PINDragon Explanation and Calculations

Guohui Zhao, Yan Wang, Kemin Xu
Department of Statistics
University of Georgia

October 20, 2015

1 Introduction

1.1 Purpose of *PINDragon*

The *PINDragon* interface is a newly designed passcode generator created in order to make passcodes more secure. The *PINDragon* interface is different from a traditional keypad in that it utilizes more properties of key layout, as discussed below. Thus, the possible combination of passcodes that clients can create through *PINDragon* is much larger than that through traditional keypad systems. In addition, it introduces a novel technique (i.e. shuffling) to rearrange the properties on each key after each password attempt, which makes decoding the passcode more difficult.

1.2 Concept of Key Properties

PINDragon adopts a K-key console design (the traditional keypad design used in our examples has $K = 10$ keys, arranged as shown in Figure 1). The traditional keypad design has only two properties, Position and Numeral, and they are linked in that the key in position [1] bears the numeral '1', the key in position [2] bears the numeral '2', ..., the key in the bottom center ([0] position) bears the numeral '0'. Under the *PINDragon* interface, however, up to P=7 properties (Position, Center Numeral, Color, Upper Left Character, Upper Right Character, Lower Left Character, and Lower Right Character) can be displayed for each key, as demonstrated in Figures 1 and 2 on page 4 of this report. Each of the P property sets contains K different levels, with each of the K levels of a property set displayed exactly once in the keyboard configuration. Table 1 below displays the $K = 10$ levels that are used for each of the $P = 7$ property sets under the current implementation of *PINDragon*.

Table 1: Baseline Property Sets and Elements

Set	Property	e1	e2	e3	e4	e5	e6	e7	e8	e9	e10
1	Position	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[0]
2	Central Numeral	1	2	3	4	5	6	7	8	9	0
3	Color	Red	Yellow	Green	Blue	Orange	White	Black	Brown	Gray	Purple
4	Upper Left	▲	■	⬠	⬡	●	♣	♡	♢	♠	★
5	Upper Right	A	B	C	D	E	F	G	H	!	@
6	Lower Left	I	J	K	L	M	N	O	P	Q	$
7	Lower Right	R	S	T	U	V	W	X	Y	Z	&

Table 2: Permutations Yielding Keypad of Figure 1

Set	**Property**	**e1**	**e2**	**e3**	**e4**	**e5**	**e6**	**e7**	**e8**	**e9**	**e10**
1	Position	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[0]
2	Central Numeral	2	1	6	4	9	5	3	8	7	0
3	Color	Blue	Green	Brown	Black	White	Orange	Yellow	Gray	Red	Purple
4	Upper Left	■	♣	★	♡	♠	⬡	▲	♢	⬠	●
5	Upper Right	B	G	@	F	!	D	A	H	C	E
6	Lower Left	I	J	K	L	M	N	O	P	Q	$
7	Lower Right	T	S	X	V	&	U	R	Y	Z	W

Table 3: Permutations Yielding Keypad of Figure 2

Set	**Property**	**e1**	**e2**	**e3**	**e4**	**e5**	**e6**	**e7**	**e8**	**e9**	**e10**
1	Position	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[0]
2	Central Numeral	8	7	4	0	1	9	3	6	5	2
3	Color	Green	Blue	Yellow	Grey	Purple	Black	White	Red	Orange	Brown
4	Upper Left	⬠	★	●	♣	♠	▲	■	⬡	⬠	♢
5	Upper Right	F	E	@	C	H	B	D	A	G	!
6	Lower Left	Q	L	N	M	J	K	P	O	$	I
7	Lower Right	&	W	X	S	U	Y	Z	V	R	T

As we will soon see, increasing either K or P improves the security of the design, but increasing either creates other difficulties. One could increase K relatively easily. For example, one could increase from $K = 10$ to $K = 12$ rather easily by adding two keys at the bottom of the traditional keypad, to the immediate left and right of the key in the [0] position. If one did this, however, one would need to create two more levels for each of other properties shown in Table 1. Increasing the number of Property sets from $P = 7$ to a larger value would be very problematic, as the keys are already extremely cluttered with information when $P = 7$. Thus, while formulas for general (K,P) will be presented in this report, we will primarily be focused on the case where $K = 10$ and $P = 7$, since these appear to be the upper limits currently feasible.

