A Heuristic Approach on Carp using Complete Ai Problems

K. Keerthi
M.Tech (CSE)
Annamacharya Institute of Technology (AITK),
Kadapa

***Abstract*— Many security primitives are based on complete mathematical problems. Using hard/complete AI problems for security is emerging as an exciting new paradigm, but has been under-explored. In this paper, it present a new security primitive based on hard AI problems, namely, a novel family of graphical password systems built on top of Captcha technology, which they call Captcha as graphical passwords (CaRP). CaRP is both a Captcha and a graphical password scheme. CaRP addresses a number of security problems altogether, such as online guessing attacks, relay attacks, and, if combined with dual-view technologies, shoulder-surfing attacks. Notably, a CaRP password can be found only probabilistically by automatic online guessing attacks even if the password is in the search set. CaRP also offers a novel approach to address the well-known image hotspot problem in popular graphical password systems, such as Pass Points, that often leads to weak password choices. CaRP is not a panacea, but it offers reasonable security and usability and appears to fit well with some practical applications for improving online security.**

Keywords*— Graphical password, password, hotspots, CaRP, Captcha, dictionary attack, password guessing attack, security primitive.

I. INTRODUCTION

A FUNDAMENTAL task in security is to create cryptographic primitives based on hard mathematical problems that are computationally intractable. For example, the problem of integer factorization is fundamental to the RSA public-key cryptosystem and the Rabin encryption. The discrete logarithm problem is fundamental to the ElGamal encryption, the Diffie- Hellman key exchange, the Digital Signature Algorithm, the elliptic curve cryptography and so on.

Using hard AI (Artificial Intelligence) problems for security, initially proposed in, is an exciting new paradigm. Under this paradigm, the most notable primitive invented is Captcha, which distinguishes human users from computers by presenting a challenge, i.e., a puzzle, beyond the capability of computers but easy for humans. Captcha is now a standard Internet security technique to protect online email and other Services from being abused by bots. However, this new paradigm has achieved just a limited success as compared with the cryptographic primitives based on hard math problems and their wide applications. Is it possible to create any new security primitive based on hard AI problems? This is a challenging and interesting open problem.

In this paper, introducing a new security primitive based on hard AI problems, namely, a novel family of graphical password systems integrating Captcha technology, which we call *CaRP (Captcha as gRaphical Passwords)*. CaRP is click-based graphical passwords, where a sequence of clicks on an image is used to derive a password. Unlike other click-based graphical passwords, images used in CaRP are Captcha challenges, and a new CaRP image is generated for every login attempt. The notion of CaRP is simple but generic. CaRP can have multiple instantiations. In theory, any Captcha scheme relying on multiple-object classification can be converted to a CaRP scheme. We present exemplary CaRPs built on both text Captcha and image-recognition Captcha. One of them is a text CaRP wherein a password is a sequence of characters like a text password, but entered by clicking the right character sequence on CaRP images.

CaRP offers protection against online dictionary attacks on passwords, which have been for long time a major security threat for various online services. This threat is widespread and considered as a top cyber security risk. Defense against online dictionary attacks is a more subtle problem than it might appear. Intuitive countermeasures such as throttling logon attempts do not work well for two reasons:

1) It causes denial-of-service attacks (which were exploited to lock highest bidders out in final minutes of eBay auctions and incurs expensive helpdesk costs for account reactivation.

2) It is vulnerable to global password attacks whereby adversaries intend to break into any account rather than a specific one, and thus try each password candidate on multiple accounts and ensure that the number of trials on each account is below the threshold to avoid triggering account lockout.

CaRP also offers protection against relay attacks, an increasing threat to bypass Captchas protection, wherein Captcha challenges are relayed to humans to solve. Koobface Was a relay attack to bypass Facebook's Captcha in creating New accounts. CaRP is robust to shoulder-surfing attacks if combined with dual-view technologies.

