Analyzing and Improving
Challenge Strings for CAPTCHAs
DRAFT 0.27

Jon Bentley and Colin Mallows

Avaya Labs, 233 Mt. Airy Road, Basking Ridge, NJ 07920, USA
{jbentley, colinm}@avaya.com

Abstract. A CAPTCHA is a Completely Automated Public Test to tell Computers and Humans Apart. Typical CAPTCHAs present a challenge string consisting of a visually distorted sequence of letters and perhaps numbers, which in theory only a human can read. Attackers of CAPTCHAs have two primary points of leverage: Optical Character Recognition (OCR) can identify some characters, while nonuniform probabilities make other characters relatively easy to guess. This paper uses a mathematical theory of assurance to characterize the probability that a correct answer to a CAPTCHA is not just a lucky guess. We examine three common types of challenges: dictionary words, Markov text, and random strings. We propose a new class of challenges based on the consonant-vowel-consonant (CVC) trigrams of psychology, and new varieties of Markov text. Theory and experiments reveal that many challenges are not as strong as believed, so we present ways to strengthen some. Almost all of our results apply immediately to automated “password generators”.

1 Introduction

The goal of this paper is to analyze the assurance of the challenge strings employed by CAPTCHAs. If a system chooses the challenge word “cat” from a small dictionary, and the subject correctly identifies it, then we have some assurance that the subject is a human who can read the phrase, and not a program that has made a lucky guess. If the subject correctly identifies the challenge string “2R97WZ8E” in which each of 8 characters is chosen at random, then we have even more assurance. Our theory of assurance takes into account attackers who employ imperfect Optical Character Recognition (OCR), clever guessing, and combinations of the two.

Section 2 introduces the framework that we will use to analyze the assurance of CAPTCHAs. We study random strings in Section 3, dictionary words in Section 4, and survey Markov text in Section 5 (the topic is described in detail in Appendix A). Section 6 introduces the psychological tool of Consonant-Vowel-Consonant or CVC trigrams as CAPTCHA challenges. The methods are compared in Section 7, and conclusions are offered in Section 8.
\section{The Analysis of CAPTCHAs}

We consider three players in a CAPTCHA. The CAPTCHA is administered by a broker, B, who presents a degraded image of a challenge string to the subject, S. That subject might be the human customer, C, who always reports the correct result. On the other hand, the subject might be an attacker, A, who attempts to use a combination of imperfect OCR and clever guessing to impersonate C.

There are several sources of randomness in this process. One is the broker’s selection of the challenge from a distribution of strings. Another is the stochastic success of the OCR in the presence of randomized visual degradation. Yet another is randomization that the attacker might use to make a guess. We must be careful to state the source of randomness assumed in any probabilistic statement.

We will begin by considering the simplest possible attack: a “blind” attacker employs no OCR but instead makes random guesses. The attacker knows the length of the challenge, \( N \), and the character set, and the distribution from which the characters are chosen. We start with the simplest character set and distribution: the characters are the binary digits 0 and 1, and each is chosen equiprobably (perhaps by tossing a fair coin); thus a challenge might be “0110110111”. The broker displays the visually degraded bit string to the subject, who enters it as text. If the answer is incorrect, under our assumptions, then we know that we have correctly detected the attacker A. If the answer is correct, though, then we have gained assurance that the subject is not A, but rather the customer C.

If the subject correctly identifies a 5-bit challenge, then we have some assurance; if he identifies 10 bits, then we have even more assurance. Because the attacker is blind and the binary digits are chosen at random, his probability of successfully identifying any particular character is precisely 1/2. This attacker could adopt any number of strategies: he could always guess a string of 1s, always guess a string of 0s, alternate between 0s and 1s, or toss a random coin with 0 on one side and 1 on the other. Before the broker B conducts the experiment, he knows that no matter what A’s policy is, the probability of A being correct is exactly 1/2 for each binary digit. The attacker’s probability of being correct on all \( N \) binary digits is therefore \((1/2)^N\), or \(2^{-N}\).