Simply assigning more properties (than the standard two) to each key will not, of itself, make the passcode more secure. For example, if the keypad always looked as shown in Figure 1, it wouldn't deter an intruder who happened to observe a user press, in sequential order, the keys in positions ([1],[2],[7],[9]). To gain access to the user's account, the intruder might think that s/he should also press the keys in positions ([1],[2],[7],[9]), or s/he might feel that s/he should press the keys with central numerals ('2','1','3','7') or perhaps the keys with colors (Blue, Green, Yellow, Red). If the keypad remains fixed for each log-in by the user, the intruder could simply press the same key sequence that s/he had observed and gain access to the user's account. The crucial innovation which the *PINDragon* system incorporates is that the K elements of each property are permuted (called a 'shuffle') each time the user logs in. So, for example, the keypad configuration shown in Figure 1 is what would be displayed if the properties (shown in unshuffled baseline form in Table 1) were permuted so as to get what is shown in Table 2. Table 3 displays the permutations which would yield the configuration shown in Figure 2. Note that the property 'Position' is never shuffled (the keys are located in fixed places on the keypad), but the other six property sets each have their $K = 10$ elements permuted, with the first element in each permutation displayed on the key in position [1], the second on the key in position [2], ..., the tenth on the key in position [0]. Simple combinatorics shows that there would be $(K!)^{P-1}$ possible shuffled keypad configurations, or 2.28×10^{39} for the case $(K = 10, P = 7)$.

Although many different keypad configurations can be generated, for the *PINDragon* system to be effective, it must be easy for a user (who knows his/her pre-set passcode) to determine which sequence of keys to push, but difficult for an intruder/observer to deduce the mechanism by which the user is selecting the key sequence. So as not to burden the user, as with traditional passcodes, a *PINDragon* user will need only to memorize, as his/her passcode, a sequence of L items. However, unlike traditional passcodes, these items won't necessarily be elements from the set of integers $0 - 9$. Since traditional passcodes are of length $L = 4$, we will demonstrate with passcodes of that length, although any length, in theory, could be used. As was the case with number of keys (K) and number of properties (P), it is easy to show that security will increase as passcode length (L) increases. However, it will be much easier, operationally, to increase L than it is to increase K or P, so in our numerical calculations of Section 3, we offer specific numerical results for $L = 4, 5, 6$, and 7. Of course, passcodes of length $L > 7$ could be used, but, at some point, users will find the codes too long to easily remember, and will balk at using them. For a passcode of length L, under the *PINDragon* system, the user must specify a property and an element of that property for each of the L keys in the passcode sequence. For example, the passcode which the user in Figures 1 and 2 pre-specified when s/he set up the account is as shown in Table 4 below.

Table 4: Passcode Used in Figures 1 and 2

Sequence	Property	Element
1	Upper Left	■
2	Lower Left	J
3	Central Numeral	3
4	Central Numeral	7