CaRP requires solving a Captcha challenge in every login. This impact on usability can be mitigated by adapting the CaRP image's difficulty level based on the login history of the account and the machine used to log in.

Typical application scenarios for CaRP include:

1) *E-Banking*: CaRP can be applied on touch-screen devices whereon typing passwords is cumbersome, esp. for secure Internet applications such as e-banks. Many e-banking systems have applied Captchas in user logins. For example, ICBC (www.icbc.com.cn), the largest bank in the world, requires solving a Captcha challenge for every online login attempt.

2) *Spam mitigation*: CaRP increases spammer's operating cost and thus helps reduce spam emails. For an email service provider that deploys CaRP, a spam bot cannot log into an email account even if it knows the password. Instead, human involvement is compulsory to access an account. If CaRP is combined with a policy to throttle the number of emails sent to new recipients per login session, a spam bot can send only a limited number of emails before asking human assistance for login, leading to reduced outbound spam traffic.

3*) Cross-device authentication:* Typing passwords is cumbersome on touch devices such as Smartphone's and tablets, where click/touch-based input is convenient. CaRP can offer the same password entry experience across different types of devices, including desktops, Smartphone's and tablets. Therefore, it is inherently a cross-device authentication mechanism, and a single implementation can simultaneously serve a wide range of different devices. On the contrary, text passwords are more friendly to desktop users, but less so to Smartphone or tablet users.

A. CAPTCHA AS GRAPHICAL PASSWORDS

A. A New Way to Thwart Guessing Attacks

In a guessing attack, a password guess tested in an unsuccessful trial is determined wrong and excluded from subsequent trials. The number of undetermined password guesses decreases with more trials, leading to a better chance of finding the password. Mathematically, let S be the set of password guesses before any trial, ρ be the password to find, T denote a trial whereas T_n denote the n-th trial, and $p(T = \rho)$ be the probability that ρ is tested in trial T . Let E_n be the set of password guesses tested in trials up to (including) T_n . The password guess to be tested in n-th trial T_n is from set $S\backslash E_{n-1}$, i.e., the relative complement of E_{n-1} in S. If $\rho \in S$, then we have

$$p\,(T = \rho | T_1 \_= \rho, \ldots, T_n{}_{-1} \neq \rho\,) > p(T = \rho), \quad (1)$$

and

$$E_n \rightarrow S$$

$$p(T = \rho | T_1 \neq \rho, \ldots, T_{n-1} \neq \rho) \rightarrow 1 \quad \}\text{with n}\rightarrow\text{s} \quad (2)$$

where $|S|$ denotes the cardinality of S. From Eq. (2), the password is always found within $|S|$ trials if it is in S; otherwise S is exhausted after $|S|$ trials. Each trial determines if the tested password guess is the actual password or not, and the trial's result is deterministic.

To counter guessing attacks, traditional approaches in designing graphical passwords aim at increasing the effective password space to make passwords harder to guess and thus require more trials. No matter how secure a graphical password scheme is, the password can always be found by a brute force attack. In this paper, we distinguish two types of guessing attacks: *automatic guessing attacks* apply an automatic trial and error process but S can be manually constructed whereas *human guessing attacks* apply a manual trial and error process.

CaRP adopts a completely different approach to counter automatic guessing attacks. It aims at realizing the following equation:

$$p(T = \rho | T_1, \ldots, T_{n-1}) = p(T = \rho), \quad \forall\, n \quad (3)$$

In an automatic guessing attack. Eq. (3) means that each trial is computationally independent of other trials. Specifically, no matter how many trials executed previously, the chance of finding the password in the current trial always remains the same. That is, a password in S can be found only *probabilistically* by automatic guessing (including brute-force) attacks, in contrast to existing graphical password schemes where a password can be found within a fixed number of trials.

