Purely Automated Attacks on PassPoints-Style Graphical Passwords*

P.C. van Oorschot, Amirali Salehi-Abari, Julie Thorpe
School of Computer Science, Carleton University

Abstract

We introduce and evaluate various methods for purely automated attacks against PassPoints-style graphical passwords. For generating these attacks, we introduce a graph-based algorithm to efficiently create dictionaries based on heuristics such as click-order patterns (e.g., 5 points all along a line). Some of our methods combine click-order heuristics with focus-of-attention scan-paths generated from a computational model of visual attention, yielding significantly better automated attacks than previous work. One resulting automated attack finds 7-16% of passwords for two representative images using dictionaries of approximately 2^{26} entries (where the full password space is 2^{43}). Relaxing click-order patterns substantially increased the attack efficacy albeit with larger dictionaries of approximately 2^{35} entries, allowing attacks that guessed 48-54% of passwords (compared to previous results of 1% and 9% on the same dataset for two images with 2^{35} guesses). These latter attacks are independent of focus-of-attention models, and are based on image-independent guessing patterns. Our results show that automated attacks, which are easier to arrange than human-seeded attacks and are more scalable to systems that use multiple images, pose a significant threat to basic PassPoints-style graphical passwords.

1 Introduction

Graphical passwords are an alternative to text passwords, whereby a user is asked to remember an image (or parts of an image) instead of a word. They are motivated in part by the well-known fact that people have superior memorability for images [19], and the promise of their suitability for small devices such as smart phones. Graphical passwords have become an active topic of research with many new proposals (see surveys [21, 29, 26]). One proposal of interest, *PassPoints* [37, 35], involves a user creating a 5-point click sequence on a background image. Usability studies have indicated that these graphical passwords have reasonable login and creation times, acceptable error rates, decent general perception [36, 37, 5], and less interference between multiple passwords [20, 11, 4] when compared to text passwords.

Our research improves our understanding of the security of *PassPoints-style graphical passwords*, i.e., schemes closely resembling PassPoints, wherein a user creates a click sequence of r points (e.g., $r = 5$) on a single background image. PassPoints-style graphical passwords have been shown to be susceptible to hot-spots, which can be exploited in *human-seeded attacks* [31, 32], whereby human-computed data (harvesting click-points from a small set of users) is used to facilitate efficient attacks. These attacks require that the attacker collect sufficient "human-computed" data for the

*Version: March 02, 2010. This work extends and updates a preliminary paper [28] and June 2008 Technical Report TR-08-15 in which parts of this work appeared. Authors listed alphabetically. Contact author: J. Thorpe (*jthorpe@scs.carleton.ca*).

target image, which is more costly for systems with multiple images. This leads us to ask whether more scalable attacks exist, and in particular, effective fully-automated attacks.

To address this question, in the present work we introduce and evaluate a set of purely automated attacks against PassPoints-style graphical passwords. Our attack method is based on the hypothesis that users are more likely to choose click-points relating to predictable preferences, e.g., logically grouping the click-points through a click-order pattern (such as five points in a straight line), and/or choosing click-points in the areas of the image that their attention is naturally drawn towards. To find parts of the image that users are more likely to attend to (salient parts of the image), we use Itti et al.'s [16] model of visual attention. We also examine click-order patterns both alone and in combination with these more salient parts of the image.

Our attacks employ sets of graphical passwords that we hypothesize are "more likely" to be chosen than others; these sets naturally define dictionaries for use in a dictionary attack. A successful such attack must be able to *efficiently* generate a dictionary containing highly probable passwords. In existing literature, the *size* of a dictionary is normally considered the most important cost for a dictionary attack, whereas the cost of dictionary generation is often neglected; the latter is reasonable if a one-time precomputation can be reused. Alternately, if the dictionary must be generated on-the-fly, or recomputed each time (e.g., for a different background image), then the cost of dictionary generation may become as or more important than the size of the dictionary itself.

We present a graph-based algorithm for attack dictionary generation whose computational cost is on the order of the number of dictionary entries. It can efficiently generate r-permutations of a given alphabet, which satisfy a predefined set of conditions (e.g., click-order heuristics), as opposed to generating all possible r-permutations and then checking which one satisfies a predefined condition.

Our methods are substantially more successful than previous purely automated attacks. Tested on the field study database of Thorpe et al. [31], some of our dictionaries find 19-83% as many passwords as human-seeded attacks (when based on independent probabilities), with only about 3% as many dictionary entries.

Our contributions include the best purely automated attacks to date against PassPoints-style graphical passwords, an evaluation of how the model of Itti et al. [16] relates to user-selected click-based graphical passwords, an efficient heuristic dictionary generation algorithm, and a new spatial clustering algorithm for grouping nearby click-points. We found that a "lazy" approach to click-order patterns (e.g., loosening the definition of a line to five points lying within some tolerance of a line) produced a substantially more successful automated attack than previous methods with comparable dictionary sizes and images [31, 10]. A click-order pattern based only on locality (all five points being near one another) was also very successful. We show how some click-order pattern dictionaries can be further optimized by applying distance constraints. Furthermore, we were able to improve these click-order pattern dictionaries using Itti's model, producing dictionaries whose relative guessing accuracy is better than human-seeded attacks based on independent probabilities [32]. These results show that purely automated attacks pose a significant threat to PassPoints-style graphical passwords.

The remainder of this paper proceeds as follows. Section 2 discusses background and terminology, including computational models of visual attention. We describe purely automated attack generation methods in Section 3, results in Section 4, related work in Section 5, and a discussion with concluding remarks in Section 6.

2 Background

Our original idea was that a significant percentage of users will choose points relating to the order that the points draw their visual attention as components of their click-based passwords, and thus that computational models of visual attention may help identify more probable click-points. As our attacks are motivated by this idea and a subset of them use computational models of visual attention, we provide some background on models of visual attention in Section 2.1. Other terminology is presented in Section 2.2.

2.1 Models of Visual Attention

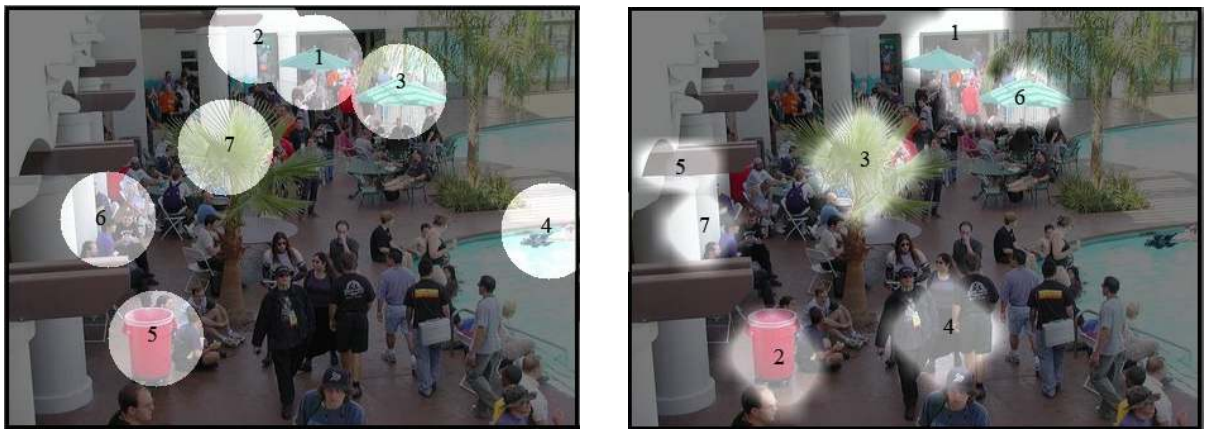
Computational models of bottom-up visual attention are normally defined by features of a digital image, such as intensity, color, and orientation [16, 15]. Feature maps are then created and used to generate a *saliency map*, which is a grayscale image where higher-intensity locations define more conspicuous areas.

Computational models of top-down visual attention can be defined by training [23]. The difficulty of these models is that the top-down task must be pre-defined (e.g., find all people in the image), and then a corpus of images that are tagged with the areas containing the subject to find (e.g., people) must be used for training. Navalpakkam et al. [22] discuss an alternate method to create a top-down model, based on *guided search* [38], which weighs visual feature maps according to the top-down task. For example, with a task of locating a red object, a red-sensitive feature map would gain more weight, giving it a higher value in the resulting saliency map. In both cases, assumptions regarding what sort of objects people are looking for are required to create such a model.