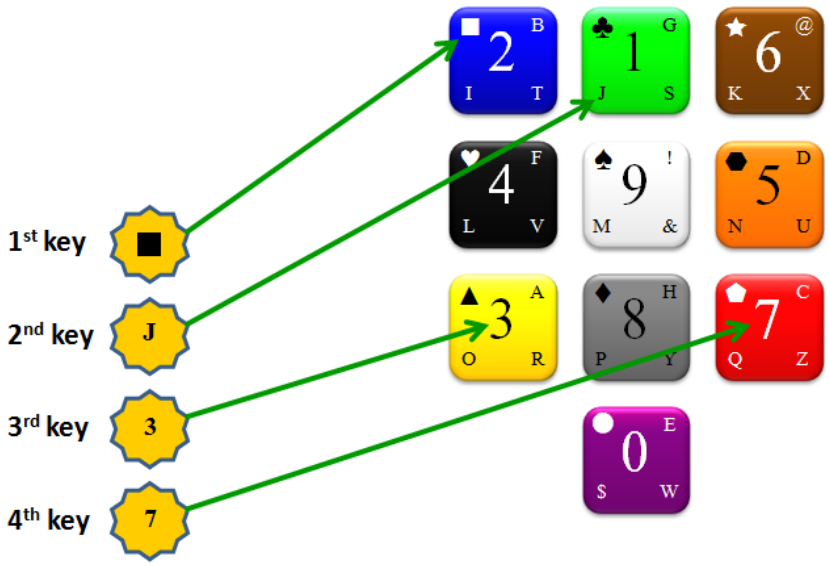

Figure 1: Keypad Layout for Self-selected Passcode before Shuffling

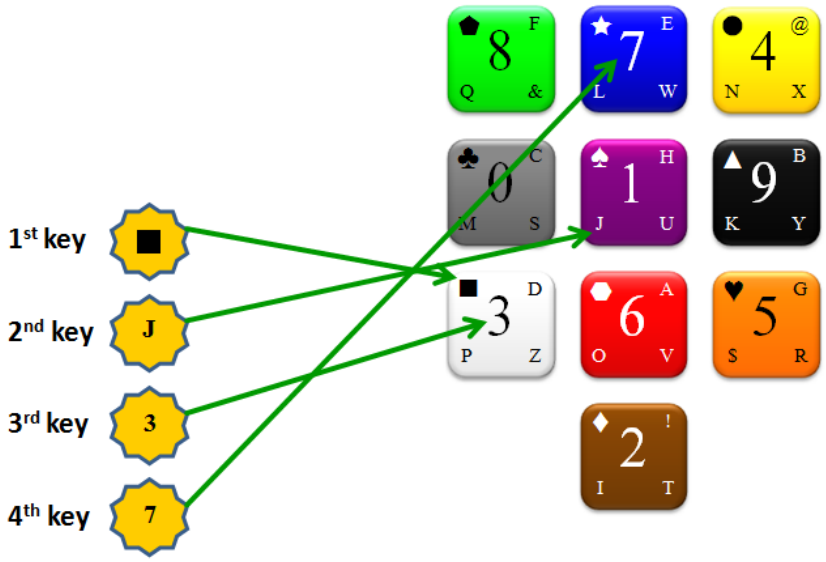

Figure 2: Keypad Layout for Self-selected Passcode after Shuffling

In setting up the passcode, for each press, the user may specify any of the 7 property types, and any element of the specified property type. We suspect that the most popular properties selected by users will be Position, Color, and Central Numeral, since all of these are easy to discern from a quick glance at the keypad. If one selects "square in upper left", as the user in Table 4 did for his/her first press, s/he would need to scan the upper left corners of each of the 10 keys until the square was observed in one of them. While this doesn't take particularly long to do, one can imagine that users accustomed to fast entry might prefer to memorize something like (Yellow, 8, Green, Middle Position [5]) as their 4-key sequence. Users who utilize a variety of different properties in their passcodes will be more securely protected than those who don't, but many users may not appreciate the extra hassle and may revert to passcodes that are easier for them (the users) to remember but which are also easier for intruders to crack.

Figures 1 and 2 demonstrate which keys (and in which order) a user who had the $L = 4$ passcode shown in Table 4 should press in order to gain access to the keypads as shuffled in Figure 1 and Figure 2. If an intruder were watching this log-on process from a distance and could observe only the positions of the keys pushed ({[1],[2],[7],[9]} for the first shuffle, and {[7],[5],[7],[2]} for the second), the intruder would have no idea what sequence should be pressed the next time the shuffled keypad was presented. If the intruder possessed a high-resolution camera (or had spyware installed on the computer), so that s/he could observe all 7 properties on each of the pressed keys, s/he might eventually decode the passcode, but it would take a while. Our goal in the next section is to demonstrate how a determined intruder might do this, and to estimate how quickly and accurately such a decoding attempt might be.

1.3 Goals of Report

The most important feature introduced by *PINDragon* to improve security is shuffling of property elements on the display keypad each time a user logs on. In the calculations which are given in Section 3 of this report, we will assume that there is random permutation of all properties (except Position) each time a new shuffle (log-in) is performed. In fact, the shuffling algorithm, as currently implemented in *PINDragon*, doesn't always permute all properties. (In Tables 1-3 above, for example, a very careful observer will note that the Lower Left sequence wasn't permuted from Table 1 to Table 2.) It has been suggested by the *PinDragon* designers that it might be optimal to permute about half (3, in this case) of the properties between shuffles, but to leave the others as they were on the previous shuffle, thus temporarily linking some properties. The usefulness of doing this is discussed at the end of Section 4, but for the remainder of this report, we will assume that independent permutation of the elements of all the (non-Position) properties is performed at each new shuffle.