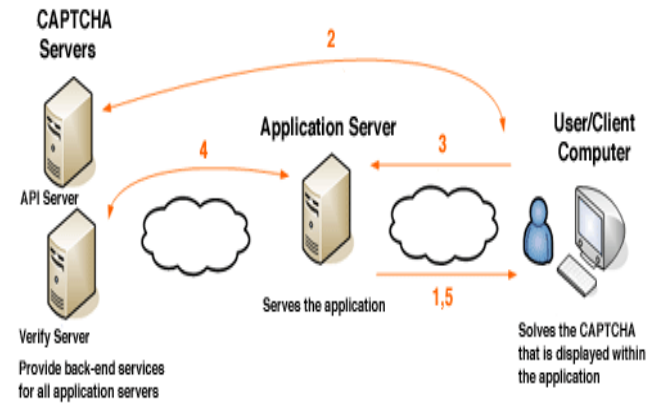
How to achieve the goal? If a new image is used for each trial, and images of different trials are independent of each other, then Eq. (3) holds. Independent images among different login attempts must contain invariant information so that the authentication server can verify claimants. By examining the ecosystem of user authentication, we noticed that human users enter passwords during authentication, whereas the trial and error process in guessing attacks is executed automatically. The capability gap between humans and machines can be exploited to generate images so that they are *computationally-independent* yet retain invariants that only humans can identify, and thus use as passwords. The invariants among images must be intractable to machines to thwart automatic guessing attacks. This requirement is the same as that of an ideal Captcha [25], leading to creation of CaRP, a new family of graphical passwords robust to online guessing attacks.

B. CaRP: An Overview

In CaRP, a new image is generated for every login attempt, even for the same user. CaRP uses an *alphabet* of visual objects (e.g., alphanumerical characters, similar animals) to generate a CaRP image, which is also a Captcha challenge. A major difference between CaRP images and Captcha images is that all the visual objects in the alphabet should appear in a CaRP image to allow a user to input any password but not necessarily in a Captcha image. Many Captcha schemes can be converted to CaRP schemes, as described in the next subsection.

CaRP schemes are clicked-based graphical passwords. According to the memory tasks in memorizing and entering a password, CaRP schemes can be classified into two categories: recognition and a new category, *recognition-recall*, which requires recognizing an image and using the recognized objects as cues to enter a password. Recognition-recall combines the tasks of both recognition and cued-recall, and retains both the recognition-based advantage of being easy for human memory and the cued-recall advantage of a large password space. Exemplary CaRP schemes of each type will be presented later.

1) SYSTEM ARCHITECTURE:



User Authentication with CaRP Schemes

Like other graphical passwords, we assume that CaRP schemes are used with additional protection such as secure channels between clients and the authentication server through Transport Layer Security (TLS). A typical way to apply CaRP schemes in user authentication is as follows. The authentication server *AS* stores a salt *s* and a hash value $H(\rho, s)$ for each user ID, where ρ is the password of the account and not stored. A CaRP password is a sequence of visual object IDs or clickable-points of visual objects that the user selects. Upon receiving a login request, *AS* generates a CaRP image, records the locations of the objects in the image, and sends the image to the user to click her password. The coordinates of the clicked points are recorded and sent to *AS* along

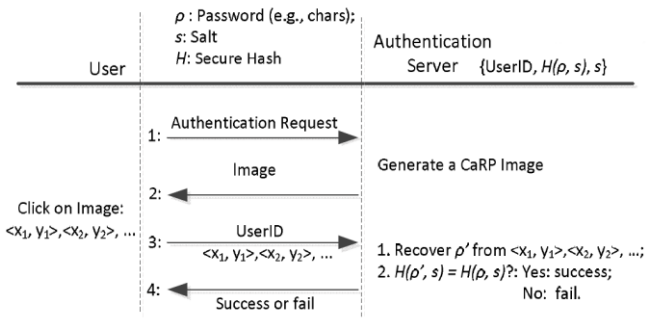


Fig. 1. Flowchart of basic CaRP authentication.