In this work, we focus on bottom-up visual attention, using the computational model of visual attention of Itti et al. [16]. We chose this model as it is well-known, and there is empirical evidence that it captures people's bottom-up visual attention [24]. The general idea is that areas of an image will be *salient* (or visually "stand out") when they differ from their surroundings. Given an input image, Itti's model outputs a focus-of-attention *scan-path* to model the locations and the order in which a human might automatically and unconsciously attend these parts of the image. The model first constructs a saliency map based on visual features. Then it uses a winner-take-all neural network with inhibition of return to define a specific focus-of-attention scan-path, intended to represent the order in which a user would scan the image.

In stage 1, the saliency map is created by decomposing the original image into a set of 50 multi-level "feature maps", which extract spatial discontinuities based on color opponency (either red-green or blue-yellow), intensity, or orientation. Each level defines a different size of the center and its surround, in order to account for conspicuous locations of various sizes. All feature maps are then combined into a single saliency map.

In stage 2, the neural network detects the point of highest salience (as indicated by the intensity value of the saliency map), and draws the focus of attention towards this location. Once an area has been attended to, inhibition of return will prevent the area from being the focus again for a period of time. Together, the neural network with inhibition of return produces output in the form of spatio-temporal attentional scan-paths, which follow the order of decreasing saliency as defined by stage 1. Two different normalization types (producing different scan-paths) can be used with the model: *LocalMax* and *Iterative* (cf. Figure 1). In LocalMax normalization, the neural network will have a bias towards those areas that are closer to the previously attended location. In Iterative normalization, the neural network will find the next most salient area that has not been inhibited. We use LocalMax herein (Iterative was found to have inferior performance [28]).



(a) LocalMax normalization (b) Iterative normalization

Figure 1: *pool* image (unmodified version from [35, 37]) with the first 7 steps in the scan-path.

2.2 Terminology and Identifying Distinguishable Points

Our overall method is based on the hypothesis that users are more likely to choose passwords consisting of click-points, each of which is a *distinguishable point*, defined as a point on a digital image that can be easily distinguished and relocated by a user – e.g., by using *referencable* points on the image (such as a corner), or *calculable* points based on other referencable parts of the image (such as object centers). Thorpe et al. [31] used corner detection to find referencable points, and Dirik et al. [10] used centroids to find calculable points. We use both approaches to define a *distinguishable points map* δ, as detailed below.

Regarding additional terminology, suppose that a user chooses a click-point c as part of a password. The *tolerable error* or *tolerance* t is the error allowed (in vertical and horizontal directions) for a click-point entered on a subsequent login to be accepted as c. This defines a *tolerance region (T-region)* centered on c. For an implementation using $t = 9$ pixels, the T-region is a 19×19 pixel square, matching the dataset [31], used herein for evaluation and comparison.

A *window cluster* is a square region of $n \times n$ pixels for some positive integer n. A *cluster* is a set of one or more points that lie within a window cluster. The geometric center of a window cluster is used as the representative of all the points within the window cluster. An *alphabet* is a set of distinct window centers.

Corner Detection. A *corner* is defined as the intersection of two edges, where an *edge* is defined by the points in a digital image where there are sharp changes in intensity [12]. We use Harris corner detection [13] as implemented by Kovesi [18]. This first identifies the edges. Those edges are then blurred to reduce the effect of any noise. Next, based on the edges, an *energy map* is generated, containing local maxima and minima. A local maximum indicates the presence of a corner. We use the recommended parameters $\sigma = 1$, $\theta = 1000$ and $r = 3$, where σ is the standard deviation of a smoothing Gaussian filter, θ is a threshold for the maximum number of corners, and r is an inhibition radius, measured in pixels around a detected corner. Figure 2 shows the *pool* image where each detected corner is illustrated by a '+'.

We also create a *binary corners map* (a specialized type of *binary map* or one-to-one mapping from its pixels of value 0 or 1 to the pixels of the original image): when a pixel is a corner in the original image, its corresponding value is 1; otherwise 0.

Centroid Detection. To find the centers of objects, we first partition the digital image into segments using *image segmentation*, by the mean-shift segmentation algorithm [7], which takes a feature (range) bandwidth, spatial bandwidth, and a minimum region area (in pixels) as input.

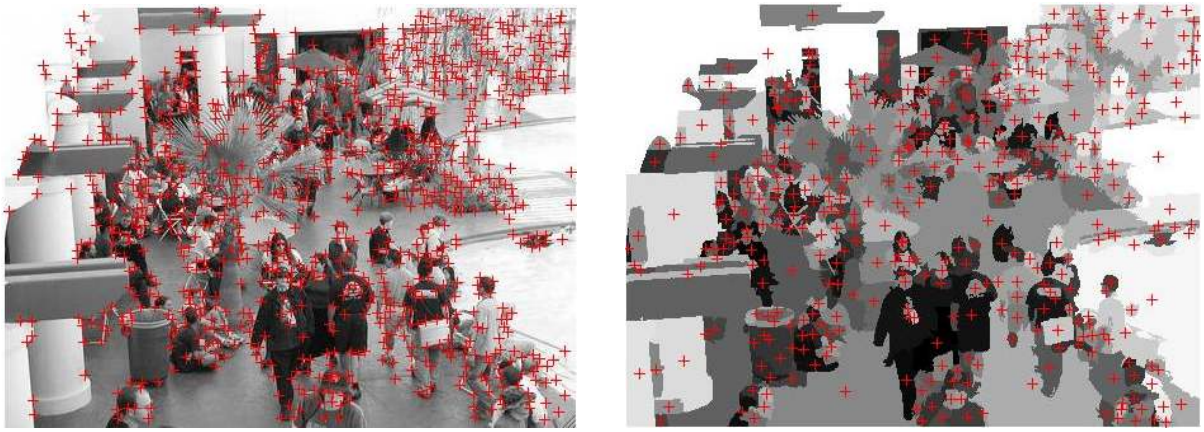


Figure 2: Corner detection (left) and center detection (right) for *pool* image.

We set these parameters to 7, 9, and 50 respectively, which we found empirically to provide an acceptable segmentation with the smallest resulting number of segments.

After segmentation, we calculate the center (X_S, Y_S) of each segment (centroid) S from the arithmetic mean of each coordinate of the segment's points: $X_S = \frac{1}{n(S)} \sum_{i \in S} x_i$ and $Y_S = \frac{1}{n(S)} \sum_{i \in S} y_i$. Here x_i and y_i are pixel coordinates, and $n(S)$ denotes the total number of pixels in segment S. The x coordinates of all points in S are involved in calculating X_S, not only those along the maximum width.

Figure 2 (right) illustrates the resulting segments of the *pool* image. The center of each segment is denoted by a '+'. We also create a *centers map*, which is a binary map of the same size as the original image. If a pixel is a center of a segment in the corresponding image, its value is 1 in the centers map; otherwise 0. Our *distinguishable points map* δ is the binary map that is the logical (inclusive) "or" of the binary centers map and binary corners map.

3 Experimental Methodology

Our attack methods are motivated by the following conjectures related to user choice. Regardless of the degree to which these conjectures are true, they have led us to significantly improved attack algorithms, which is our objective herein, rather than proving or disproving these conjectures per se.

Conjecture 1 *Users are likely to collectively prefer some areas of an image more than others, with the aggregate effect that a significant subset of users will choose passwords composed of click-points that are among the collectively preferred click-points.*

Conjecture 2 *Points that are collectively preferred can be identified by image processing tools (e.g., models of visual attention).*

Another factor that could influence the complexity (and thus a user's preference) of a password is the relationship between click-points that compose a password. People are better at recalling fewer pieces of visual information [8], motivating Conjecture 3.

Conjecture 3 *Since people find it easier to recall fewer pieces of information, a significant subset of users are likely to choose sets of points that can be grouped, e.g., by following one of a set of simple click-order patterns (such as left to right).*

Conjectures 1, 2, and 3 lead us to pursue attacks that use click-order patterns and image processing methods to create efficient, ordered attack sub-dictionaries. Some of our dictionaries use a *window clustering* algorithm, described in Section 3.1, to represent a set of click-points in a more compact way. We describe our attack alphabets in Section 3.2; one is based on image processing methods. To generate an attack dictionary based on heuristics (in our case these being click-order patterns), we introduce a general graph-based algorithm in Section 3.3. We describe specific click-order patterns and their specification in Section 3.4.

3.1 Window Clustering Algorithm

We assume that an attacker's goal is to guess the largest number of passwords with the fewest guesses. After creating a set of points for a guessing dictionary (which might be used in passwords in any ordering of five clicks), those within the same tolerance region could be redundant (effectively guessing the same point). We use the term *clustering* to mean normalizing a set of points to a single value. The intuition behind clustering is that given the system error tolerance, one point would be accepted as a correct entry for all others within its tolerance region.

