional CAPTCHAs rather than CAPTCHaStar. Table III reports the success rate and the average solving time for each of the eight challenges described above.

		CAPTCHaStar						Text	
	Test	T1	T2	T3	T4	T5	T6	T7	T8
Succ.	Rate (%)	78.7	90.2	90.6	50.4	85.1	76.6	62.7	46.9
Dif	ficulty	1.9	2.4	2.6	3.4	2.9	3.1	2.4	2.7
Succ.	Avg (s)	14.4	17.5	22.2	54.1	30.2	28.5	11.0	14.9
Succ.	Std	9.8	9.3	15.8	33.5	20.2	19.7	5.4	6.1
Fail	Avg (s)	14.7	18.2	33.1	49.0	38.8	40.0	12.6	21.2
ran	Std	13.5	10.7	21.2	33.5	26.6	25.6	8.8	17.8

TABLE III: Survey results for CAPTCHaStar and text-based (Text) CAPTCHAs.

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): a possible reason for this could be users having different abilities in solving CAPTCHaStar challenges. 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). We believe that users found T4 more difficult to be solved because it requires to discover two images (i.e., original stars for two images, plus the *noisy stars*). In particular, T2 shows a success rate that is some 90%, which is higher than the 84% for text-based CAPTCHAs reported



(a) Success rates over time.

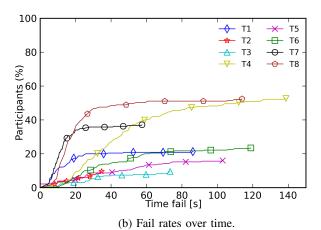


Fig. 8: Success and fail rate over time.

in [7]. We underline that in our text-based CAPTCHAS T7 and T8 (where we used current reCaptcha used by Google), we observed a success rate of 62.7% (for the simpler test with only one word). In figures 8a and 8b, we report in the domain of time the percentages of the participants that solved and failed a challenge, respectively.

We highlight that text-based challenges (T7 and T8) rapidly approach to their maximum within some 20 seconds, while CAPTCHaStar challenges reach a higher success rate in just a few more seconds. Indeed, the average time to solve T2 is some 17 seconds, which is some 6 seconds higher than the best time for text-based CAPTCHAs (i.e., 11 seconds for T7). We believe that this is an acceptable value.

In order to validate the choice of the *usability tolerance* value, we report in Figure 9 a chart that now takes into account the data obtained from the main user study. This chart confirms our choice of five as being a reasonable value of *usability tolerance*. In fact, increasing the *usability tolerance* (which also means giving more chances to adversaries) to a value above 5, brings only a limited benefit in the participants success rate – see how the slope of most curves varies at *usability tolerance* equal to 5.

Finally, we asked users to rate the "ease of understanding" of the system on a scale from 1 (very simple) to 10 (very

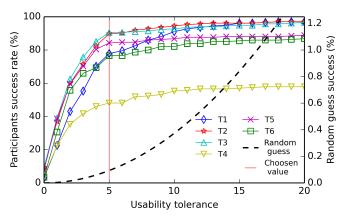


Fig. 9: Success rate of the participants of the main user study (scale on the left-hand side of the graph) and success rate of random guess attack (scale on the right-hand side of the graph), varying the value of *usability tolerance*.

difficult), and the results show an average value of 4.53, with standard deviation of 2.53. In general, the comments received from the participants were positive. In particular, more than one gave comments similar to the following: "I think CAPTCHaStar is better than text-based CAPTCHAs because it's a kind of game". Other participants said something like "text-based CAPTCHAS may require less time than yours, but I prefer CAPTCHaStar because I actually enjoyed it".

E. Learning effect evaluation

Furthermore, we underline that in our user study, users were never trained before to solve our CAPTCHA, while trained users might need a smaller amount of time to solve CAPTCHaStar. To verify this claim, we analyzed the performance of 25 users that repeated the whole survey (i.e., tests from T1 to T8) at least three times. The results of this analysis are reported in Figure 10.