To recover a password successfully, each user-clicked point must belong to a single object or a clickable-point of an object. Objects in a CaRP image may overlap slightly with neighboring objects to resist segmentation. Users should not click inside an overlapping region to avoid ambiguity in identifying the clicked object. This is not a usability concern in practice since overlapping areas generally take a tiny portion of an object.

Security of user authentication web applications relies on two parties: Hypertext Transfer Protocol Secure (HTPS) and Hyper Text Transfer Protocol (HTTP) Authentication. It builds a secure communication channel between client and Web Server.

The CaRP schemes in some embodiments such as CaRP for Web Applications may be implemented to work with the widely used HTTP Basic Authentication. Note that for both HTTP authentications, TLS/SSL provides the communication security between a client and a web server.

It is a flow diagram of an illustrative process for implementing CaRP with internet protocols such as HTTP Basic Access Authentication .The process Involves acts implemented at a client and security administrator which may be embodied in a web server .The security administrator may store User Id and Password indicators.

At security Administrator receives an access request from the client. The access request may optionally include the user's user ID.

At security administrator generates a CaRP image. In some embodiments, the CaRP image includes all of the potential password-elements that make up the primitive domain. In other embodiments the CaRP image includes less than all of the potential password elements that make up the primitive domain.

The security administrator sends the CaRP image to the client .A user selects regions/points on the received image to enter the user's password.

The coordinates [(x1,y1),(x2,y2)….(xk,yk)] of the selected points on the CaRP image are provided to the security administrator .In some embodiments the user's userID is also provided, using the HTTP Basic Authentication, to the security administrator with the coordinates [(x1,y1),(x2,y2)…(xk,yk)] of the selected points on the CaRP image.

The security administrator may map the coordinates back on the CaRP image previously provided to the client and recover a sequence of purported password-elements, p´ that the user selects on the CaRP image at the client. The Administrator may determine whether the sequence of password-elements that makes up the user's password.

For Example, the security administrator may retrieve the salt s of the associated with the user-ID; calculate a hash value stored for the account.

Access is granted only if the purported password is verified to be the same as the actual password, e.g., hash value of the password agrees with the hash value stored for the account.

The security administrator denies access, then the process may return to back, and different CaRP image is generated .Each generated CaRP image is computationally uncorrelated with another CaRP image.

The security Administrator may deny access after a number of failed attempts.

In some embodiments, the security Administrator has the capability to recover a user's password p in a successful authentication procedure. The Security admin does not required storing the user passwords. Instead, only a password indicator for a password may be recorded. Further security measure can be taken to make hard or impossible for the security admin to recover passwords.

II. CARP SCHEMES

1. RECOGNITION-BASED CARP

For this type of CaRP, a password is a sequence of visual objects in the alphabet. Per view of traditional recognition-based graphical passwords, recognition-based CaRP seems to have access to an infinite number of different visual objects. We present two recognition-based CaRP schemes and a variation next.

A. ClickText

ClickText is a recognition-based CaRP scheme built on top of text Captcha. Its alphabet comprises characters without any visually-confusing characters. For example, Letter "O" and digit "0" may cause confusion in CaRP images, and thus one character should be excluded from the alphabet. A ClickText password is a sequence of characters in the alphabet, e.g., ρ ="AB#9CD87", which is similar to a text password. A ClickText image is generated by the underlying Captcha engine as if a Captcha image were generated except that all the alphabet characters should appear in the image. During generation, each character's location is tracked to produce ground truth for the location of the character in the generated image. The authentication server relies on the ground truth to identify the characters corresponding to user-clicked points. In ClickText images, characters can be arranged randomly

Fig. 2. A ClickText image with 33 characters

on 2D space. This is different from text Captcha challenges in which characters are typically ordered from left to right in order for users to type them sequentially. Fig. 2 shows a ClickText image with an alphabet of 33 characters. In entering a password, the user clicks on this image the characters in her password, in the same order, for example "A", "B", "#", "9", "C", "D", "8", and then "7" for password ρ = "AB#9CD87".