We introduce a clustering algorithm (*window clustering*), based on setting a window of fixed size (not necessarily the same size as the tolerance region) over the largest number of points it can cover. We then replace those candidate points inside the window with the geometric *center* of the window. Thus, the center of the cluster is not necessarily one of the original input points (in contrast to a previous clustering algorithm [31]).

More precisely, window clustering is a greedy algorithm with a fixed window size. Starting with all candidate points, find the next position for the window that covers the maximum number of remaining points (ties are broken arbitrarily). Then store the center of the window to represent the points in the window, and erase the corresponding points. Continue this process until no candidate points remain. The candidate points we use are the points with value 1 in B_i of Section 3.2.1, and the window size is set to 19×19.

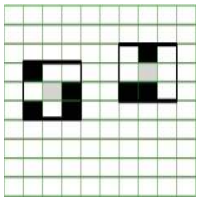


Figure 3: Window clustering.

Figure 3 shows an example set of candidate points with black squares, where each square represents a pixel. These 7 candidate points are covered with two 3×3 windows and will be represented by the centers of the two windows illustrated with grey squares.

3.2 Attack Alphabets

Motivated by Conjectures 1 and 2, we introduce a set of alphabets based on Itti's computational model of bottom-up visual attention that we call VA, introduced in Section 3.2.1. For comparison, we introduce an alphabet in Section 3.2.2 that covers the entire image.

3.2.1 Visual-Attention Based Alphabet (VA)

We used the Saliency Toolbox [34] implemented in Matlab. The weights of all feature maps used by the toolbox are set to one, which assigns orientation, intensity and colors the same level of importance. Normalization type is set as *LocalMax* and for all other settings, we use their default. With the resulting scan-path, we generate an ordered set of alphabets for use in a guessing attack as described in the paragraphs below. We generate a map of candidate click-points using the distinguishable points map δ as a bitmask to the resulting attracted regions of the image (i.e., using a logical "and"). We then refine this binary map of candidate points by using the window clustering algorithm of Section 3.1.

In this way, we generate a set of binary maps S', whose n elements represent the cumulative n steps in the scan-path (element i contains all steps from the first to the i^{th} step). More specifically, the i^{th} element in S' is the binary map B_i, which is generated from all scan-path steps up to step i: $S' = \{B_1, B_2, ..., B_n\}$ where $B_i = A_1 \vee A_2 \vee ... \vee A_i$ and each element A_j is a binary map representing the j^{th} step in the scan-path. In other words, $B_i = B_{i-1} \vee A_i$ and $B_1 = A_1$. Note that each element A_j contains many pixels, for example in Figure 1a, A_7 would include *all* highlighted pixels within the circle labeled 7. Next, we calculate $\{C_1, C_2, ..., C_n\}$ where $C_i = B_i \wedge \delta$, the intersection (logical "and") of each element B_i with δ, and run the window clustering algorithm on each C_i to produce D_i, the resulting set of cluster centers (which are pixel locations on the image). The final set of attack alphabets VA $= \{D_1, D_2, \ldots, D_n\}$ is ordered by the number of scan-path steps it includes.

The VA set of alphabets uses the scan-path to prioritize the dictionary entries; the first alphabet uses only the first scan-path element, the second alphabet uses both the first and second scan-path elements, etc. The order of each alphabet within VA is motivated by the intuition that users may choose their click-points based on points that are in the focus-of-attention scan-path – but not necessarily in the order of the scan-path. For example, users may choose click-points according to a click-order pattern.

We use VA to create a dictionary motivated by Conjectures 1 and 2, and call it $VA{-}H$, where heuristic H denotes a specific click-order pattern. The attack sub-dictionary Z_i, of 5-permutations of the elements of D_i that satisfy a heuristic H, is generated by the dictionary generation algorithm explained in Section 3.3. The final attack dictionary set $Z = \{Z_1, Z_2, \ldots, Z_n\}$ is ordered by the number of steps in the scan-path that are considered, e.g., all passwords from dictionary Z_2 are tried as candidate passwords only after those in Z_1 are exhausted.

3.2.2 Alpha Alphabet (α)

To examine the efficacy of attack dictionaries based on click-order pattern heuristics alone (i.e., without additional image processing methods as in VA), we also used the following alphabet α with click-order patterns to generate dictionaries. α is the set of pixels each of which is a window center, upon partitioning an image into T-regions by placing a grid of 19×19 windows (i.e., the same size as the T-region) over the image; see Figure 4. Note that the T-region used in creating α depends only on the system error tolerance, independent of the parameter τ used later in different relaxation modes.

3.3 Dictionary Generation Algorithm

Suppose an attacker plans to generate a dictionary consisting of the subset of the r-permutations (of a specific alphabet A) where the elements of the subset must also satisfy some predefined conditions (e.g., click-order patterns). One approach is to generate all possible r-permutations of A and then keep only those which fulfill the predefined conditions (heuristics). In this approach, assuming the

Figure 4: Each '+' represents a character in the alphabet (α).

heuristic check is constant-time, the time complexity of dictionary generation, Θ_A, is $O(\frac{n!}{(n-r)!})$, where $n = |A|$. Note that the dictionary cardinality, Θ_D, can be significantly smaller than Θ_A ($\Theta_D << \Theta_A$), depending on the predefined conditions.

Instead we present a graph-based algorithm for dictionary generation whose computational cost is equal to the number of dictionary entries ($\Theta_D = \Theta_A$). At a high-level, this efficiency is achieved by creating a graph that captures only those elements that fulfill the predefined heuristic, where all paths of length $r-1$ are output as passwords. For heuristics corresponding to a small portion of the password space, this is much more efficient than iterating through the entire password space. We first partition each condition (heuristic) into a number of sub-conditions (sub-heuristics). Define each sub-heuristic h in the form of a binary relation R_h on the set A, i.e., a subset of $A \times A$ [27]. Thus, $R_h \subseteq A \times A$. Given $(a, b) \in R_h$, both a and b belong to A and this ordered pair satisfies h. Represent each relation R_h by a matrix M_h with entries

$$M_h(i,j) = \begin{cases} 1 & if\ (a_i, a_j) \in R_h \\ 0 & if\ (a_i, a_j) \notin R_h \end{cases}$$

where $A = \{a_1, ..., a_n\}$. M_h can be represented by a digraph G_h with vertices corresponding to the indices i of a_i and edge $< i, j >$ from i to j if and only if $M_h(i, j) = 1$.

To generate the sub-dictionary D_h whose entries satisfy the sub-heuristic h and are also r-permutations of A, we find all paths of length $r-1$ in the graph G_h, using Algorithm 1. A path source can be any node of G_h. A path of length $r-1$, which is one of the r-permutations of A, is represented by $p =< i_1, i_2, ..., i_r >$.

Algorithm 1

```
GenerateDictionary (G_h, r)      % G_h is a digraph with vertices 1 to n.
D_h ← ∅
for i = 1 to n do
   paths ← FindPaths(i, G_h, r − 1)   % paths is a set of paths of length r − 1 (i.e., passwords).
   D_h ← D_h ∪ paths
end for
return D_h
```

$FindPaths(source, G, l)$ is a recursive function which finds all paths of length l from node *source* in digraph G; see Algorithm 2. It finds paths of length $l-1$ for all neighbors of *source* and then prepends *source* to those paths.[1] For the base case of a path of length one, each path includes

[1]While a recursive version of FindPaths is presented here for conceptual simplicity, an iterative version is used in practice for efficiency.

the source and one neighbor. Algorithm 1 generates a sub-dictionary for a sub-heuristic, such as a right-to-left and bottom-to-top (RL_BT) click-order pattern, which is a sub-heuristic of the DIAG pattern heuristic (see Section 3.4). The union of the sub-dictionaries is the final attack dictionary.

Algorithm 2

```
FindPaths(source, G, l)
S ← ∅                                   % Upon termination, S is a set of paths of length l.
for all nodes i ∈ Neighbors(source)     % Neighbors(i) = set of neighbors of node i.
  if l > 1 then
    paths ← FindPaths(i, G, l − 1)
    for all p ∈ paths
      p ← Prepend(source, p)            % Prepend(i, p) prepends node i to path p.
    end for
  else
    paths ← {< source, i >}
  end if
  S ← S ∪ paths
end for
return S
```

3.4 Click-order Patterns, Relaxation and Constraints

We examine three click-order patterns alone (DIAG, LINE, and Localized Omni-Direction), and with what we call *lazy* and *super-lazy* variations (for both DIAG and LINE) that relax the definition of the patterns. DIAG includes any sequence of 5 click-points that follow *both* a consistent vertical and horizontal direction (e.g., straight lines in any direction, most arcs, and step-patterns); LINE includes any sequence of 5 click-points that follow either a vertical or horizontal line (see Fig. 5). Localized Omni-Direction includes all sequences of 5 click-points in which each two consecutive points are constrained by only a predefined distance constraint (but not by direction).