Following Bentley and Mallows [2005, Section 2], we take as axiomatic that assurance is measured by the probability \( p \) that an attacker can guess a secret. This probability may be either \textit{a priori} (when it is computed before the secret is chosen) or \textit{a posteriori} (when it is computed with respect to a particular secret that has been chosen). We could conduct the entire analysis in terms of probabilities, but we will find it more convenient to speak in terms of “bits of assurance” against a certain class of attackers, which is \(-\log p\). This yields the following

\textbf{Definition 1.} We write \( \text{assurance}(A) = -\log P(A) \), where \( P(A) \) is the probability that an attacker A correctly guesses a secret.

In the simple CAPTCHA that presents a string of random binary digits, the secret is the string. We saw above that a blind attacker guesses all \( N \) binary digits correctly with \textit{a priori} probability \( 2^{-N} \). Because \( \log 2^{-N} = -N \), a string of \( N \) random binary digits

\footnote{Bentley and Mallows [2005] focus on the case of a secret selected by a customer, such as a Personal Identification Number (PIN) associated with a bank ATM card. Because the customer cares primarily about his particular PIN, he is concerned primarily with \textit{a posteriori} assurance. In this context of CAPTCHAs, \textit{a priori} assurance is of primary interest.}
gives N bits of assurance against any blind attacker. We chose this example precisely so that each displayed bit provides exactly one bit of assurance.

3 Random Strings

The definition of assurance generalizes easily to challenge strings chosen uniformly from a fixed set.

Theorem 2. A challenge string chosen uniformly from a universe of size \( N \) provides \( \lg N \) bits of \textit{a priori} assurance against a blind attacker.

Proof. Whatever strategy the blind attacker uses to make his guess, the probability that the broker generates that particular guess is \( 1/N \), which corresponds to \(-\lg 1/N = \lg N\) bits of assurance. QED.

The randomness in this statement is provided by the broker’s selection. If the challenge is chosen uniformly from a dictionary of \( N \) words, the result is \( \lg N \) bits of assurance. If the challenge consists of \( L \) symbols chosen at random from an alphabet of size \( A \), the result is \( \lg A^L = L \lg A \) bits of assurance.

This section will consider challenge strings composed of random letters and numbers. We might start by considering the 26 upper case letters and 10 digits, for a total of 36 characters. But because the letter O and the digit 0 are easily confused, as are the letter I and the digit 1, we will omit those four characters to leave 24 letters and 8 digits, for a total of 32 characters. Here are 7-character challenges randomly chosen in this fashion, presented without human editing:

348UVLP 6F4ZRGD BPLDPVV 2R97WZ8 SHK3AJ7 MP589ET

Dictionary words are easy for subjects to recognize and to remember (for hunt-and-peck typists who look away from the CAPTCHA to the keyboard as they type the word), but provide little assurance. Random strings like those above are their dual: they provide much assurance, but are difficult to recognize and to remember.

By Theorem 2, uniformly selecting one of 32 characters provides \( \lg 32 = 5 \) bits of assurance against a blind attacker. In general, \( N \) such characters provide \( N \lg N \) bits of assurance.

But what if the attacker is not blind, but rather uses imperfect OCR? We will refer to information gained from such OCR as a \textit{hint}. If an attacker has the hint of reading \( K \) out of \( N \) of the characters, then we are left with \( 5(N-K) \) bits of assurance. So if the OCR guesses 6 of 8 characters correctly, the \( 2 \times 5 = 10 \) remaining bits of assurance means that an attacker can succeed with probability of \( 2^{-10} = 1/1024 \).

Assuming that an attacker can read precisely 6 out of 8 characters is simple to analyze but not particularly realistic. We hope that we will eventually be able to study OCR programs that associate a probability with every potential value of a character (Section A.7 briefly examines that topic). For the remainder of this paper, though, we will assume the simpler model that imperfect OCR recognizes each character with a fixed probability. Chellapilla, Larson, Simard and Czerwinski [2005, Section 3.1] use a similar model to characterize their (very effective) attacks on commercial CAPTCHAs. We will assume that when the OCR identifies a character correctly, it knows that it has done so. When it cannot identify the character, it fails with no partial knowledge about its value. This lemma characterizes the probability of success for an attacker using this simple model.
Lemma 3. If an attacker can correctly identify a challenge letter with probability $p$, and with probability $q = 1 - p$ must uniformly guess from an alphabet of size $A$, then his probability of successfully identifying a letter is $p + q/A$.

