Optimal Discretization for High-Entropy Graphical Passwords

Kemal Bicakci

TOBB University of Economics and Technology, Ankara, Turkey

Abstract. In click-based graphical password schemes that allow arbitrary click locations on image, a click should be verified as correct if it is close within a predefined distance to the originally chosen location. This condition should hold even when for security reasons the password hash is stored in the system, not the password itself. To solve this problem, a robust discretization method has been proposed [4], recently. In this paper, we show that previous work on discretization does not give optimal results with respect to the entropy of the graphical passwords and propose a new discretization method to increase the password space. To improve the security further, we also present several methods that use multiple hash computations for password verification.

Keywords: authentication, password security, graphical passwords, discretization.

1 Introduction

Graphical passwords offer a viable alternative to alphanumeric passwords and they have the potential to eliminate well-known problems such as low entropy attributed to classical password based authentication. Among the numerous graphical password schemes proposed so far, one promising approach is the click-based schemes e.g. Passpoints [1], Cued Click Points [2]. The difference of Passpoints and CCP from the earliest example of click-based scheme due to Blonder [3] is that the image is not partitioned into pre-designed regions and users are free to select any point(s) on the image. By allowing any arbitrary click locations, the goal is to improve the security by increasing the password space and achieving higher password entropies.

Allowing any arbitrary click location(s) brings a problem, however. It is not reasonable to expect users while they are being authenticated to reselect exactly the same points selected during registration. The natural solution is to set a tolerance value *r* so that a click point is accepted if it is checked and verified to be in the *r*-neighborhood of the original location registered in the system.

This simple solution works well if the password is stored in plaintext on the server side. However, in order to prevent someone to access the password file on the server and steal all the passwords, it is generally preferred to compute and store the hash value of the password. Here, for security reasons, the hash function used is designed in a way such that only one bit change in the input gives a totally different output value. For this reason, it is impossible for the server, now knowing only the hash value, to decide whether the password entered is in the acceptable tolerance region or not.

This problem has been identified and a solution is proposed recently by Birget et. al. [4]. In their solution, before the hash function is applied, graphical passwords are discretized using grid maps so that all possible password entries in tolerable regions are represented by single values. As also noted in [4], one important requirement in the design of a good discretization is to minimize the loss of precision. The password security roughly depends on the size of the password space and if the discretization is not optimized, it would lead to a smaller password space and less secure graphical passwords. In this paper, we propose several new discretization methods for graphical passwords to improve graphical password security by increasing the password space.

2 Previous Work

The image to be clicked to form the graphical password is represented by a function *f: [0,a] x [0,b]→[0,1].* The domain of the (two-dimensional) image is *R = [0,a] x [0,b]* and we can simply assume *[0,a]* and *[0,b]* are intervals in the integers *Z*. Discretization (quantization) of *R* can be simply done by choosing a positive number *q* (called the quantum) and an offset *(φ,ψ)* where $|\varphi|$, $|\psi| < q$ and superimpose a square grid on the rectangle. The grid has $\lfloor a/q \rfloor + 1$ vertical lines and $\lfloor b/q \rfloor + 1$ horizontal lines. Figure 1 shows three such grids on the same image *R*.

This discretization can be described by a grid map which tells us which points of the rectangle *R* are mapped to which grid vertices.

$$d : (x, y) \in [0, a]\, x\, [0, b] \quad \mapsto \quad \left(\left\lfloor \frac{x-\varphi}{q} \right\rfloor, \left\lfloor \frac{y-\psi}{q} \right\rfloor \right).$$

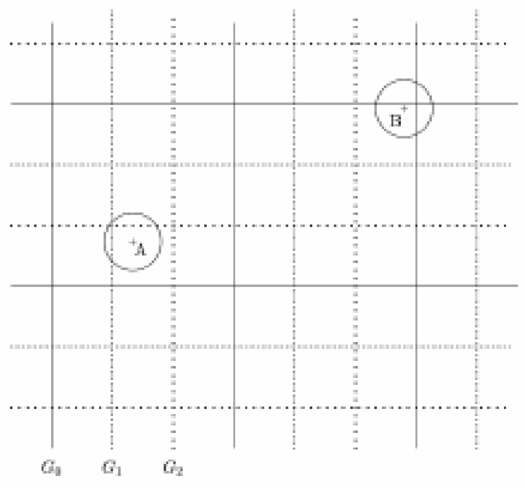

Fig. 1. Discretization in [4]: The three grids G_0, G_1, and G_2 (A is safe in G_0, B is safe in G_1, and G_2).