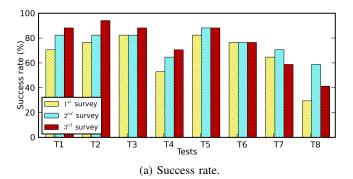
From Figure 10, we can observe that these users improved their performance to solve CAPTCHaStar challenges as they repeated the survey (i.e., increasing the success rate and decreasing of the completion time), while their performance on text-based challenges (T7 and T8) remained quite the same. These 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 adhoc 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).



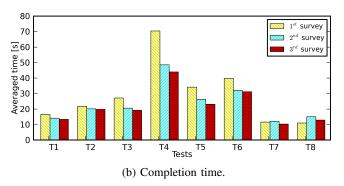


Fig. 10: Learning effect on repeating the survey.

- 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 the challenge randomly 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 alters the original picture, as described in Section III-A.
- 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 V-C.

- Random Choice (A12): For the sake of usability, CAPTCHaStar also accepts as a correct answer the neighborhood of the solution (according to the value of usability tolerance parameter). Nevertheless, the probability of success of a 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 a relay attack into practice.
- 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 a 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 minimizes 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). In order to achieve the highest possible success rate, we chose to consider the whole research space, instead of a sub-sample. The computational cost of the attack can be really high, depending on the implemented heuristic. We implemented the automatic solver and the heuristics described below using the 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 (we chose these parameters since test T2 was the test with the highest success rate). In this evaluation, we used a PC with a 2.3 GHz Intel Pentium B970 and 4 GB memory.

1) Minimize height/width of stars (MinSize): We define with 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:

$$MinSize(k) = \left(\max_{s \in S^k} x_s - \min_{s \in S^k} x_s\right) + \left(\max_{s \in S^k} y_s - \min_{s \in S^k} y_s\right).$$

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 of dividing the drawable space into 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:

$$f_{score}(t) = |2 \cdot \sum_{i=1}^{25} \sum_{j=1}^{25} t_{ij} - 25^{2}|.$$

The heuristic is defined as:

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

The value of the *sensitivity* parameter (δ) 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 δ 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 to detect 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:

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

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:

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

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. Using sampling pairs on this algorithm would reduce the search space, and thus the execution time. However, we underline that this attack will remain useless, since its performance is less than 2% of success. We expect that the use of sampling pairs would further reduce this success rate.

In Figure 11a and Figure 11b, we report how the success rates of the heuristics described above vary at the change of parameters δ and ψ . From Figure 11a, we observe that for $\psi=70\%$, the success rate is always smaller than 3%. From Figure 11b, 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%).

From Table IV, 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%.

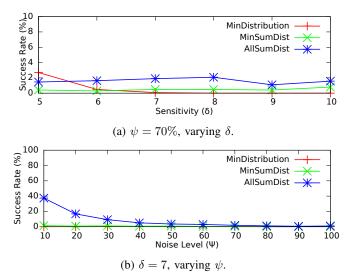


Fig. 11: Comparison of success rates on variations of δ and ψ .

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

TABLE IV: 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. We implemented this tool using scikit-learn libraries [30]. In the following, we report in details how we built such 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; and in Section V-C3, we describe the actual attack and we evaluate its performance.

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 [22] 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 a new unlabeled example belongs. 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 [16], 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_{ω}^{k} of squared tiles. The parameter ω is the amount of pixels in a tile side. For each considered value of ω , we build a vector $F_{\omega} = \langle f_1, ..., f_n \rangle$ 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_{ω} must be different from each other. For this reason, we apply k-mean clustering method on a training set of candidate tiles with side ω , using K=n and h as similarity metric. At the end of the clustering procedure, we obtain a vector F_{ω} , 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:

$$d_i^k = |\{t \in T_\omega^k : f_i = \underset{l \in F_\omega}{\operatorname{argmin}} \ h(l,t)\}|, \quad \forall i = 1,..,n.$$

The values in the features vectors D_{ω}^{k} are normalized, from 0 to 1. In practical terms, for a fixed ω , this procedure produces a vector of features D_{ω}^{z} , starting from a cursor position z that corresponds to the state S^{z} .