B. ClickAnimal

Captcha Zoo is a Captcha scheme which uses 3D models of horse and dog to generate 2D animals with different textures, colors, lightings and poses, and arranges them on a cluttered background. A user clicks all the horses in a challenge image to pass the test. Fig. 3 shows a sample challenge wherein all the horses are circled red

Fig. 3. Captcha Zoo with horses circled red.

ClickAnimal is a recognition-based CaRP scheme built on top of Captcha Zoo with an alphabet of similar animals such as dog, horse, pig, etc. Its password is a sequence of animal names such as ρ = "Turkey, Cat, Horse, Dog…" For each animal, one or more 3D models are built. The Captcha generation process is applied to generate ClickAnimal images: 3D models are used to generate 2D animals by applying different views, textures, colors, lightning effects, and optionally distortions. The resulting 2D animals are then arranged on a cluttered background such as grassland. Some animals may be occluded by other animals in the image, but their core parts are not occluded in order for humans to identify each of them. Fig. 4 shows a ClickAnimal image with an alphabet of 10 animals. Note that different views applied in mapping 3D models to 2D animals, together with occlusion in

the following step, produce many different shapes for the same animal's instantiations in the generated images. Combined with the additional anti-recognition mechanisms applied in the mapping step, these make it hard for computers to recognize animals in the generated image, yet humans can easily identify different instantiations of animals.

C. AnimalGrid

The number of similar animals is much less than the number of available characters. ClickAnimal has a smaller alphabet, and thus a smaller password space, than ClickText. CaRP should have a sufficiently-large effective password space to resist human guessing attacks. AnimalGrid's password space can be increased by combining it with a grid-based graphical password, with the grid depending on the size of the selected animal.

DAS is a candidate but requires drawing on the grid. To be consistent with ClickAnimal, we change from drawing to clicking: *Click-A-Secret (CAS)* wherein a user clicks the grid cells in her password. *AnimalGrid* is a combination of ClickAnimal and CAS. The number of grid-cells in a grid should be much larger than the alphabet size. Unlike DAS, grids in our CAS are object-dependent, as we will see next. It has the advantage that a correct animal should be clicked in order for the clicked grid-cell(s) on the follow-up grid to be correct. If a wrong animal is clicked, the follow-up grid is wrong. A click on the correctly labeled grid-cell of the wrong grid would likely produce a wrong grid-cell at the authentication server side when the correct grid is used.

To enter a password, a ClickAnimal image is displayed first. After an animal is selected, an image of $n \times n$ grid appears, with the grid-cell size equaling the bounding rectangle of the selected animal. Each grid-cell is labeled to help users identify. Fig. 4 shows a 6×6 grid when the red turkey in the left image of Fig. 4 was selected. A user can select zero to multiple grid-cells matching her password. Therefore a password is a sequence of animals interleaving with grid-cells, e.g., ρ = "Dog, Grid(2), Grid(1); Cat, Horse, Grid(3)", where Grid(1) means the grid-cell indexed as 1, and grid-cells after an animal means that the grid is determined by the bounding rectangle of the animal. A password must begin with an animal.

When a ClickAnimal image appears, the user clicks the animal on the image that matches the first animal in her password. The coordinates of the clicked point are recorded. The bounding rectangle of the clicked animal is then found

interactively as follows: a bounding rectangle is calculated and displayed, e.g., the white rectangle shown in Fig. 4.

Fig. 4. A ClickAnimal image (left) and 6 × 6 grid (right) determined by red turkey's bounding rectangle

The user checks the displayed rectangle and corrects inaccurate edges by dragging if needed. This process is repeated until the user is satisfied with the accuracy of the bounding rectangle. In most cases, the calculated bounding rectangle is accurate enough without needing manual correction.