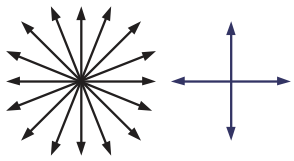


Figure 5: Directions included by DIAG (left) vs. LINE (right).

We break down the DIAG heuristic into four sub-heuristics LR_TB, LR_BT, RL_TB and RL_BT based on the mnemonics: LR (left-to-right), RL (right-to-left), TB (top-to-bottom), BT (bottom-to-top). More specifically, the defining relations for the DIAG heuristic are ($\tau \geq 0$ is a relaxation parameter, described further below):

$$R_{LR\_BT} = \{(P_i, P_j) \mid x_i \leq x_j + \tau \; AND \; y_i \geq y_j - \tau \; AND \; D_E(P_i, P_j) \leq T_d\}$$
$$R_{RL\_BT} = \{(P_i, P_j) \mid x_i \geq x_j - \tau \; AND \; y_i \geq y_j - \tau \; AND \; D_E(P_i, P_j) \leq T_d\}$$
$$R_{LR\_TB} = \{(P_i, P_j) \mid x_i \leq x_j + \tau \; AND \; y_i \leq y_j + \tau \; AND \; D_E(P_i, P_j) \leq T_d\}$$
$$R_{RL\_TB} = \{(P_i, P_j) \mid x_i \geq x_j - \tau \; AND \; y_i \leq y_j + \tau \; AND \; D_E(P_i, P_j) \leq T_d\}$$

where $P_i = (x_i, y_i)$ and $P_j = (x_j, y_j)$ are two independent (not necessarily successive) points, and the alphabet is $A = \{P_1, P_2, ..., P_n\}$. By convention, the positive y axis extends downward from the top-left pixel of the image. $D_E(P_i, P_j) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$ is the Euclidean distance between P_i and P_j.

Having observed some users choose successive click-points within a bounded distance from the previous click-point motivated us to define a distance constraint threshold T_d for consecutive points in LINE and DIAG ($T_d = \infty$ removes the constraint). When $T_d \neq \infty$, we subscript the dictionary with T_d.

As noted earlier, the DIAG heuristic has four sub-heuristics and thus separate graphs. For each, we generate a sub-dictionary using Algorithm 1. The DIAG dictionary is the union of four sets of passwords: $D_{DIAG} = D_{LR\_TB} \cup D_{LR\_BT} \cup D_{RL\_TB} \cup D_{RL\_BT}$. Similarly, $D_{LINE} = D_{LR} \cup D_{RL} \cup D_{BT} \cup D_{TB}$, where the four sub-dictionaries in LINE are generated using the defining relations:
$R_{LR} = \{(P_i, P_j) \mid (x_i \leq x_j + \tau)\ AND\ |y_i - y_j| \leq \tau\ AND\ D_E(P_i, P_j) \leq T_d\}$
$R_{RL} = \{(P_i, P_j) \mid (x_i \geq x_j - \tau)\ AND\ |y_i - y_j| \leq \tau\ AND\ D_E(P_i, P_j) \leq T_d\}$
$R_{BT} = \{(P_i, P_j) \mid (y_i \geq y_j - \tau)\ AND\ |x_i - x_j| \leq \tau\ AND\ D_E(P_i, P_j) \leq T_d\}$
$R_{TB} = \{(P_i, P_j) \mid (y_i \leq y_j + \tau)\ AND\ |x_i - x_j| \leq \tau\ AND\ D_E(P_i, P_j) \leq T_d\}$

For LINE and DIAG, the allowance $\tau \geq 0$ serves to relax the pattern, motivated as follows. Although the user might be inclined to select points along a line, the elements of that line may be influenced by which click-points the user otherwise prefers. In the absence of linear structures in the underlying image, the collection of four line segments, obtained by connecting successive pairs of user-selected click-points, may only approximate a line. Therefore, two variations on both DIAG and LINE allow "lazier" lines (see Figure 6): "lazy" uses $\tau = 19$ and "super-lazy" uses $\tau = 28$. Our default relaxation value is $\tau = 9$ pixels (equal to the system tolerance t). We denote a dictionary using a lazy or super-lazy τ with superscripts + and ++ respectively.

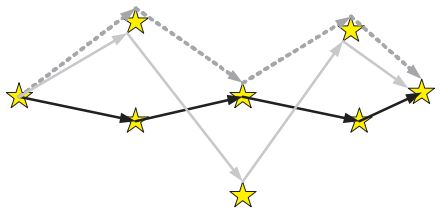


Figure 6: Laziness modes: normal (black), lazy (dashed grey), and super-lazy (solid grey).

A Localized Omni-Direction (LOD) dictionary is generated based on the single defining relation $R_{LOD} = \{(P_i, P_j) \mid |x_i - x_j| \leq \tau\ AND\ |y_i - y_j| \leq \tau\}$. Here τ serves as the distance constraint parameter on the two alphabet points P_i and P_j. Using Algorithm 1, we generate the corresponding dictionary denoted LOD^x, where $x = \tau$.

4 Experimental Results

We examine each click-order pattern (DIAG, LINE, and LOD) with each of the attack alphabets described in Section 3.2.[2] To evaluate each click-order pattern independent of visual attention models, we use the α alphabet; to evaluate the effect of using a visual attention model, we use the VA set of alphabets. To allow comparison, we tested these methods by trying to guess user

[2]This corrects preliminary results reported earlier [28] (some dictionary sizes are up to one bit larger, and some attack efficacies are slightly improved).

passwords from a previous PassPoints field study dataset [31, 5]. The study was 7-weeks or longer (depending on the user), involving 223 user accounts on a web-based implementation of PassPoints allowing access to course notes, assignment solutions, and tutorials. We focus on the field study rather than the related lab study for increased ecological validity of the passwords' long-term memorability. Participants were from two first year courses for computer science students, and a first year course for non-computer science students enrolled in a science degree. Participants used one of two background images, *pool* or *cars* (see Figure 7), preselected to be representative of highly detailed usable images at 451×331 pixels.

Figure 7: *cars* (originally from [14]). See Figure 1 for *pool*.

Passwords had 5 click-points, no two within $t = 9$ pixels of another (vertically and horizontally). Consistent with the previous study, we used only the final passwords exercised by each user (and recalled at least once). These 223 user accounts mapped to 189 distinct users (34 users were in two classes; all but one of them were assigned a different image for each account). Overall, 114 user accounts used *pool* and 109 used *cars* as a background image.

4.1 DIAG Results

We first examine the DIAG pattern using the α alphabet, with all laziness modes (see Table 1). The results for the dictionaries marked '+' and '++' are the same because the T-regions are non-overlapping, and the difference between $\tau = 19$ and 28 is 9, which is not greater than τ and thus does not include any more T-regions (the same is true for Table 2) .

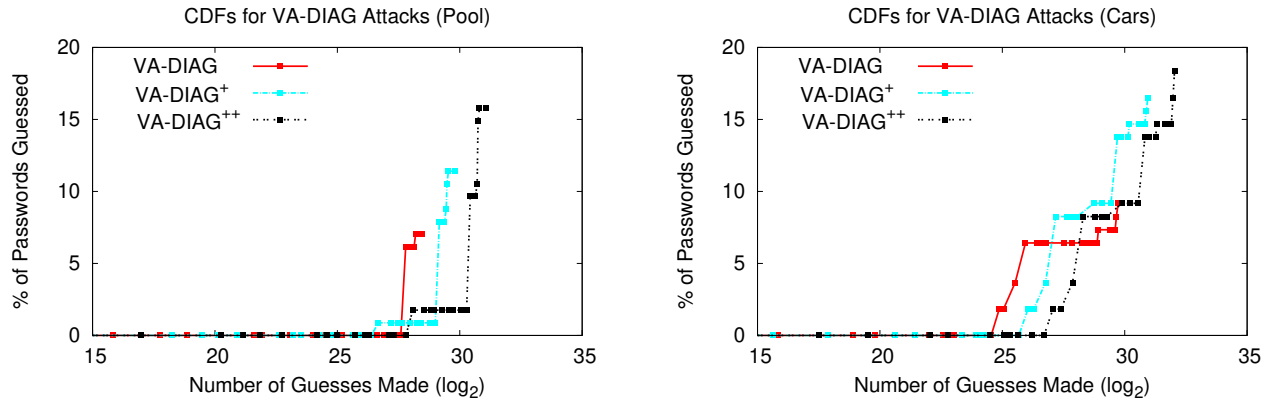
Table 1: Results for DIAG click-order heuristic.