A major purpose of the report is to calculate the probability that a passcode with specific length (L) can be cracked by a malicious intruder. The answer does not depend on L alone; it also depends on the number of keys (K) on the keypad, the number of properties (P) associated with each key, and the number of shuffles (S) which the intruder has viewed. This is explained in great detail in Sections 2 and 3 of this report. Section 2 defines the variable names and outlines the process by which an intruder would attempt to infer the

passcode. Section 3 contains theoretical formulae for calculating the probability of detection as a function of (K, P, L, and S), along with some specific numerical calculations of interest. Section 4 presents our conclusions.

2 Definitions and Decoding Process

In the *PINDragon* problem, we define the following variables:

- P = Number of available Properties (including Position)
- K = Number of Keys on keypad
- L = Length of passcode
- S = Number of views of successful entries on shuffled keyboards

We assume that each of the K keys utilizes all P possible properties and that each of the properties has K unique levels, with exactly one level of each property used on each of the K keys each time a shuffle is performed. For the calculations below, we assume that the K levels of each (non-Position) property are randomly (and *independently*) permuted on each shuffle, with the first element in each Property class permutation assigned to key position [1], the second to key position [2], ..., to key position [K]. We further assume that a passcode of length L for the user has previously been stored in the *PINDragon* system, where the information for each key in the passcode sequence is of the form (Sequence Number, Property Used, Element Used), such as shown in Table 5 below:

Table 5: Sample Passcode of Length L=4

Sequence	Property	Element
1	Upper Left	■
2	Color	Orange
3	Center Numeral	3
4	Lower Left	J

We assume that intruder doesn't know the user's passcode, but believes that at some future date s/he can gain access to a first-level identification verification for the user. (In current practice, this would be equivalent to stealing a user's bankcard or being able to log-in as the user. Under current technology, the only remaining safeguard at that point is something like a 4-digit PIN code. If the intruder knows or can guess the users's PIN code and has the user's card, s/he has unfettered access to the user's accounts.) Under the conventional system, an intruder who observed a user log in once previously would be able to successfully log in to that user's account after that one viewing (assuming s/he observed the initial log-in clearly). Under the *PINDragon* system, an intruder won't be able to deduce the code after one viewing, but as more information accumulates, s/he will be able to do so. The process for doing so is illustrated with a specific example below – general probability formulae are provided in the following section.

2.1 Intrusion Example

Let us suppose that we have the case where there are $K = 10$ keys, $P = 7$ properties, and that the *PINDragon* user has specified the $L = 4$ passcode shown in Table 5 above. Let us further suppose that intruder observes $S = 3$ shuffles (log-ons) for this user, where the first log-in results in the keypad configuration shown in Figure 1, the second log-in results in the keypad configuration of Figure 2, and the third in an un-shown configuration. (Note that the arrows drawn on Figures 1 and 2 are not relevant here, as they refer to a user with Passcode as given in Table 4, while we are assuming that this user is using the slightly different passcode shown in Table 5). Assuming the user types his passcode in correctly, if the intruder were very careful and had a camera to record the sequence of keys pressed at each of the $S = 3$ log-ins, s/he would be ale to make a summary table like Table 6 below.

Table 6: Algorithm to decode the passcode

Press	Shuffle	Position	Color	Center #	UpLeft	UpRight	LowLeft	LowRight
First	S=1	[1]	Blue	2	■	B	I	T
	S=2	[7]	White	3	■	D	P	Z
	S=3	[0]	White	8	■	E	L	&
	Shuffle	Position	Color	Center #	UpLeft	UpRight	LowLeft	LowRight
Second	S=1	[6]	Orange	5	●	D	N	U
	S=2	[9]	Orange	5	♡	G	$	R
	S=3	[9]	Orange	4	♦	F	Q	X
Press	Shuffle	Position	Color	Center #	UpLeft	UpRight	LowLeft	LowRight
Third	S=1	[7]	Yellow	3	▲	A	O	R
	S=2	[7]	White	3	■	D	P	Z
	S=3	[8]	Red	3	♠	!	K	W
Press	Shuffle	Position	Color	Center #	UpLeft	UpRight	LowLeft	LowRight
Fourth	S=1	[2]	Green	1	♣	G	J	S
	S=2	[5]	Purple	1	♠	H	J	U
	S=3	[3]	Grey	0	⬡	D	J	Y