This simply way of "single" grid discretization is not a satisfactory solution for graphical passwords because of the "edge problem". Suppose that in Figure 1, only the grid G_0 is used. No matter how large the grids are (how big q is), we can always find a point whose tolerance region overlaps the grid. For instance, the tolerance region of point B overlaps grid G_0 in Figure 1. This is not acceptable in Birget et al.'s approach because it may lead to unintended changes in the output.

In [4], it was shown that in a two-dimensional image, using three grids G_0, G_1, and G_2 having the same quantum $q=6r$ and staggered with offset values *(-2rk, -2rk) for k=0, 1, 2*, every point is r-safe in at least one of these three grids. This is depicted in Figure 1. Here, r-safe means the tolerance region does not overlap and thus no unintended change is seen in the output. For robust discretization, first of all safe grid choice map is done as follows:

$$\gamma : (x,y) \in [0,a]\, x\, [0,b] \quad \mapsto \quad k \in 0\,,1,\,2$$

The robust grid map for a two-dimensional image that gives r-safety is then defined by

$$g : (x,y,k) \in [0,a]\, x\, [0,b]\, x\, [0,2] \quad \mapsto (\left(\left\lfloor \frac{x-2rk}{6r} \right\rfloor, \left\lfloor \frac{y-2rk}{6r} \right\rfloor\right)).$$

where k is the grid identifier which is stored in cleartext. So if there are c click points, the system stores

$$(k_1, k_2, k_3, \cdots, k_c) \quad and \quad HASH(g(x_1, y_1, k_1) \,||\, g\ (x_2, y_2, k_2) \,||\, g(x_3, y_3, k_3) \,||\, \cdots \,||\, g(x_c, y_c, k_c))$$

Upon entry of the password by the user, x_i and y_i values are extracted, the system computes the hash value and compares it with the already stored one. If the hashes do not match, the authentication fails.

Size of the Password Space: Using robust discretization, in a graphical password scheme parameterized by tolerance value r and the quantum value $q = 6r$, if number of click points is c and the image size is a x b, the number of possible passwords is

$$P = (\lfloor a/6r \rfloor \lfloor b/6r \rfloor)^c$$

3 Our Solution

Remark: In the rest of the discussion, we concentrate on the single dimension, the value $X \in [0, a]$. This gives us the flexibility of selecting optimum discretization parameters (e.g. offset values) per single dimension which results in improved overall security for the graphical password. A graphical password with c clicks on a two-dimensional image can be represented by the concatenation of $2c$ one-dimensional values. We still need to perform only one hash computation to store the hash value in the system as

$$(HASH(d(x_1) \,||\, d(y_1) \,||\, d(x_2) \,||\, d(y_2) \,||\, \cdots \,||\, d(x_c) \,||\, d(y_c))$$

where d is the discretization method used.

Our discretization methods fall into two types. As the first type, we allow nonzero offset values. We look at the case of offset values set to zero as the second type. We will present each type in the following two subsections, respectively.

3.1 Discretization Using Offsetting

Now, as a generalization and improvement of Birget et al.'s robust discretization method, we look at discretization methods that use offsetting where offset values are stored in cleartext together with the hash value of the password. The previous work by Birget et. al. can be considered as one such example because grid identifiers serve the purpose of offset values. As we have already mentioned, our critical observation is that since the grid map should avoid the edge problem for both of the two dimensions at the same time, previous work could not optimize the discretization result. We will show that by independently selecting offset values for each one-dimensional value in the graphical password, we can achieve much better results.

Terminology: We use small letter *x* as the value entered to the system in the registration and *X* is the value entered for authentication. By rule, *X* should be accepted if $x - r \le X < x + r$ for tolerance value *r*.

Remark: For ease of notation, we slightly change the definition of *r*-safety (only in this subsection). Previously *r*-safety for a value *x* is defined as *X* values satisfying $x\text{-}r \le X \le x\text{+}r$. We change the inequality as $x\text{-}r \le X < x\text{+}r$.

Theorem 1: In one-dimensional discretization $d : x \in [0,a] \mapsto \left(\left\lfloor \frac{x-\varphi}{q} \right\rfloor\right)$. parameterized by tolerance value *r*, quantum *q* and offset value φ, to obtain zero discretization error, *q* value should be set at least to *2r*.