2) Classifiers training: We trained 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 a RF classifier with 60 Decision trees estimators, and we use an SVM classifier with Radial Basis Function (RBF) as the 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 ω .

We underline that an attacker has 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, which allows us to provide the neighbors of the solution to the classifier.

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 as the value

of parameter ω changes. 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 as 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_{λ} cursor positions. For each classifier with a specific ω , we define C_{ω} 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_{ω} often presents multiple local maximums and large plateaus. For this reason, an attacker must find the cursor position k_{sol} that corresponds to a global maximum for the function C_{ω} :

$$k_{sol} = \underset{k \in K_{\lambda}}{\operatorname{argmax}} C_{\omega}(S^k).$$

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 ω . We executed the attacks on a high end PC with a 3.16 GHz Intel Xeon X5460 and 32 GB of RAM. Figure 12a and Figure 12b 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 12a).

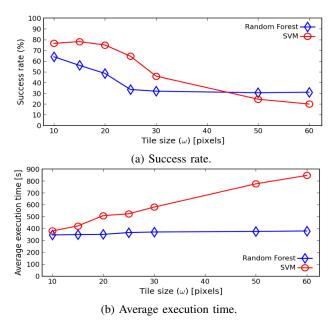


Fig. 12: Success rate and average execution time for machine learning-based attacks, varying the tile size (ω) .

The time required to build the features vectors D_{ω}^k , $\forall k \in K_{\lambda}$, remains stable at around 340 seconds. On one hand, the time required to compute the probability values $C_{\omega}(S^k)$, $\forall k \in K_{\lambda}$

 K_{λ} , increases linearly using SVM classifier, according to the value of ω . 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 a success rate of more than 90% 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 [6], machine learning based attacks achieve some 50% success rate in only two seconds against Baidu and eBay CAPTCHAs.

VI. DISCUSSION

In this section, we first compare our solution with other image-based CAPTCHAs (Section VI-A), then we discuss limitations of CAPTCHaStar (Section VI-B) and finally we discuss some future work (Section VI-C).

A. Comparison with other image-based CAPTCHAS

Comparing our solution with the state-of-the-art of image-based CAPTCHAs (presented in Section II-B), our proposal turns out to be more resilient against attacks. In particular, Table V 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. On the other hand, our proposal is designed to resist all of them, while maintaining a high usability level.

CAPTCHA design	Indirect attack	Exhaustion of the DB	Leak of the DB	Pure relay attack	Stream relay attack	Machine learning	Random
Asirra [11]	/	/	X	Х	low	low	0.02%
Collage [35]	/	X	X	X	low	high	16.60%
Deep [27]	/	1	/	X	low	high	0.20%
Motion [36]	/	X	X	1	low	high	25.00%
Video [20]	1	1	X	X	low	high	0.30%
Noise [29]	1	1	1	1	mid	mid	$\sim 0.00\%$
Cursor [38]	/	1	/	1	low	low	$\sim 0.00\%$
Jigsaw [14]	1	X	X	X	low	mid	6.66%
PlayThru [1]	X	X	X	/	high	high	$\sim 0.00\%$
CAPTCHaStar	/	/	/	/	high	high	0.09%

TABLE V: Protection against the threats in Section II-B.

B. Limitations

Unfortunately, we (as well as other CAPTCHA proposers) are not able to prove that our CAPTCHA is secure against all the possible attacks. We believe the best a researcher can do in such cases is to consider both current traditional attacks

(see our Section V-A) and ad-hoc ones (see our sections V-B and V-C). As an example, Asirra [11] has been later proven to be breakable [16], [45]. The same goes for other CAPTCHAs, such as ReCaptcha (both versions of 2011 and 2013) and the ones employed by CNN, Wikipedia, Yahoo, Microsoft [42] and PlayThru [1].