Dictionary	Entries	% guessed	
		pool	*cars*
$\alpha DIAG$	$2^{33.02}$	21.1%	27.5%
$\alpha DIAG^{+}$, $\alpha DIAG^{++}$	$2^{35.26}$	48.2%	54.1%

Table 1 reports that the $\alpha DIAG^{+}$ dictionary guesses 48% of passwords for *pool* and 54% of passwords for *cars* with dictionaries of approximately 2^{35} entries; these results do not involve use of any visual attention modeling. Previous purely automated attacks [31] on the same password database, with a dictionary of 2^{35} entries, guessed 9.1% (*cars*) and 0.9% (*pool*).

We also combine VA with the DIAG click-order patterns at all laziness modes. The cumulative distribution function (CDF) of results (until each dictionary is exhausted) are provided in Figure

8. This creates what might be viewed as a more efficient dictionary: for $DIAG^{++}$ finding 15.8-18.3% of passwords using dictionaries of less than 2^{32} entries, i.e., finding nearly $\frac{1}{3}$ as many as the $\alpha DIAG^{++}$ dictionary, with almost 10 times fewer entries.



(a) VA-DIAG CDFs for *pool*.

(b) VA-DIAG CDFs for *cars*.

Figure 8: CDFs for DIAG with VA.

4.2 LINE Results

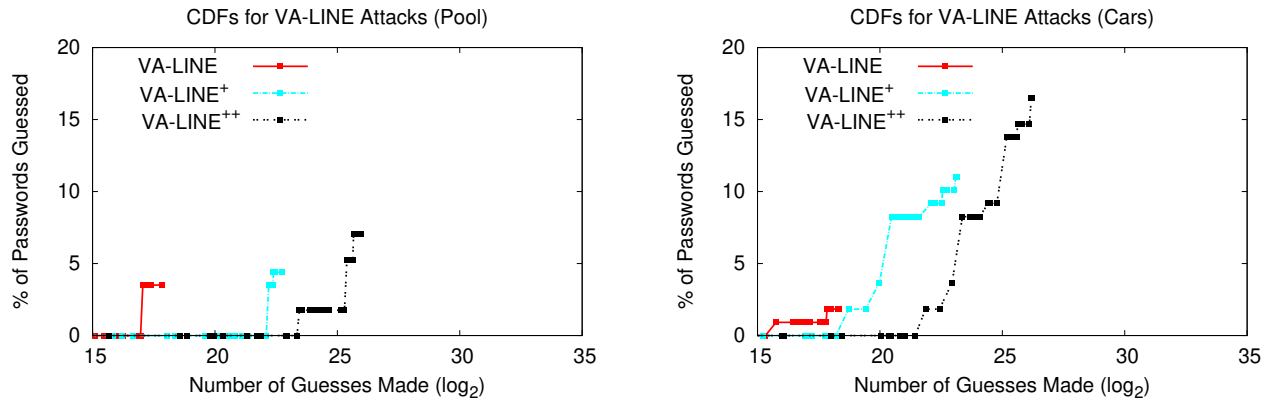
Table 2 provide results for the LINE pattern using the α alphabet. $\alpha LINE^{+}$ finds 23.7% of passwords for *pool*, and 52.3% for *cars* using a dictionary of 2^{29} entries. Using $\alpha LINE^{+}$ (vs. $\alpha DIAG^{+}$), the percentage found drops only minorly from 54% to 52% for *cars* and from 48% to 24% for pool (still about half as many), despite a dictionary about 75 times smaller. Thus for both images, $\alpha LINE^{+}$ is more efficient (in terms of efficacy relative to dictionary size) than any variation of DIAG. This might, in retrospect, be less surprising for *cars*, given the line structures in the orientation of the cars, than for *pool*.

Table 2: Results for LINE click-order heuristic.

Dictionary	Entries	% guessed	
		pool	*cars*
$\alpha LINE$	$2^{20.88}$	3.5%	22.0%
$\alpha LINE^{+}$, $\alpha LINE^{++}$	$2^{29.02}$	23.7%	52.3%

We also combine VA with the LINE patterns. The cumulative distribution function (until each dictionary is exhausted) is provided in Figure 9. As with the DIAG pattern, the VA alphabet creates a more efficient dictionary relative to its size. The $LINE^{++}$ variation guesses 7.0-16.5% of passwords using a dictionary of less than $2^{26.2}$ entries, finding $\frac{1}{4}$ to $\frac{1}{3}$ as many passwords as $\alpha LINE^{++}$ with a dictionary 7 times smaller.

For all click-order patterns, dictionaries composed from the VA alphabet work better on *cars* than *pool*. This is at least partially because corner and center detection does not work as well on *pool* as on *cars*.



(a) *pool*. (b) *cars*.

Figure 9: VA-LINE CDFs.

4.3 Localized Omni-Direction Results (LOD)

We tested the LOD pattern using alphabets VA and α, with distance constraints $T_d = 20$, 40, 60, 80, and 100 pixels.

First we generated dictionaries using α and tested on the above-mentioned dataset. Table 3 and Figure 10 show that the LOD pattern is comparable to $\alpha DIAG^+$ for *pool*. αLOD^{100} has 2^{35} entries and finds 47.4% of passwords on *pool*, whereas $\alpha DIAG^+$ (Section 4.1) found a comparable number of passwords on *pool* with essentially the same sized dictionary. For this dataset, $\alpha LINE^+$ (Section 4.2) remains the best attack strategy for *cars*.

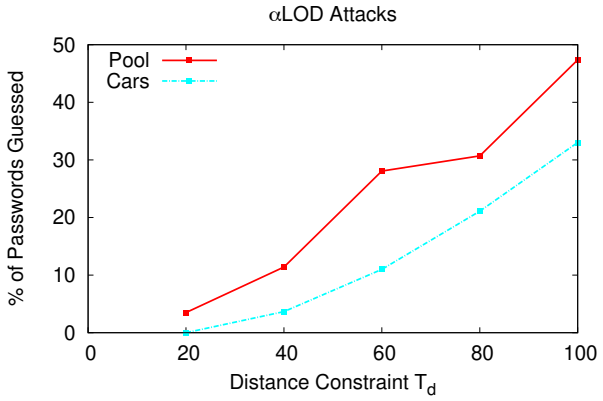


Figure 10: αLOD attacks: passwords guessed as a function of distance constraint T_d.

Next we generated LOD^{100} dictionaries using VA and tested them on the same dataset. The results (see Figure 11) showed that VA-LOD^{100} found $\frac{1}{3}$ as many as αLOD^{100}, with a dictionary 8 times smaller. The VA-LOD^{100} dictionary contains 2^{32} entries and found 14.0-15.6% of passwords.

Table 3: Dictionary Sizes for αLOD Attacks.

Dictionary	αLOD^{20}	αLOD^{40}	αLOD^{60}	αLOD^{80}	αLOD^{100}
Entries	$2^{19.61}$	$2^{26.28}$	$2^{30.19}$	$2^{32.94}$	$2^{35.02}$

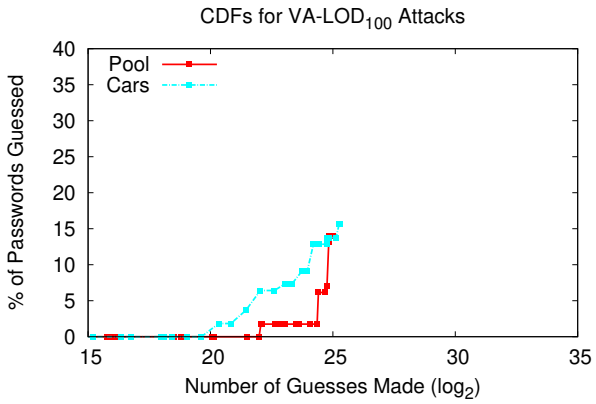


Figure 11: CDFs of $VA\text{-}LOD^{100}$ for both *pool* and *cars*.

4.4 Effect of Distance Constraint

Here we explore the impact of adding distance constraints to the DIAG and LINE pattern attacks. The LOD pattern, defined only by distance constraint, was described separately in Section 4.3.

First we examine DIAG and LINE using the α alphabet and distance constraints $T_d = 25$, 50, 75, 100, 125, and 150. The efficacy as a function of T_d, and for all laziness modes, is provided in Figure 12. The dictionary sizes for each distance constraint and pattern is provided in Table 4.