Proof: Because the inequality $x\text{-}r \le X < x\text{+}r$ holds, *X* can take *2r* distinct values. The discretization $d : x \in [0,a] \mapsto \left(\left\lfloor \frac{x-\varphi}{q} \right\rfloor\right)$ gives the same output for at most *q* distinct values therefore *q* should be at least *2r*. □

To obtain optimum *q* value, the offset value φ should be set according to the following rule:

```
if (x mod q ≥r) then φ= x mod r
if (x mod q < r) then φ= (x mod q) - r
```

The rationale of this rule is to satisfy $(x - \varphi = p \bmod q)$ so that $\left\lfloor \frac{x-\varphi-p}{q} \right\rfloor = \left\lfloor \frac{x-\varphi}{q} \right\rfloor = \left\lfloor \frac{x-\varphi+p-1}{q} \right\rfloor$ and there will not be any discretization error for the tolerable region.

Example: For *x=83, q=10, r=5,* φ *=(83 mod 10 -5 = -2.* Therefore for *78* ≤ *X* < *88,*

$d : X \in [0, a] \quad \mapsto \quad \left(\left\lfloor \frac{X - \varphi}{q} \right\rfloor \right)$ gives the same output 8.

Example: For *x=38, q=10, r=5,* φ *=(83 mod 5) = 3* Therefore for 33 ≤ *X* < *43,*

$d : X \in [0, a] \quad \mapsto \quad \left(\left\lfloor \frac{X - \varphi}{q} \right\rfloor \right)$ gives the same output 3.

To summarize, with a discretization method where φ and $HASH\left(\left\lfloor \frac{x - \varphi}{2r} \right\rfloor \right)$ are stored in the system, the check can be done with a single hash and the password space is equal to $\left(\left\lfloor \frac{a}{2r} \right\rfloor \right)$.

3.2 Optimum Discretization Using Multiple Hash Verifications

Generally speaking, passwords guarded by one-way hash functions can be strengthened against brute force attacks by making the hash functions work slower. This technique is effective also against dictionary attacks. Recent work on graphical passwords shows that there may be hotspots on the image and dictionary attacks are powerful against graphical passwords as well [5].

An easy way to implement slow hash functions is repeatedly iterate a standard hash function on the original password. If the scheme is parameterized by the value *k*, then *k* hash computations should be carried out by the system to generate the password hash and also to verify the password and the computation required for brute force attack is increased by the same amount; *k*-times.

If we apply the above method for a one-dimensional value of a graphical password, $h = HASH^k(d(x))$ is computed and stored in the system (together with offset values if used) and to check the value X, $HASH^k(d(X))$ is computed and compared with the value *h.*

As far as graphical passwords are of concern, we have another alternative for improving security using multiple hash computations for verifying the password. In this subsection we introduce this alternative and in the next section we compare its security and performance properties with what we call **"iteration method"** described above.

In the previous work [4], Birget et. al. start designing their discretization method with an assumption. They assume that the discretization must give a single output value for all the input values in the tolerable region so that the output of the discretization will lead to a single hash output at the end. We argue that this is not strictly necessary. Due to the fact that hash functions are computationally cheap, a solution, though it might yield multiple hash outputs, is acceptable (and even desirable if it has better security properties) if it works correctly (i.e. verifies all the passwords in the tolerable region and rejects the rest) using a reasonable amount of hash computations.

Lemma 2: For one-dimensional discretization $d : x \in [0,a] \quad \mapsto \quad \left(\left\lfloor \frac{x}{q} \right\rfloor \right)$, let r be the tolerance value and offset value be zero, the discretization error is at least ± 1 independent of the quantum value q.

Proof: The variable X can take any value between 0 and a. The minimum tolerance value r is equal to 1. Let x take the value $x=(kq-1)$ where k is a positive integer. The discretization result of x is $k-1$ whereas with the tolerance value added the result becomes $X = k$ that results in a discretization error of $+1$.

Let x take the value $x=kq$ where k is a positive integer. The discretization result of x is k whereas with the tolerance value subtracted, the result becomes $X = k-1$ that results in a discretization error of -1. These two results are equivalent to the statement of the Lemma. □

Lemma 3: For one-dimensional discretization *d,* let r be the tolerance value and offset value be zero. The minimum quantum value q that gives discretization error of at most ± 1 is equal to r.

Proof: Taking the tolerance value r into account, the input X takes the value $x-r \leq X \leq x+r$. The discretization result of X is between $\left(\left\lfloor \frac{x-r}{q} \right\rfloor \right) \leq d(X) \leq \left(\left\lfloor \frac{x+r}{q} \right\rfloor \right)$.