Once the bounding rectangle of the selected animal is identified, an image of $n \times n$ grid with the identified bounding rectangle as its grid-cell size is generated and displayed. If the grid image is too large or too small for a user to view, the grid image is scaled to a fitting size. The user then clicks a sequence of zero to multiple grid-cells that match the grid-cells following the first animals in her password, and then gets back to the ClickAnimal image. For the example password ρ given previously, she clicks a point inside grid-cell(2), and then a point inside grid-cell(1) to select the two grid-cells. The coordinates of user-clicked points on the grid image (the original one before scaling if the grid image is scaled) are recorded. The above process is repeated until the user has
finished entering her password. The resulting sequence of coordinates of user-clicked points, e.g., "AP(150,50), GP(30,66), GP(89,160), AP(135,97),…" where "AP(x,y)" denotes the point with coordinates (x,y) on a ClickAnimal image, and "GP(x,y)" denotes the point with coordinates (x,y) on a grid image, is sent to the authentication server.

Using the ground truth, the server recovers the first animal from the received sequence, regenerates the grid image from the animal's bounding rectangle, and recovers the clicked grid-cells. This process is repeated to recover the password the user clicked. Its hash is then calculated and compared with the stored hash.

2. RECOGNITION-RECALL CARP

In recognition-recall CaRP, a password is a sequence of some invariant points of objects. An *invariant point* of an object (e.g. letter "A") is a point that has a fixed relative position in different incarnations (e.g., fonts) of the object, and thus can be uniquely identified by humans no matter how the object appears in CaRP images. To enter a password, a user must identify the objects in a CaRP image, and then use the identified objects as cues to locate and click the invariant points matching her password. Each password point has a tolerance range that a click within the tolerance range is acceptable as the password point. Most people have a click variation of 3 pixels or less [18]. TextPoint, a recognition-recall CaRP scheme with an alphabet of characters, is presented next, followed by a variation for challenge-response authentication.

A. TextPoints

Characters contain invariant points. Fig. 5 shows some invariant points of letter "A", which offers a strong cue to memorize and locate its invariant points. A point is said to be an *internal point* of an object if its distance to the closest boundary of the object exceeds a threshold. A set of internal invariant points of characters is selected to form a set of *clickable points* for TextPoints. The internality ensures that a clickable point is unlikely occluded by a neighboring character and that its tolerance region unlikely overlaps with any tolerance region of a neighboring character's clickable points on the image generated by the underlying Captcha engine. In determining clickable points, the distance between any pair of clickable points in a character must exceed a threshold so that they are perceptually distinguishable and their tolerance regions do not overlap on CaRP images. In addition, variation should also be taken into consideration. For example, if the center of a stroke segment in one character is selected, we should avoid selecting the center of a similar stroke segment in another character. Instead, we should select

Fig. 5. Some invariant points (red crosses) of "A".

a different point from the stroke segment, e.g., a point at one-third length of the stroke segment to an end. This variation in selecting clickable points ensures that a clickable point is context-dependent: a similarly structured point may or may not be a clickable point, depending on the character that the point lies in. Character recognition is required in locating clickable points on a TextPoints image although the clickable points are known for each character. This is a task beyond a bot's capability.

A password is a sequence of clickable points. A character can typically contribute multiple clickable points. Therefore TextPoints has a much larger password space than ClickText.

Image Generation. TextPoints images look identical to ClickText images and are generated in the same way except that the locations of all the clickable points are checked to ensure that none of them is occluded or its tolerance region overlaps another clickable point's. We simply generate another image if the check fails. As such failures occur rarely due to the fact that clickable points are all internal points, the restriction due to the check has a negligible impact on the security of generated images.

Authentication. When creating a password, all clickable points are marked on corresponding characters in a CaRP image for a user to select. During authentication, the user first identifies her chosen characters, and clicks the password points on the right characters. The authentication server maps each user-clicked point on the image to find the closest clickable point. If their distance exceeds a tolerable range, login fails. Otherwise a sequence of clickable points is

recovered, and its hash value is computed to compare with the stored value.