In this paper, we evaluated the attacks that we were able to think up with our best efforts. Such evaluation suggests that our proposal is still more resilient than some state-of-the-art CAPTCHAs. Indeed, BAIDU and eBay CAPTCHAs can be broken with a success rate of 50% in just two seconds [7]).

We demonstrated that the problem of classifying an arrangement of dots as forming a shape or not is learnable by a SVM classifier. However, as shown by the attack evaluation reported in Section V-C: the bottleneck for an attack, that uses such shape recognition as a building block, is the generation of the sampled search space (i.e., the possible configurations by varying the mouse coordinates). Finally, such complexity can be further increased by enlarging the drawable space (also reducing the success rate of random chance attack).

C. Future work

As a future work, we plan to further 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 perform some qualitative experiments in the laboratory, especially to evaluate the usability on mobile devices. However, we highlight that CAPTCHaStar already takes into account the recommendations to improve CAPTCHA usability on smartphones, recently proposed by Chiasson et al. in [34]. In fact, our proposal has the following properties, which are desirable features for CAPTCHAs on a smartphone:

- Our design focuses only on a single task, avoiding optional features.
- The input mechanism of CAPTCHaStar on touch-enabled devices (described in Section III-D) is cross-platform, and it does not interfere with normal operations of the browser.
- It is possible to isolate CAPTCHaStar from the rest of the web form.
- CAPTCHaStar minimizes bandwidth usage. Indeed, the data transmission with the server is limited to the exchange of trajectories parameters of the stars, and the verification of the user final answer.
- Our design does not require any additional feature nor library to be installed on the browser.

Finally, we plan to investigate the possibility of leveraging 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 [45]. We strongly believe that improving CAPTCHaStar

challenges in this way will further increase the resiliency of our proposal against machine learning-based attacks.

VII. CONCLUSIONS

In this paper, we proposed CAPTCHaStar, a novel imagebased CAPTCHA that leverages the innate human ability to recognize shapes in a confused environment [19]. Our study demonstrates that our proposal meets both security and usability requirements for a good CAPTCHA design.

We described in detail our prototype implementation and the selection of parameters involved in challenge generation. Data collected from our user study confirmed the usability of our proposal. Indeed, users were able to obtain a success rate higher than 90%, which is better than the success rates of CAPTCHAs currently used in websites [7] 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 CAPTCHa-Star against traditional and automated ad-hoc attacks. Indeed, these attacks were shown to be ineffective (for a proper setting of our parameters). We also performed an attack leveraging a machine learning classifier, which 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 [6] (with a probability of some 50%).

REFERENCES

- AreYouAHuman game based CAPTCHAs. http://areyouahuman.com, June 2013.
- [2] Introducing "NoCAPTCHA reCAPTCHA". googleonlinesecurity. blogspot.co.uk/2014/12/are-you-robot-introducing-no-captcha.html, Dec. 2014.
- [3] H. S. Baird, A. L. Coates, and R. J. Fateman. Pessimalprint: a reverse turing test. *International Journal on Document Analysis and Recognition*, 5(2-3):158–163, 2003.
- [4] H. S. Baird and T. P. Riopka. ScatterType: a reading CAPTCHA resistant to segmentation attack. In *Proceedings of the Conference on Electronic Imaging*, pages 197–207. International Society for Optics and Photonics, 2005
- [5] N. Ben-Asher, J. Meyer, S. Moller, and R. Englert. An experimental system for studying the tradeoff between usability and security. In Proceedings of the International Conference on Availability, Reliability and Security (ARES), pages 882–887. IEEE, 2009.
- [6] E. Bursztein, J. Aigrain, A. Moscicki, and J. C. Mitchell. The end is nigh: Generic solving of text-based captchas. In *Proceedings of the 8th* Workshop on Offensive Technologies (WOOT 14). USENIX, 2014.
- [7] E. Bursztein, S. Bethard, C. Fabry, J. C. Mitchell, and D. Jurafsky. How good are humans at solving CAPTCHAs? a large scale evaluation. In *Proceedings of the Symposium on Security and Privacy (SP)*, pages 399–413. IEEE, 2010.
- [8] E. Bursztein, M. Martin, and J. Mitchell. Text-based CAPTCHA strengths and weaknesses. In *Proceedings of the 18th Conference on Computer and communications security*, pages 125–138. ACM, 2011.