Table 6 illustrates the simple algorithm which the intruder could use to decode the passcode. Each key of the *PINDragon* keypad contains one element of each of the $P = 7$ properties (Position, Color, Center Numeral, Upper Left Character, Upper Right Character, Lower Left Character, and Lower Right Character). An intruder with a camera could clearly record which key was being pressed on each of the $L = 4$ presses of a $K = 10$ keypad. For the j-th press for each shuffle, the intruder could make a list, like that shown in Table 6, showing what element for each property is on the pressed keys. The intruder doesn't know which property on a certain press is the correct property, but s/he can very quickly deduce which is **not** correct. For the first pressed button, even by $S = 2$, the intruder would know that the property is Upper Left (i.e. Shape) and that the shape element is ■, since that is the only property among the 7 whose element is repeated in shuffles 1 and 2. For presses j=2, j=3, and j=4, the intruder doesn't know for sure what the property is after $S = 2$ shuffles, but s/he has narrowed it down to two properties for each, and by the time s/he has observed $S = 3$, s/he knows that the answers are j=2; Property=Color(Orange), j=3; Property=Center Numeral(3), j=4; Property=LowLeft(J). So, after $S = 3$ shuffle viewings, in this case, the intruder would know the code and could gain access. This is not a specially

chosen example – we will calculate in the next section that if $K = 10$, $P = 7$, and $L = 4$, there is a 78.6% chance that the intruder would know the passcode within $S = 3$ shuffled viewings, with the probability rising to 97.6% for $S = 4$ viewings.

3 Formulae Under Complete Independent Random Shuffling

We desire to calculate the probability that an intruder will be able to fraudulently log in as a user whom s/he has previously observed. (We are assuming that the intruder has somehow gained the card to insert in the system so that the *PINDragon* system knows who the user should be, and is simply verifying whether the intruder is entering the correct passcode. We assume that the intruder has previously viewed S successful logins on shuffled keypads for this specific user.) Below, we give formulae for calculating two probabilities, **P(Know)** and **P(Guess)**, as a function of (P,K,L,S). The first, **P(Know)** is the probability that the intruder will know the passcode after S viewings (or equivalently, the probability that the intruder will successfully gain entry if s/he tries to gain access only when s/he knows the complete code). The second, **P(Guess)**, is the probability that the intruder will gain access if s/he enters the key-codes s/he knows and randomly guesses among the eligible options for the key-codes s/he doesn't know. Of course, for fixed values of (P,K,L,S), P(Know) is a subset of P(Guess).

3.1 Calculation of P(Know)

If the *PINDragon* intruder has viewed $S = 0$ or $S = 1$ successful logins for one user on shuffled keypads, it is impossible that s/he would know the passcode. However, after an *PINDragon* intruder has viewed $S(S > 1)$ successful logins on shuffled keypads, there is non-zero probability that the code will be cracked. This will occur when no property except the true key property has a repeated element over the S shuffles observed. The probability that any element of a property will repeat on the same key for all S viewings is $(\frac{1}{K})^{S-1}$, since every property has exactly K unique levels. In order to know the exact level of the property of the passcode, there should be no repeat levels of all the other $(P-1)$ properties. Therefore, equation (1) is used to calculated $P(Know)$ of the L length of passcode.

$$P(Know) = \left[1 - \frac{1}{K^{S-1}}\right]^{(P-1)L} ; (S > 1) \tag{1}$$

3.2 Calculation of P(Guess)

It is reasonable that for a fixed number (S) of shuffles, $P(Guess)$ is a little larger than $P(Know)$. If an intruder has never viewed a Shuffling (S=0), the probability of guessing is the same as in the traditional case ($(\frac{1}{K})^L$), since the intruder has probability of $\frac{1}{K}$ to guess one key of the passcode correctly. If an intruder has had $S = 1$ viewing, there is zero probability that s/he knows the passcode, but his/her odds of guessing improve slightly from $S = 0$ case, since s/he can eliminate a few properties. This increased probability of guessing when $S = 1$ is shown in equation (3). In general, the formula for the probability of guessing for

$S \geq 2$ is given by equation (4) below. This is a somewhat complex formula which conditions on the number of elements repeated after S Shufflings, assuming that the intruder correctly entered the keys which s/he knows, and randomly guesses among the eligible elements for the other keys.