It is worth comparing potential password points between TextPoints and traditional click-based graphical passwords such as PassPoints In PassPoints, salient points should be avoided since they are readily picked up by adversaries to mount dictionary attacks, but avoiding salient points would increase the burden to remember a password. This conflict does not exist in TextPoints. Clickable points in TextPoints are salient points of their characters and thus help remember a password, but cannot be exploited by bots since they are both *dynamic* (as compared to static points in traditional graphical password schemes) and *contextual*:

- **Dynamic:** locations of clickable points and their contexts (i.e., characters) vary from one image to another. The clickable points in one image are computationally inde-pendent of the clickable points in another image, as we will see in Section VI-B.
- **Contextual:** Whether a similarly structured point is a clickable point or not depends on its context. It is only if within the right context, i.e., at the right location of a right character.

These two features require recognizing the correct contexts, i.e., characters, first. By the very nature of Captcha, recognizing characters in a Captcha image is a task beyond computer's capability. Therefore, these salient points of characters cannot be exploited to mount dictionary attacks on TextPoints.

B. TextPoints4CR

For the CaRP schemes presented up to now, the coordinates of user-clicked points are sent directly to the authentication server during authentication. For more complex protocols, say a challenge-response authentication protocol, a response is sent to the authentication server instead. TextPoints can be modified to fit challenge-response authentication. This variation is called
TextPoints for Challenge-Response or *TextPoints4CR*.
Unlike TextPoints wherein the authentication server stores a salt and a password hash value for each account, the server in TextPoints4CR stores the password for each account. Another difference is that each character appears only once in a TextPoints4CR image but may appear multiple times in a TextPoints image. This is because both server and client in TextPoints4CR should generate the same sequence of discretized grid-cells independently. That requires a unique way to generate the sequence from the shared secret, i.e., password. Repeated characters would lead to several possible sequences for the same password. This unique sequence is used as if the shared secret in a conventional challenge-response authentication protocol.

In TextPoints4CR, an image is partitioned into a fixed grid with the discretization grid-cell of size μ along both directions. The minimal distance between any pair of clickable points should be larger than μ by a margin exceeding a threshold to prevent two clickable points from falling into a single grid-cell in an image. Suppose that a guaranteed tolerance of click errors along both x-axis and y-axis is τ , we require that $\mu \geq 4\tau$.

Image Generation. To generate a TextPoints4CR image, the same procedure to generate a TextPoints image is applied. Then the following procedure is applied to make every click-able point at least τ distance from the edges of the grid-cell it lies in. All the clickable points, denoted as set f, are located on the image. For every point in f, we calculate its distance along x-axis or y-axis to the center of the grid-cell it lies in. A point is said to be an *internal point* if the distance is less than $0.5\mu-\tau$ along both directions; otherwise a *boundary point*. For each boundary point in f, a nearby internal point in the same grid-cell is selected. The selected point is called a *target point* of the boundary point. After processing all the points in f, we obtain a new set f^- comprising internal points; these are either internal clickable points or target points of boundary clickable points. Mesh warping , widely used in generating text Captcha challenges, is then used to warp the image so that f maps to f^- . The result is a TextPoint4CR image wherein every clickable point would tolerate at least τ of click errors. Selection of target points should try to reduce warping distortion caused by mapping f to f^-.

Authentication. In entering a password, a user-clicked point is replaced by the grid-cell it lies in. If click errors are within τ, each user-clicked point falls into the same grid-cell as the original password point. Therefore the sequence of grid-cells generated from user-clicked points is identical to the one that the authentication server generates from the stored password of the account. This sequence is used as if the shared secret between the two parties in a challenge-response authentication protocol.
Unlike other CaRP schemes presented in this paper, Text-Points4CR requires the authentication server to store pass-words instead of their hash values. Stored passwords must be protected from insider attacks; for example, they are encrypted with a master key that only the authentication server knows. A password is decrypted only when its associated account attempts to log in.