Proof. The first term reflects the probability of identifying the letter outright, and the second term multiplies the probability of having to guess by the probability of success by guessing. QED.

Let’s suppose that we wish to achieve 20 bits of assurance against an attacker with OCR capable of recognizing half the letters, on the average. Since 4 letters suffice to achieve 20 bits of assurance against a blind attacker, we might (erroneously) assume that 8 characters would suffice against an attacker who can read each letter with probability $1/2$. But by Lemma 3, we know that the probability of success on any letter is $p + q/A$, or $1/2 + 1/64 ~ 0.5156$, so each letter gives $-\lg 33/64 ~ 0.956$ bits of assurance. We therefore require 21 such characters to achieve 20 bits of assurance.

We will refer to a successful attack of this form as a “lucky long shot”. If OCR recognizes each character with probability $p$, then the “long shot” of recognizing all $N$ characters occurs with probability $p^N$. If the OCR recognizes each character with probability $p = 1/8$, then each character in the challenge string provides about 3 bits of assurance against a lucky long shot.

Some visual CAPTCHAs use challenges that are random strings of decimal digits; this is particularly appropriate when the input device is a telephone keyboard or voice recognition. Each decimal digit gives $\lg 10 \sim 3.32$ bits of assurance. Our examples throughout the rest of this paper will use the 32 characters of this section, which give 5 bits of assurance. Employing all 26 upper and lower case letters, 10 digits and two other characters gives 64 characters and thereby 6 bits. The maximum number of keyboard characters is usually reckoned as 96, which gives an upper bound of 6.585 bits of assurance and numerous headaches (such as the space character and distinguishing commas from single quotes).

4 Dictionaries

All challenge strings are vulnerable to lucky long shot attacks, including strings chosen as words from a dictionary. But how do dictionary words fare against clever guessing and partial OCR?

For our first experiment, we obtained a copy of the EZ-Gimpy dictionary from openanonymity.sourceforge.net/HTML/scripts/ez-gimpy/dictionary

That dictionary contains 561 one-syllable English words, and a CAPTCHA selects one uniformly. By Theorem 2, a blind attacker has probability of success by guessing of $1/561 ~ 0.00178$, or 9.13 bits of assurance.

This a priori statement holds before the CAPTCHA program has randomly chosen its challenge word. After the program has chosen the word “right”, for instance, the attacker’s probability of success might be 0 (if the attacker has already chosen “wrong”) or 1 (if the attacker has already chosen “right”) or $1/2$ (if the attacker will soon uniformly choose between “right” and “wrong”).

Imperfect OCR comes in many varieties. Let us first suppose that the OCR program does not even attempt to recognize individual characters, but has segmentation adequate to count the letters in the challenge word. This attacker therefore has the
hint of knowing the length of the challenge word. The dictionary contains a total of 561 words; of those, 240 have 4 letters, 195 have 5 letters, and 126 have 6 letters. If the attacker knows only the hint of letter count, then he succeeds with probability

$$\frac{240}{561} \times \frac{1}{240} + \frac{195}{561} \times \frac{1}{195} + \frac{126}{561} \times \frac{1}{126} = \frac{3}{561}$$

The first term reflects the fact that the challenge word has 4 letters with probability 240/561, and the attacker succeeds in that case with probability 1/240 by guessing a uniformly selected word. Thus if the attacker is able to count letters, his probability of success becomes 3/561 \(\approx 0.00535\), and the assurance drops by \(\lg 3 \approx 1.59\) bits from 9.13 to 7.55.

We can generalize that analysis to a theorem that is useful for analyzing many attacks that exploit context.

**Theorem 4.** Suppose that a CAPTCHA chooses a challenge uniformly from a universe of size \(N\), and a hint partitions that universe into \(K\) non-empty equivalence classes. Then a guessing attacker who exploits precisely that hint can guess the challenge with \(a \text{ priori}\) probability \(K/N\).