$$P(Guess) = \left[\left(\frac{1}{K}\right)\right]^{L}; (S = 0) \tag{2}$$

$$P(Guess) = \left[\left(\frac{1}{P}\right)\left(1 + \frac{P-1}{K}\right)\right]^{L}; (S = 1) \tag{3}$$

$$P(Guess) = \left[\sum_{j=0}^{P-1} \binom{P-1}{j} \left(\frac{1}{K^{S-1}}\right)^{j} \left(1 - \frac{1}{K^{S-1}}\right)^{P-1-j} \left(\frac{1}{1+j}\right)\right]^{L}; (S \geq 2) \tag{4}$$

Tables 7 and 8 below list **P(Know)** and **P(Guess)** for various values of increasing S with $P = 7$, $K = 10$, and $L = 4, 5, 6$, and 7, assuming complete random independent shuffling. The actual *PINDragon* system, since it uses correlated shuffling, will yield slightly lower probabilities of intrusion, as explained in Section 4.

Table 7: Probability of Successful Intrusion (P=7, K=10, L=4 and 5)

	L=4		L=5	
	P(Known)	P(Guess)	P(Known)	P(Guess)
S=0	0.0000	0.0001	0.0000	0.0000
S=1	0.0000	0.0027	0.0000	0.0006
S=2	0.0798	0.3085	0.0424	0.2299
S=3	0.7857	0.8871	0.7397	0.8609
S=4	0.9763	0.9881	0.9704	0.9851
S=5	0.9976	0.9988	0.9970	0.9985

Table 8: Probability of Successful Intrusion (P=7, K=10, L=6 and 7)

	L=6		L=7	
	P(Known)	P(Guess)	P(Known)	P(Guess)
S=0	0.0000	0.0000	0.0000	0.0000
S=1	0.0000	0.0001	0.0000	0.0000
S=2	0.0225	0.1714	0.0120	0.1277
S=3	0.6964	0.8355	0.6557	0.8109
S=4	0.9646	0.9822	0.9588	0.9792
S=5	0.9964	0.9982	0.9958	0.9979

4 Conclusion

PINDragon is a novel passcode generator which offers clients more choices when designing their passcodes. It introduces a new algorithm - the shuffle - to further increase the difficulty

in decoding. In the present report, we assume that an intruder has viewed a user's successful logins on shuffled keypads and want to fraudulently log in as the client. We computed the probability that the intruder would know the correct passcode after certain viewings, and the probability that the intruder will gain access by optimized guessing. Based on the probability results from a length of passcode from 4 to 7, we conclude that up to five views are enough to decode all passcodes ($P(Know) > 0.99$ and $P(Guess) > 0.99$). Longer passcodes are not considered in the report since we doubt that clients would use them. However, there is no doubt that increasing the length (L) of the passcode is the single easiest modification to decrease the probability that the intruder, after a fixed number of views (S), could decipher the passcode.

The actual *PINDragon* system, which uses correlated shuffling, will yield slightly lower probabilities of intrusion for fixed values of (P, K, L, S). This will occur because correlated shuffling will tend to increase simultaneous runs of repeated elements from different properties on the same key, this making it more difficult to make a unique identification until more views have been observed. This is counter-balanced by the fact that non-shuffling makes it easier for an intruder to gain entry simply by "doing what was done before". The *PINDragon* formulators are correct that the optimal correlation to induce (in order to hinder intruder detection capabilities) would occur if about half of all properties are randomly selected to be permuted while the other half are not permuted. Unfortunately, a closed form answer for P(Know) and P(Guess) under this scenario isn't possible. Such an answer could be obtained by simulation, if desired. Preliminary results indicate that this technique isn't really that effective at preventing a correct guess – perhaps requiring one additional shuffle viewing than would be required under independent sampling. Certainly, this 'half shuffle' technique would be much less effective at preventing detection than simply increasing the passcode length (L) by 1.

Overall, a determined intruder will eventually defeat *PINDragon*. However, such very determined anatagonists are probably not the main enemy against which *PINDragon* is competing. For the cases $S = 1$ and $S = 2$, which are probably most common in real life, the system seems fairly secure, with the even the riskiest ($L = 4$) passcode length not being guessable more than 31% of the time. Similarly, even if a potential intruder gains many viewings (S), if s/he can't get a good look at all the properties on the selected keys, s/he will not be able to guess using the detection algorithm introduced in Section 2.1. In that sense, the detection probabilities shown in Tables 7 and 8 are worst-case scenarios – the "in practice" detection probabilities for the *PINDragon* system are likely much lower than those shown.