- [9] K. Chellapilla, K. Larson, P. Y. Simard, and M. Czerwinski. Computers beat Humans at Single Character Recognition in Reading based Human Interaction Proofs (HIPs). In *Proceedings of the Conference on Email* and Anti-Spam, 2005.
- [10] A. S. El Ahmad, J. Yan, and L. Marshall. The robustness of a new captcha. In *Proceedings of the Third European Workshop on System* Security, pages 36–41. ACM, 2010.
- [11] J. Elson, J. R. Douceur, J. Howell, and J. Saul. Asirra: a captcha that exploits interest-aligned manual image categorization. In *Proceedings* of the Conference on Computer and Communications Security, pages 366–374. ACM, 2007.
- [12] R. Ferzli, R. Bazzi, and L. J. Karam. A captcha based on the human visual systems masking characteristics. In *Proceedings of the International Conference on Multimedia and Expo*, pages 517–520. IEEE, 2006.
- [13] C. A. Fidas, A. G. Voyiatzis, and N. M. Avouris. On the necessity of user-friendly CAPTCHA. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 2623–2626. ACM, 2011.
- [14] H. Gao, D. Yao, H. Liu, X. Liu, and L. Wang. A novel image based CAPTCHA using jigsaw puzzle. In Proceedings of the 13th International Conference on Computational Science and Engineering (CSE), pages 351–356. IEEE, 2010.
- [15] S. GlobalStats. Canvas (basic support). http://caniuse.com/#feat=canvas, Sept. 2014.
- [16] P. Golle. Machine learning attacks against the asirra CAPTCHA. In Proceedings of the 15th Conference on Computer and communications security, pages 535–542. ACM, 2008.
- [17] G. E. Hinton. To recognize shapes, first learn to generate images. Progress in brain research, 165:535–547, 2007.
- [18] I. F. Ince, I. Yengin, Y. B. Salman, H.-G. Cho, and T.-C. Yang. Designing captcha algorithm: splitting and rotating the images against ocrs. In Proceedings of the Third International Conference on Convergence and Hybrid Information Technology (ICCIT), volume 2, pages 596–601. IEEE, 2008.
- [19] G. Kanizsa and G. Kanizsa. Organization in vision: Essays on Gestalt perception. Praeger New York, 1979.
- [20] K. A. Kluever and R. Zanibbi. Balancing usability and security in a video CAPTCHA. In *Proceedings of the 5th Symposium on Usable Privacy and Security*, page 14. ACM, 2009.
- [21] R. Kosara, C. G. Healey, V. Interrante, D. H. Laidlaw, and C. Ware. User studies: Why, how, and when? *Computer Graphics and Applications*, 23(4):20–25, 2003.
- [22] S. B. Kotsiantis, I. Zaharakis, and P. Pintelas. Supervised machine learning: A review of classification techniques. IOS Press, 2007.
- [23] D. Lopresti. Leveraging the CAPTCHA problem. In *Human Interactive Proofs*, pages 97–110. Springer, 2005.
- [24] M. Mohamed, S. Gao, N. Saxena, and C. Zhang. Dynamic cognitive game captcha usability and detection of streaming-based farming. In Proceedings of the NDSS Workshop on Usable Security (USEC). Internet Society, 2014.
- [25] 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 *Proceedings of the 9th Symposium on Information, computer* and communications security, pages 195–206. ACM, 2014.
- [26] M. Motoyama, K. Levchenko, C. Kanich, D. McCoy, G. M. Voelker, and S. Savage. Re: Captchas-understanding captcha-solving services in an economic context. In *Proceedings of the USENIX Security Symposium*, volume 10, page 3, 2010.
- [27] H. Nejati, N.-M. Cheung, R. Sosa, and D. C. Koh. DeepCAPTCHA: an image CAPTCHA based on depth perception. In *Proceedings of the 5th Multimedia Systems Conference*, pages 81–90. ACM, 2014.
- [28] D. A. Norman. The design of everyday things: Revised and expanded edition. Basic books, 2013.
- [29] M. Okada and S. Matsuyama. New captcha for smartphones and tablet pc. In *Proceedings of the Consumer Communications and Networking Conference (CCNC)*, pages 34–35. IEEE, 2012.
- [30] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python, http://scikit-learn.org/. *Journal of Machine Learning Research*, 12:2825–2830, 2011.
- [31] S. Poslad. Ubiquitous computing: smart devices, environments and interactions. John Wiley & Sons, 2011.