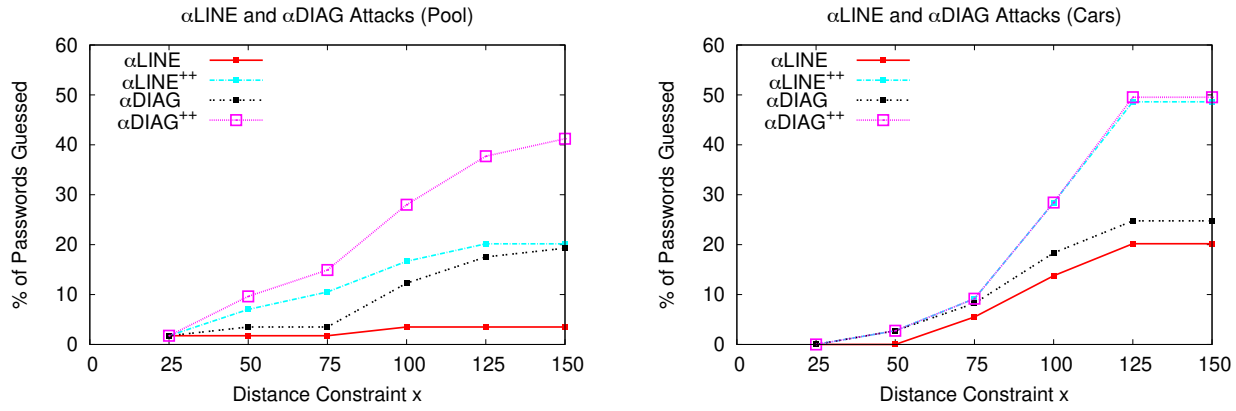


Figure 12: Attack efficacy as a function of distance constraint $x = T_d$ (for various dictionaries).

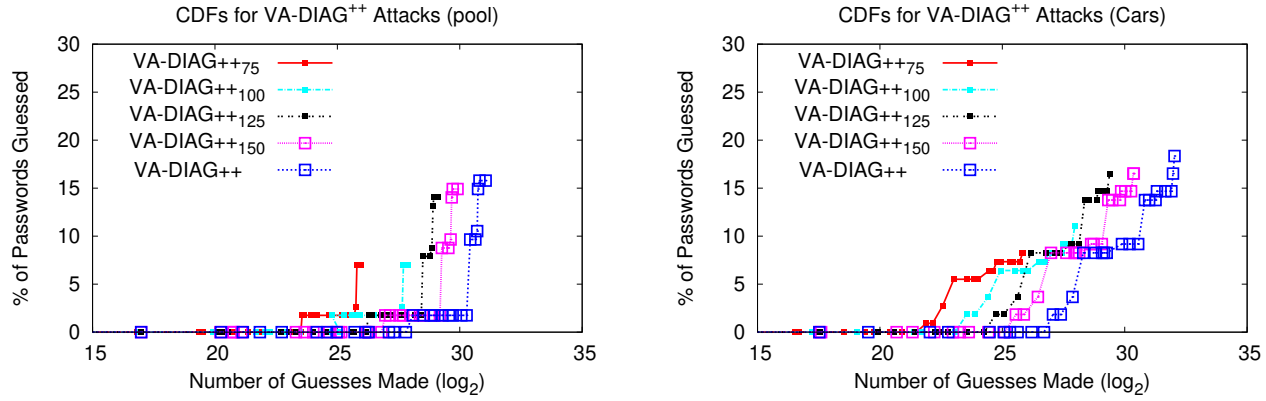
In Figure 12, of course, the less restrictive the distance constraint is, the more passwords the resulting dictionary finds; however, at each distance constraint studied, the dictionary size is reduced substantially while maintaining a (proportionally) large percent of the guessing accuracy. For instance, consider $T_d = 150$ with the $\alpha DIAG^{++}$ dictionary, i.e., $\alpha DIAG^{++}_{150}$: for *cars*, 50% of passwords are guessed (compared to 54% when no distance constraint is used) and for *pool*,

Table 4: Results for click-order heuristics when combined with distance constraints.

	Distance constraint T_d					
Dictionary	25	50	75	100	125	150
$\alpha LINE_x$	$2^{10.44}$	$2^{14.25}$	$2^{16.42}$	$2^{18.78}$	$2^{19.74}$	$2^{19.97}$
$\alpha LINE_x^{++}$	$2^{16.58}$	$2^{23.55}$	$2^{25.12}$	$2^{26.90}$	$2^{27.50}$	$2^{27.94}$
$\alpha DIAG_x$	$2^{14.44}$	$2^{21.34}$	$2^{25.00}$	$2^{28.20}$	$2^{29.86}$	$2^{31.00}$
$\alpha DIAG_x^{++}$	$2^{15.58}$	$2^{24.65}$	$2^{27.83}$	$2^{30.66}$	$2^{32.06}$	$2^{33.07}$

41% (compared to 48% when no distance constraint is used, i.e., $T_d = \infty$). Thus a more than 4 times smaller $\alpha DIAG_{150}^{++}$ dictionary still guesses 92-85% of the passwords that $\alpha DIAG^{++}$ does. This trend holds for each distance constraint we used, for each dictionary (LINE and DIAG with different laziness modes). This suggests that distance constraint (from smallest to largest) is an effective method for prioritizing an attack dictionary. This attack is surprisingly effective even with small distance constraints (e.g., $T_d = 50$); in particular $\alpha DIAG_{50}^{++}$ guesses 3% and 10% of passwords on *cars* and *pool* respectively, with a dictionary of fewer than 2^{25} entries.

We next consider distance constraints with the VA-$DIAG^{++}$ dictionary. The CDFs in Figure 13 show that these same distance constraints are also effective for dictionaries generated with the VA alphabet: VA-DIAG$_{125}^{++}$ guesses nearly as many passwords as VA-DIAG$_{\infty}^{++}$ (i.e., with no distance constraint), despite dictionaries 4 to 8 times smaller for both images. Also, VA-based dictionaries are more effective than those generated with α: VA-DIAG$_{125}^{++}$ is 8 times smaller than $\alpha DIAG_{125}^{++}$ and yet still guesses more than $\frac{1}{3}$ as many passwords.



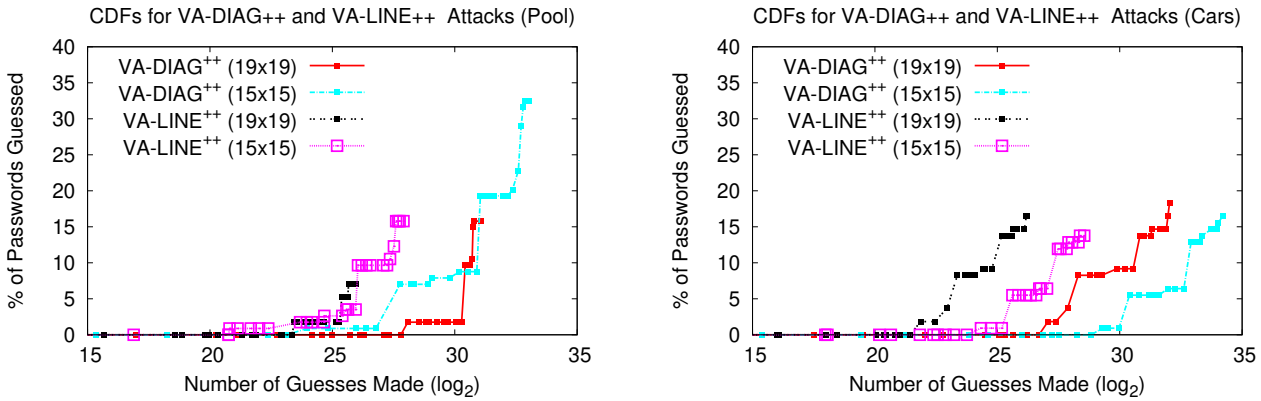
(a) *pool*. (b) *cars*.

Figure 13: VA-DIAG^{++} CDFs.

4.5 Further Optimizations: Window Clustering Size

Our previous experiments herein used a default window size of 19×19 pixels in the window clustering algorithm. We hypothesized that smaller window sizes would introduce less error in VA (since clustering introduces some inherent information loss), increasing its effectiveness. Of course, smaller window sizes imply a larger alphabet. We re-ran the experiments for VA-$DIAG^{++}$ and VA-$LINE^{++}$ using a window size of 15×15 to see how it impacts both guessing effectiveness and dictionary size (number of guesses). The results are plotted in comparison to the default window

size in Figure 14. For *pool*, the smaller window size increases the overall number of passwords found: for $VA\text{-}DIAG^{++}$, 32% vs. 16% with the larger window size. The dictionary size increases by a factor of 4. This suggests that the dictionary is less efficient relative to its size, but reducing the window size provides a viable attack continuation strategy. The results shown in Figure 14 suggest that VA (and thus Itti's model of visual attention) was more accurate for *pool* than *cars*, and in general we expect variation across images.