There are two relevant cases for q:

i. $r/2 < q < r$:

$$\left(\left\lfloor \frac{x-r}{q} \right\rfloor \right) \leq d(X) \leq \left(\left\lfloor \frac{x+r}{q} \right\rfloor \right) \rightarrow \left(\left\lfloor \frac{x}{q} \right\rfloor - \left\lfloor \frac{r}{q} \right\rfloor - 1 \right) \leq d(X) \leq \left(\left\lfloor \frac{x}{q} \right\rfloor + \left\lfloor \frac{r}{q} \right\rfloor + 1 \right)$$

See how this inequation holds for $x=kq+c$ and $r=q+d$ where $c+d\geq q$.

ii. $q = r$:

$$\left(\left\lfloor \frac{x-r}{q} \right\rfloor\right) \leq d(X) \leq \left(\left\lfloor \frac{x+r}{q} \right\rfloor\right) \rightarrow \left(\left\lfloor \frac{x}{q} \right\rfloor - \left\lfloor \frac{r}{q} \right\rfloor\right) \leq d(X) \leq \left(\left\lfloor \frac{x}{q} \right\rfloor + \left\lfloor \frac{r}{q} \right\rfloor\right)$$

since $\left\lfloor \frac{r}{q} \right\rfloor$ =1, this completes the proof of lemma. □

Corollary: Discretization error increases as q decreases and on its limit it is equal to $\pm r$ when $q = 1$ (no discretization).

More formally stated, discretization error is equal to $\pm\left\lfloor \frac{r}{q} \right\rfloor$ for $q \leq r$.

Definition: For the value X, if the discretization error is $\pm e$ and $h = HASH(d(x))$ is stored in the system, then the total number of hash computation required to check the value X is (at maximum) $2e+1$.

The following pseudocode can be used to check the value X.

```
for (i=-e; i≤e; i++)
    if (h==HASH(d(X)+i)) then return(PASS);
return(FAIL);
```

Theorem 4: For one-dimensional discretization $d : x \in [0,a] \mapsto \left(\left\lfloor \frac{x}{q} \right\rfloor\right)$ parameterized by tolerance value r and quantum $q\leq r$, then maximum number of hash computations required to be performed for off-line brute force attack is

$$P = \lfloor a / q \rfloor$$

Proof: First of all, we should make the differentiation between the related terms of size of the password space and total number of hash computation required for brute force attack. Because

(1) an attacker does not need to try one by one each element in the password space.

(2) testing a single element in the password space requires not only a single hash computation.

Once a point *x* as is checked, *2e* (e is value of the quantization error) points in the neighborhood is also checked automatically. Therefore, to check a space *[0,a],* a clever attacker performs $\left\lfloor \frac{a}{q*2e} \right\rfloor$ checks instead of $\left\lfloor \frac{a}{q} \right\rfloor$ checks and each check requires *2e+1* hash computations. So, approximately $\left\lfloor \frac{a}{q*2e} \right\rfloor * (2e+1) \approx \left\lfloor \frac{a}{q} \right\rfloor$ hash computations are required to check all the values in one-dimension. □

Also note that to have this security level, number of hash computations required to verify the one-dimensional user's password is

$$H = (2\left\lfloor \frac{r}{q} \right\rfloor + 1)$$

In summary, without offsetting and by allowing discretization errors, the minimum number of hash computations required to check one-dimensional graphical password is *3* which gives a security strength of $P = \lfloor a/r \rfloor$. We can further increase verification hash computations to increase *P* which is upper bounded by $P = a$ (for $H = 2r+1$).

4 Comparison

If we compare the results of discretization method explained in subsection 3.2 (we call it the **discretization-with-error method**) with the iteration method when used together with offsetting, we see that for the same amount of verification hash computations $H=2r+1$, they are almost same in their capability to improve the password security against brute force attacks. In a more careful treatment, we make the following observations:

1. For registering or updating the password in the system, the discretization-with-error method only performs one hash computation whereas iteration method uses the same number of hash computations as it uses to verify the password.
2. After introducing their experimental studies to evaluate the usability of graphical passwords, Wiedenbeck et. al. [6] noted that users are likely to click within smaller tolerance values as

their performance become more automated with long-term, regular use of the password. Therefore, if following routine is preferred for discretization-with-error method, the number of hash computations required to verify the password will be reducing by time while hash computations for brute force attack remains same.

```
if (h==HASH(d(X))) then return(PASS);
     for (i=1; i≤e; i++){
          if (h==HASH(d(X)+i)) then return(PASS);
          if (h==HASH(d(X)-i)) then return(PASS);}
return(FAIL);
```

On the other hand, this argument does not hold for the iteration method.

3. Iteration method is more flexible in the sense that we can freely select the number of iterations. On the other hand, to have the same security level, in the discretization-with-error method, number of hash computations is fixed to *2r+1*.
4. Iteration method, since it is used with the offsetting, does require supplementary cleartext information storage which is not required in discretization-with-error method.