- [32] G. reCaptcha. recaptcha plugins (php). https://developers.google.com/recaptcha/docs/php, Nov. 2014.
- [33] G. Reynaga and S. Chiasson. The usability of CAPTCHAs on smartphones. In *Proceedings of the International Conference on Security and Cryptography (SECRYPT)*, pages 427–434, 2013.
- [34] G. Reynaga, S. Chiasson, and P. van Oorschot. Exploring the usability of captchas on smartphones: Comparisons and recommendations. In *Proceedings of the NDSS Workshop on Usable Security (USEC)*. Internet Society, 2015.
- [35] M. Shirali-Shahreza and S. Shirali-Shahreza. Advanced collage captcha. In Proceedings of the Fifth International Conference on Information Technology: New Generations (ITNG), pages 1234–1235. IEEE, 2008.
- [36] M. Shirali-Shahreza and S. Shirali-Shahreza. Motion captcha. In Proceedings of the Conference on Human System Interactions, pages 1042–1044. IEEE, 2008.
- [37] M. H. Shirali-Shahreza and S. Shirali-Shahreza. *Distinguishing Human Users from Bots*. IGI Global, 2014.
- [38] V. Thomas and K. Kaur. Cursor CAPTCHAImplementing CAPTCHA using mouse cursor. In Proceedings of the Conference on Wireless and Optical Communications Networks (WOCN), pages 1–5. IEEE, 2013.
- [39] H. D. Truong, C. F. Turner, and C. C. Zou. iCAPTCHA: the next generation of CAPTCHA designed to defend against 3rd party human

- attacks. In Proceedings of the International Conference on Communications (ICC), pages 1–6. IEEE, 2011.
- [40] L. Von Ahn, M. Blum, and J. Langford. Telling humans and computers apart automatically. *Communications of the ACM*, 47(2):56–60, 2004.
- [41] w3techs. Usage of server-side programming languages for websites. http://w3techs.com/technologies/overview/programming_language/all/, Oct. 2014.
- [42] J. Yan and A. S. El Ahmad. A low-cost attack on a microsoft CAPTCHA. In Proceedings of the 15th Conference on Computer and communications security, pages 543–554. ACM, 2008.
- [43] J. Yan and A. S. El Ahmad. Usability of CAPTCHAs or usability issues in CAPTCHA design. In *Proceedings of the 4th symposium on Usable* privacy and security, pages 44–52. ACM, 2008.
- [44] Q.-B. Ye, T.-E. Wei, A. B. Jeng, H.-M. Lee, and K.-P. Wu. DDIM-CAPTCHA: A novel drag-n-drop interactive masking CAPTCHA against the third party human attacks. In *Proceedings of the Conference on Technologies and Applications of Artificial Intelligence (TAAI)*, pages 158–163. IEEE, 2013.
- [45] B. B. Zhu, J. Yan, Q. Li, C. Yang, J. Liu, N. Xu, M. Yi, and K. Cai. Attacks and design of image recognition CAPTCHAs. In *Proceedings of the 17th Conference on Computer and communications security*, pages 187–200. ACM, 2010.