(a) *pool* (b) *cars*

Figure 14: CDFs for $VA{-}DIAG^{++}$ and $VA{-}LINE^{++}$ using different window clustering sizes.

We expect that this is related to image processing measures working better for *cars*. Prior to applying the window clustering algorithm, our VA attacks use corner and centroid detection which both introduce some image-dependent error (E_c), which is greater for images not containing rectangular shapes (e.g., *pool*). The window clustering algorithm, through reducing the number of points, loses information and thus creates some error (E_w) dependent on the window clustering size. The overall error of our image processing methods (E_t) depends on both E_w and E_c. Since *cars* has more rectangular shapes (cars, license plates, etc.), we expect $E_c(cars) < E_c(pool)$. Because $E_w(cars) = E_w(pool)$, $E_t(cars) < E_t(pool)$. E_t should be less than the T-region size (here, $t = 19$) to successfully guess a click-point. Reducing the clustering size decreases E_w, thus decreasing E_t. For images where E_c was large, reducing E_w may compensate enough to bring the overall error E_t below the overall error tolerance of the system (in our case $t = 19$), which increases the success rate.

5 Related Work

Many graphical passwords schemes have been proposed to date, and several surveys are available [29, 21, 26]. Here we concentrate on click-based graphical password schemes, wherein a user clicks on a set of points on one or more presented background images, and work related to guessing attacks on graphical passwords.

In *V-go* [25], users click on a sequence of predefined objects in the picture. In Blonder's proposal [1], users click on a set of predefined tap regions. Jansen et al. [17] propose a variation, which requires users to click an ordered sequence of visible squares imposed on a background image; the squares are intended to help users repeat click-points in subsequent logins.

PassPoints [35, 37, 36] allows users to click a sequence of five points anywhere on an image while allowing a degree of error tolerance; studies suggest promising usability [36, 35, 37, 5]. A

related commercial system designed for the Pocket PC, called *VisKey*, allows the user to choose the number of click-points and to set the error tolerance.

The security of click-based graphical passwords has been examined [10, 31, 28, 32]; for security analyses of other types of graphical schemes see also Davis et al. [9] and van Oorschot et al. [33]. One way that an attacker could predict hot-spots is by using image processing tools to locate areas of interest. Dirik et al. [10] use an image processing tool for guessing PassPoints passwords to guess single-session user passwords for two images, one being a particularly simple image. For the other image, their method guessed 8% of passwords using an attack dictionary with 2^{32} entries where the full space was 2^{40} entries. In other work, Thorpe et al. [31] examine an automated method (based on a variation of Itti et al.'s [16] model of visual attention), guessing 9.1% and 0.9% of passwords on two images, using an attack dictionary with 2^{35} entries compared to a full pasword space of 2^{43} passwords. The method of Thorpe et al. [31] focused only on a variation of stage 1 (recall Section 2.1), ordering an attack dictionary based on the raw values of the resulting saliency map, whereas the present paper uses the entire model including stage 2. In a preliminary version of the present work, Salehi-Abari et al. [28] guess 8-15% of passwords for two representative images using dictionaries of less than $2^{24.6}$ entries, and about 16% of passwords on each of these images using dictionaries of less than $2^{31.4}$ entries, where the full space is 2^{43}.

Basic click-order patterns were first introduced and evaluated in combination with human-seeded attacks [31, 30]; the only pattern in common with the present work is regular DIAG (i.e., without any "laziness" relaxation). Chiasson et al. [3] analyze a set of patterns for three click-based graphical password schemes: PassPoints [35, 37, 36], and two variants named Cued Click-Points (CCP) [6] and Persuasive Cued Click-Points (PCCP) [2]. In CCP and PCCP, a user clicks on a single point on each of five images, where each image (except the first image) is dependent on the previous click-point. They show that the design of the interface impacts whether users select click-points in some predictable patterns, and implied that such patterns in user choice might reduce the effective password space. The present paper mathematically models click-order patterns and uses them to mount purely automated attacks, demonstrating and experimentally quantifying the degree to which certain patterns can be used to efficiently search the password space.

Thorpe et al. [31, 32] introduce human-seeded attacks and demonstrate their efficacy against Passpoints-style graphical passwords. Human-computed data sets (harvesting click-points from a small set of users) were used in two human-seeded attacks against passwords from a field study on two different images: one based on a first-order Markov model [32], another based on an independent probability model [31]. Using their human-computed data sets (harvested from a single-session lab study), a dictionary based on independent probabilities contained $2^{31.1} - 2^{33.4}$ entries and found 20-36% of field study passwords, and a dictionary based on the first-order Markov model found 4-10% of field study passwords within 100 guesses. These attacks require the attacker to collect sufficient click-points for each image, and are image dependent, thus requiring per-image costs for systems with multiple images.

6 Discussion and Concluding Remarks

We provide what is to our knowledge the best automated attack against PassPoints-style graphical passwords to date. Combining click-order patterns with our laziest relaxation rule yielded highly effective dictionaries. We were able to further reduce the dictionary size while retaining some accuracy using Itti et al.'s [16] computational model of bottom-up visual attention. We also explored distance constraints to determine how effective they were at guessing passwords both alone (captured by LOD), and combined with other click order patterns (DIAG and LINE). We found that

although distance constraints alone do not provide a more effective dictionary than other click-order patterns, they appear to provide a reasonable heuristic for prioritizing a dictionary (from shorter to longer distance constraints).

Our results are a significant improvement on previous work for purely automated guessing PassPoints-style graphical passwords. The dictionary that guessed the greatest percentage of passwords is $DIAG$ with the laziest relaxation rule ($\alpha DIAG^{++}$), finding over 48% of user passwords in our dataset for each of two images. Previous approaches found only 1-9% [31] for the same images and user study password database, and 8% [10] for a single (comparable) image. Another notable result is that the LINE dictionary, with the laziest relaxation rule ($\alpha LINE^{++}$) is less than 2% of the size of $\alpha DIAG^{++}$ and still guesses 24-52% of passwords.

Although the lazy click-order dictionary sizes are not as small as previous dictionaries used for human-seeded attacks [32], when we combine them with a model of visual attention (recall Section 2.1), our results are comparable to the human-seeded (independent probability) results for *cars*. Further reduction in the dictionary size can be achieved by using a distance constraint; for example, constraining the distance to 125 pixels in DIAG dictionaries (with the visual attention model) reduces the number of dictionary entries by 75-87%, but the efficacy only drops by 1-2%.

In all of our dictionaries using click-order patterns, a larger percentage of passwords were guessed using the α alphabet (*not* based on a visual attention model). However, using the VA alphabet (based on a visual attention model) provided a more efficient dictionary in that the reduction of the number of entries in the dictionary was more than the reduction in guessing accuracy; for example VA$-DIAG^{++}$ guessed 33% as many passwords as $\alpha DIAG^{++}$ using a dictionary with 10% of the entries. The relative "efficiency" of the VA dictionaries, however, cannot be extended in their present form to guess a larger percentage of passwords, because the full dictionaries are exhausted.

These results suggest that automated attacks provide an effective alternative to a human-seeded attack against PassPoints-style graphical passwords. Furthermore, they allow continuation of an attack using click-order patterns (without any prioritization through visual attention models or other means), guessing more passwords overall than human-seeded methods. Finally, purely automated attacks are arguably much easier for an attacker to launch (removing the requirement of humans to index the images), especially if large image datasets are used. We emphasize that a number of our attack dictionaries (those composed from the α alphabet), do not rely on visual attention techniques or any image-specific pre-computation, implying that the actual dictionaries are the same for all images, though the attack results (i.e., their effectiveness) are image-dependent and of course depend also on the actual passwords chosen by any users in question. We do not expect that these attacks can be effectively applied to multi-image click-based schemes (e.g., PCCP [2]).

Our findings might be used to help background image selection, although precise details of an exact method remain unclear. As one possibility, corners and centroids of images might be extracted, and used to build a click-order heuristic graph (as in our dictionary generation algorithm); the images that generate a larger resulting dictionary might indicate a more attack-resistant image. Another possibility might be to measure the amount of structure an image has, assuming that such image structure would encourage click-order patterns.

Our overall results indicate that Itti's model of visual attention with the parameters and implementation we used, when all permutations of points in the scan-path are considered, models a meaningful percentage (from an attacker viewpoint) of user passwords. If we assume that Itti's model using the default settings is an accurate representation of bottom-up visual attention, our results are consistent with bottom-up visual attention being one part of a broader criteria for selecting click-points. Alternately, these click-points might be chosen according to some other phenomenon that happens to have a non-trivial intersection with this model of bottom-up visual attention.