5 Acknowledgment

We gratefully acknowledge extremely valuable scientific discussion with Professor Jaxk Reeves, Director of the UGA Statistical Consulting Center and Professor Nicole Lazar, UGA Statistics Department Head.

Appendix R code

```
##############################################Calculation
################################################L=4
###P know
L=4
###S=2
(dbinom(0, size=6, prob=0.1) )^L
###S=3
(dbinom(0, size=6, prob=0.01) )^L
###S=4
(dbinom(0, size=6, prob=0.001) )^L
###S=5
(dbinom(0, size=6, prob=0.0001) )^L
###P guess
L=4
###S=2
s=rep(0,7)
for (k in 1:7){
s[k]=dbinom(k-1, size=6, prob=0.1)/(k)
}
(sum(s))^L
###S=3
s=rep(0,7)
for (k in 1:7){
s[k]=dbinom(k-1, size=6, prob=0.01)/(k)
}
(sum(s))^L
###S=4
s=rep(0,7)
for (k in 1:7){
s[k]=dbinom(k-1, size=6, prob=0.001)/(k)
}
(sum(s))^L
###S=5
s=rep(0,7)
for (k in 1:7){
s[k]=dbinom(k-1, size=6, prob=0.0001)/(k)
}
(sum(s))^L
```

```
################################################L=5
###P know
L=5
###S=2
(dbinom(0, size=6, prob=0.1) )^L
###S=3
(dbinom(0, size=6, prob=0.01) )^L
###S=4
(dbinom(0, size=6, prob=0.001) )^L
###S=5
(dbinom(0, size=6, prob=0.0001) )^L

###P guess
L=5
###S=2
s=rep(0,7)
for (k in 1:7){
s[k]=dbinom(k-1, size=6, prob=0.1)/(k)
}
(sum(s))^L
###S=3
s=rep(0,7)
for (k in 1:7){
s[k]=dbinom(k-1, size=6, prob=0.01)/(k)
}
(sum(s))^L
###S=4
s=rep(0,7)
for (k in 1:7){
s[k]=dbinom(k-1, size=6, prob=0.001)/(k)
}
(sum(s))^L
###S=5
s=rep(0,7)
for (k in 1:7){
s[k]=dbinom(k-1, size=6, prob=0.0001)/(k)
}
(sum(s))^L

################################################L=6
###P know
L=6
###S=2
(dbinom(0, size=6, prob=0.1) )^L
###S=3
(dbinom(0, size=6, prob=0.01) )^L
```

```
###S=4
(dbinom(0, size=6, prob=0.001) )^L
###S=5
(dbinom(0, size=6, prob=0.0001) )^L
###P guess
L=6
###S=2
s=rep(0,7)
for (k in 1:7){
s[k]=dbinom(k-1, size=6, prob=0.1)/(k)
}
(sum(s))^L
###S=3
s=rep(0,7)
for (k in 1:7){
s[k]=dbinom(k-1, size=6, prob=0.01)/(k)
}
(sum(s))^L
###S=4
s=rep(0,7)
for (k in 1:7){
s[k]=dbinom(k-1, size=6, prob=0.001)/(k)
}
(sum(s))^L
###S=5
s=rep(0,7)
for (k in 1:7){
s[k]=dbinom(k-1, size=6, prob=0.0001)/(k)
}
(sum(s))^L

##################################################L=7
###P know
L=7
###S=2
(dbinom(0, size=6, prob=0.1) )^L
###S=3
(dbinom(0, size=6, prob=0.01) )^L
###S=4
(dbinom(0, size=6, prob=0.001) )^L
###S=5
(dbinom(0, size=6, prob=0.0001) )^L
```

```
###P guess
L=7
###S=2
s=rep(0,7)
for (k in 1:7){
s[k]=dbinom(k-1, size=6, prob=0.1)/(k)
}
(sum(s))^L
###S=3
s=rep(0,7)
for (k in 1:7){
s[k]=dbinom(k-1, size=6, prob=0.01)/(k)
}
(sum(s))^L
###S=4
s=rep(0,7)
for (k in 1:7){
s[k]=dbinom(k-1, size=6, prob=0.001)/(k)
}
(sum(s))^L
###S=5
s=rep(0,7)
for (k in 1:7){
s[k]=dbinom(k-1, size=6, prob=0.0001)/(k)
}
(sum(s))^L
```