III. SECURITY ANALYSIS

A. Security of Underlying Captcha

Computational intractability in recognizing objects in CaRP images is fundamental to CaRP. Existing analyses on Captcha security were mostly case by case or used an approximate process. No theoretic security model has been established yet. Object segmentation is considered as a computationally-expensive, combinatorial-hard problem [30], which modern text Captcha schemes rely on. According to [30], the complexity of object segmentation, C, is exponentially dependent of the number M of objects contained in a challenge, and polynomially dependent of the size N of the Captcha alphabet: $C = \alpha^{M} P(N)$, where $\alpha > 1$ is a parameter, and $P()$ is a polynomial function. A Captcha challenge typically contains 6 to 10 characters, whereas a CaRP image typically contains 30 or more characters. The complexity to break a Click-Text image is about $\alpha^{30} P(N)/(\alpha^{10} P(N)) = \alpha^{20}$ times the complexity to break a Captcha challenge generated by its underlying Captcha scheme. Therefore ClickText is much harder to break than its underlying Captcha scheme. Fur-the rmore, characters in a CaRP scheme are arranged two-dimensionally, further increasing segmentation difficulty due to one more dimension

to segment. As a result, we can reduce distortions in ClickText images for improved usability yet maintain the same security level as the underlying text Captcha. ClickAnimal relies on both object segmentation and multiple-label classification. Its security remains an open question.

As a framework of graphical passwords, CaRP does not rely on any specific Captcha scheme. If one Captcha scheme gets broken, a new and more robust Captcha scheme may appear and be used to construct a new CaRP scheme. In the remaining security analysis, we assume that it is intractable for computers to recognize any objects in any challenge image generated by the underlying Captcha of CaRP. More accurately, the Captcha is assumed to be *chosen-pixel attack (CPA)*-secure defined with the following experiment: an adversary *A* first learns from an arbitrary number of challenge images by querying a ground-truth oracle *O* as follows: *A* selects an arbitrary number of internal object-points and sends to *O*, which responds with the object that each point lies in. Then *A* receives a new challenge image and selects an internal object-point to query *O* again.

This time *O* chooses a random bit $b \leftarrow \{0, 1\}$ to determine what to return: It returns the true object if $b = 1$; otherwise a false object selected with a certain strategy. *A* is asked to determine whether the returned object is the true object that the internal object-point lies in or not. A Captcha scheme is said to be *CPA-secure* if *A* cannot succeed with a probability non-negligibly higher than ½, the probability of a random guess.

CONCLUSION

They have proposed CaRP, a new security primitive relying on unsolved hard AI problems. CaRP is both a Captcha and a graphical password scheme. The notion of CaRP introduces a new family of graphical passwords, which adopts a new approach to counter online guessing attacks: a new CaRP image, which is also a Captcha challenge, is used

Overall, our work is one step forward in the paradigm of using hard AI problems for security. Of reasonable security and usability and practical applications, CaRP has good potential for refinements, which call for useful future work. More importantly, we expect CaRP to inspire new inventions of such AI based security primitives.

REFERENCES

1. R. Biddle, S. Chiasson, and P. C. van Oorschot, "Graphical passwords: Learning from the first twelve years," *ACM Comput. Surveys*, vol. 44, no. 4, 2012.
2. (2012, Feb.). *The Science Behind Passfaces* [Online]. Available:http://www.realuser.com/published/ScienceBehindPassfaces.pdf
3. I. Jermyn, A. Mayer, F. Monrose, M. Reiter, and A. Rubin, "The design and analysis of graphical passwords," in *Proc. 8th USENIX Security Symp.*, 1999, pp. 1–15.
4. H. Tao and C. Adams, "Pass-Go: A proposal to improve the usability of graphical passwords," *Int. J. Netw. Security*, vol. 7, no. 2, pp. 273–292, 2008.
5. S. Wiedenbeck, J. Waters, J. C. Birget, A. Brodskiy, and N. Memon, "PassPoints: Design and longitudinal evaluation of a graphical password system," *Int. J. HCI*, vol. 63, pp. 102–127, Jul. 2005.