These results raise interesting questions regarding how visual attention relates to user choice in graphical passwords. For example, might some users choose the first point according to bottom-up visual attention, and then the rest in a top-down manner such that they are somehow similar to the first? Alternately, might the entire process be top-down, based on whether the user can find five objects that are similar in some way? Such a top-down theory would be substantially more difficult to model an attack on, but if possible to implement, its results might offer interesting insight.

Finally, our attacks could be used to help inform more secure design choices in implementing PassPoints-style graphical passwords. Proactive checking rules for PassPoints-style graphical passwords might be created based on the click-order pattern attacks herein; for example, disallowing LINE or DIAG patterns (for all laziness modes), and disallowing passwords where too few click-points are further than 150 pixels away from the previous click-point. Of course, any such proactive checking rules would need to be tested to ensure that the usability impact is acceptable and that security is not impacted in other unexpected ways.

Acknowledgments

The first author is a Canada Research Chair in Internet Authentication and Computer Security, and acknowledges NSERC funding of this chair, a Discovery Grant, and NSERC ISSNet.

References

[1] G. Blonder. Graphical Passwords. United States Patent 5559961, 1996.

[2] S. Chiasson, A. Forget, R. Biddle, and P.C. van Oorschot. Influencing Users Towards Better Passwords: Persuasive Cued Click-Points. In *Proceedings of HCI, British Computer Society*, 2008.

[3] S. Chiasson, A. Forget, R. Biddle, and P.C. van Oorschot. User Interface Design Affects Security: Patterns in Click-Based Graphical Passwords. *International Journal of Information Security*, 8(5), 2009.

[4] S. Chiasson, A. Forget, E. Stobert, P.C. van Oorschot, and R. Biddle. Multiple Password Interference in Text Passwords and Click-Based Graphical Passwords. In *16th ACM Conference on Computer and Communications Security (CCS)*, 2009.

[5] S. Chiasson, P.C. van Oorschot, and R. Biddle. A Second Look at the Usability of Click-Based Graphical Passwords. In *Proceedings of the 3rd Symposium on Usable Privacy and Security (SOUPS)*, 2007.

[6] S. Chiasson, P.C. van Oorschot, and R. Biddle. Graphical Password Authentication Using Cued Click Points. In *Proceedings of the European Symposium on Research in Computer Security (ESORICS)*, 2007.

[7] D. Comaniciu and P. Meer. Mean Shift: A Robust Approach Toward Feature Space Analysis. *IEEE Trans. PAMI*, 24(5):603–619, 2002.

[8] N. Cowan. The Magical Number 4 in Short-Term Memory: A Reconsideration of Mental Storage Capacity. *Behavioral and Brain Sciences*, 24:87–185, 2000.

[9] D. Davis, F. Monrose, and M.K. Reiter. On User Choice in Graphical Password Schemes. In *Proceedings of the 13th USENIX Security Symposium*, 2004.

[10] A. Dirik, N. Memon, and J.-C. Birget. Modeling User Choice in the PassPoints Graphical Password Scheme. In *3rd Symposium on Usable Privacy and Security (SOUPS)*, 2007.

[11] K. M. Everitt, T. Bragin, J. Fogarty, and T. Kohno. A Comprehensive Study of Frequency, Interference, and Training of Multiple Graphical Passwords. In *CHI '09: Proceedings of the 27th International Conference on Human Factors in Computing Systems*, 2009.

[12] Rafael C. Gonzalez and Richard E. Woods. *Digital Image Processing (3rd Edition)*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 2006.

[13] C.G. Harris and M.J. Stephens. A Combined Corner and Edge Detector. In *Proceedings of the Fourth Alvey Vision Conference*, pages 147–151, 1988.

[14] Ian Britton (Reproduced by Permission of). Image Ref: 21-35-3. `http://www.freefoto.com`.

[15] L. Itti and C. Koch. Computational Modeling of Visual Attention. *Nature Reviews Neuroscience*, 2(3):194–203, Mar 2001.

[16] L. Itti, C. Koch, and E. Niebur. A Model of Saliency-Based Visual Attention for Rapid Scene Analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(11):1254–1259, 1998.

[17] W. Jansen, S. Gavrilla, V. Korolev, R. Ayers, and Swanstrom R. Picture password: A visual login technique for mobile devices. NIST Report - NISTIR7030, 2003.

[18] P. D. Kovesi. MATLAB and Octave Functions for Computer Vision and Image Processing. School of Computer Science & Software Engineering, The University of Western Australia. `http://www.csse.uwa.edu.au/~pk/research/matlabfns/`.

[19] S. Madigan. Picture Memory. In John C. Yuille, editor, *Imagery, Memory and Cognition*, pages 65–89. Lawrence Erlbaum Associates Inc., N.J., U.S.A., 1983.

[20] Wendy Moncur and Grégory Leplâtre. Pictures at the ATM: Exploring the Usability of Multiple Graphical Passwords. In *CHI '07: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 887–894, New York, NY, USA, 2007. ACM Press.

[21] F. Monrose and M. K. Reiter. Graphical Passwords. In L. Cranor and S. Garfinkel, editors, *Security and Usability*, chapter 9, pages 147–164. O'Reilly, 2005.

[22] Vidhya Navalpakkam and Laurent Itti. Modeling the Influence of Task on Attention. *Vison Research*, 45:205–231, 2005.

[23] A. Oliva, A. Torralba, M. Castelhano, and J. Henderson. Top Down Control of Visual Attention in Object Detection. *Journal of Vision*, 3(9):253–256, 2003.

[24] N. Ouerhani, R. von Wartburg, H. Hugli, and R. Muri. Empirical Validation of the Saliency-based Model of Visual Attention. *Electronic Letters on Computer Vision and Image Analysis*, 3(1):13–24, 2004.

[25] Passlogix. `http://www.passlogix.com`, site accessed Feb. 2, 2007.

[26] K. Renaud. Guidelines for designing graphical authentication mechanism interfaces . *International Journal of Information and Computer Security*, 3(1):60–85, 2009.

[27] Kenneth H. Rosen. *Discrete Mathematics and Its Applications*. McGraw-Hill, 2002.

[28] A. Salehi-Abari, J. Thorpe, and P.C. van Oorschot. On Purely Automated Attacks and Click-Based Graphical Passwords. In *Proceedings of the 24th Annual Computer Security Applications Conference (ACSAC)*, 2008.

[29] Xiaoyuan Suo, Ying Zhu, and G. Scott Owen. Graphical Passwords: A Survey. In *Proceedings of the 21st Annual Computer Security Applications Conference (ACSAC)*, 2005.

[30] J. Thorpe. *On the Predictability and Security of User Choice in Passwords*. PhD thesis, Carleton University, 2008.

[31] J. Thorpe and P.C. van Oorschot. Human-Seeded Attacks and Exploiting Hot Spots in Graphical Passwords. In *Proceedings of the 16th USENIX Security Symposium*, 2007.

[32] P.C. van Oorschot and J. Thorpe. On Predicting and Exploiting Hot-Spots in Click-Based Graphical Passwords, 2008. School of Computer Science, Carleton University, Technical Report TR-08-21 (Journal version currently under review).

[33] P.C. van Oorschot and J. Thorpe. On Predictive Models and User-Drawn Graphical Passwords. *ACM Transactions on Information and System Security*, 10(4):1–33, January 2008.

[34] Dirk Walther and Christof Koch. 2006 Special Issue: Modeling Attention to Salient Proto-objects. *Neural Network*, 19(9):1395–1407, 2006.

[35] S. Wiedenbeck, J. Waters, J.C. Birget, A. Brodskiy, and N. Memon. Authentication Using Graphical Passwords: Basic Results. In *Human-Computer Interaction International (HCII)*, 2005.

[36] S. Wiedenbeck, J. Waters, J.C. Birget, A. Brodskiy, and N. Memon. Authentication Using Graphical Passwords: Effects of Tolerance and Image Choice. In *Proceedings of the 1st Symposium on Usable Privacy and Security (SOUPS)*, 2005.

[37] S. Wiedenbeck, J. Waters, J.C. Birget, A. Brodskiy, and N. Memon. PassPoints: Design and Longitudinal Evaluation of a Graphical Password System. *International Journal of Human-Computer Studies (Special Issue on HCI Research in Privacy and Security)*, 63:102–127, 2005.

[38] J.M. Wolfe. Guided Search 2.0: A Revised Model of Visual Search. *Psychonomic Bulletin and Review*, 1(2):202–238, 1994.