In the below table, we summarize the comparisons of all the methods discussed so far (image has dimensions *a* x *b* and number of password clicks is *c)*.

Table 1. Security and performance comparison of discretization methods

	# hash required for breaking the password	**# hash required for checking password**
Birget et. al.'s method [4]	$(\lfloor a/6r \rfloor \lfloor b/6r \rfloor)^c$	1
Discretization using offsetting	$(\lfloor a/2r \rfloor \lfloor b/2r \rfloor)^c$	1
Iteration method	$(\lfloor a/2r \rfloor \lfloor b/2r \rfloor)^c \, x \, k$	k
Discretization-with-error method	$(\lfloor a/2r \rfloor \lfloor b/2r \rfloor)^c \, x \, k$	$\leq k = q(2r+1)$ $q \in \mathbb{Z}^+$

Table 1 shows that it is possible to improve the security of robust discretization method proposed by Birget et. al. by a factor of 3^{2c} . If a further security improvement is desired, we also show it is possible with an increased cost for checking the passwords.

5 Discussion

5.1 Reducing Number of Clicks

There are different ways to appreciate the results explained in section 4. For instance, by using one of our proposed methods instead of Birget et al.'s method it is possible to decrease the number of clicks used to generate the user's graphical password while having the same security level.

Example: In the example given in [4], *a=330 mm, b= 260 mm* and *r = 1 mm* values are used. Number of possible passwords can be computed and is around 7×10^{16} for $c = 5$ clicks. Since in our method *q* is reduced from *6 mm* to *2 mm*, number of possible passwords is almost same (around 5×10^{16}) with one less click, $c = 4$.

5.2 Incorrect Clicks

In all of the methods we propose in this paper, a click that is at a distance of *r+1* to the original click location is guaranteed to be treated as incorrect. In the previous work, this distance is equal to $5\sqrt{2}r \approx 7.07r$.

5.3 Hybrid Schemes

Hybrid schemes are possible i.e. for one or more dimensions we can use one method, in others we can prefer others. For hybrid schemes, an optimization problem is as follows:

"Given the values *a, b, c, r* and also the real-life statistical data for user clicks (how close accepted user clicks are to the original click locations), find the best discretization strategy to maximize the number of hash computations required for breaking the password while keeping the average verification cost below a certain limit." We leave this problem as a future work.

5.4 Dynamically Changing Tolerance Values

Previous work noted that users are likely to click within smaller tolerance values as their performance become more automated with long-term, regular use of the password. Using this fact, in order to safeguard against various attacks (e.g. guessing attacks) it is possible to decrease the tolerance value by time. This dynamic change can be realized without modifying the password hash and other system parameters only if discretization-with-error method is preferred.

6 Conclusion

In this paper, we presented several discretization methods for click-based graphical passwords to optimize the password space and the strength against brute force attacks. In the first of these methods, by considering each *x* and *y* values of user clicks separately, we succeeded in selecting optimum offset values for discretization. The method we have proposed increases the password space by 3^{2c} where *c* is the number of clicks.

As a second method, instead of using hash iterations to slow down the brute force search, we show that discretization errors can be allowed to have a similar security improvement. We also prove the maximum upper limit for this improvement in the methods using discretization errors.

References

[1] S. Wiedenbeck, J. Waters, J.C. Birget, A. Brodskiy, N. Memon, ``PassPoints: Design and longitudinal evaluation of a graphical password system", *International J. of Human-Computer Studies (Special Issue on HCI Research in Privacy and Security)*, 63 (2005) 102-127.

[2] S. Chiasson, P.C. van Oorschot, R. Biddle. Graphical Password Authentication Using Cued Click Points. Springer-Verlag LNCS 4734 (2007), pp.359-374.

[3] G.E. Blonder, "Graphical Passwords", *United States Patent 5559961* (1996).

[4] J.C. Birget, D. Hong, N. Memon, "Graphical Passwords Based on Robust Discretization", *IEEE Transactions on Information Forensics and Security* 1(3) (Sept. 2006) 395-399.

[5] J. Thorpe, P.C. van Oorschot. Human-Seeded Attacks and Exploiting Hot-Spots in Graphical Passwords. *Proc. 16th USENIX Security Symposium*, Aug.6-10 2007, Boston, MA.

[6] S. Wiedenbeck, J. Waters, J.C. Birget, A. Brodskiy, N. Memon, ``Authentication using graphical passwords: Effects of tolerance and image choice", *Proc. Symposium on Usable Privacy and Security*, July 2005.