**Proof.** Suppose that the \(K\) classes have sizes \(N_1, N_2, \ldots, N_k\), where \(N_i > 0\) and \(N_1 + N_2 + \ldots + N_K = N\). Because the challenge is chosen uniformly, the probability that it is from class \(I\) is \(N_i/N\). If the challenge is in class \(I\), then a random guess from that class succeeds with probability \(1/N_i\). The overall probability of success is therefore

$$\sum_{1 \leq i \leq K} N_i/N \times 1/N_i = \sum_{1 \leq i \leq K} 1/N = K/N$$

QED.

Thus a hint that partitions the challenge universe into \(K\) classes subtracts \(\lg K\) bits of assurance. Suppose, for instance, that the dictionary contained just one fewer word of each of length 4, 5 and 6, and one additional word of each of length 7, 8 and 9; that would double the probability of success by a letter-counting attacker!

Let's next assume that the attacker can identify only the first letter in the challenge word. This dictionary uses 24 of the 26 possible first letters (no words begin with either x or z), so that gives 24 equivalence classes and a probability of success of 24/561 = 0.0428 for 4.55 bits of assurance. Knowing the first letter roughly halves the assurance. This table summarizes the interaction of the hints of first letter, last letter, and word length:

<table>
<thead>
<tr>
<th>Hints</th>
<th>Partitions</th>
<th>Probability</th>
<th>Bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>1</td>
<td>0.0018</td>
<td>9.132</td>
</tr>
<tr>
<td>First</td>
<td>24</td>
<td>0.0428</td>
<td>4.547</td>
</tr>
<tr>
<td>Last</td>
<td>19</td>
<td>0.0339</td>
<td>4.884</td>
</tr>
<tr>
<td>Length</td>
<td>3</td>
<td>0.0053</td>
<td>7.547</td>
</tr>
<tr>
<td>First + Last</td>
<td>205</td>
<td>0.3654</td>
<td>1.452</td>
</tr>
<tr>
<td>First + Length</td>
<td>67</td>
<td>0.1194</td>
<td>3.066</td>
</tr>
<tr>
<td>Length + Last</td>
<td>51</td>
<td>0.0909</td>
<td>3.459</td>
</tr>
<tr>
<td>First + Last + Length</td>
<td>331</td>
<td>0.5900</td>
<td>0.761</td>
</tr>
</tbody>
</table>
The last letter is powerful, the first is more so, and length helps a little. While the roughly 9 bits of assurance in the original 561 words is not too strong to begin with, the CAPTCHA becomes surprisingly vulnerable with a small amount of additional information. Knowing just the first and last letter of the word, for instance, gives success in more than one attack out of every three.

It is important to remember that these \textit{a priori} probabilities apply before the challenge word is uniformly selected. Knowing the first letter, last letter and length of the word partitions the dictionary into 331 classes and gives a probability of success of almost 60 percent. If the uniformly selected challenge happens to be “ready”, it is the only word in its class and the attacker guesses it with probability 1. If the uniformly chosen challenge is “store”, then there are 14 words in its class (the largest such class), and the attacker’s probability of success is 1/14. (The second largest class consists of the 7 5-letter words that begin with p and end with e.)

We computed the number of character classes with variants of this Awk program:

```awk
{ s = substr($1,1,1) length($1) \
  substr($1, length($1), 1) 
  c[s] = 1 } END { for (i in c) cnt++ 
  print cnt }
```

The statements in the first pair of brackets are executed for each line in the dictionary, which contains one word per line. The first assignment creates a signature by concatenating the first character of the word, its length, and its last character, so the signature of “ready” is “r5y”. The second assignment stores that signature as an index in an associative array. The statements in the second bracket pair are executed after the entire dictionary has been read; the first statement iterates over the associative array to count its elements, and the second statement prints that count. The resulting program has optimal run time and space in both a Random Access Model and a Decision Tree Model of computation.

We will turn now to a larger dictionary. We started with an old copy of the Unix file `/usr/dict/words`, which contains 72,275 English words. We removed all words that contained non-alphabetic characters (such as ABC’s), converted upper case letters to lower case, and removed words that were thereby duplicated (the name Sue and the verb sue, for instance). The result was a list of 71,661 English words. We then removed the 3622 words with four or fewer characters as being too short for CAPTCHAs and the 6459 words with 13 or more characters as being too long; short words might be easy to recognize, and long words are certainly hard for users to type. The result was a list of 61,580 words.

Choosing one of those words uniformly as a challenge gives 15.91 bits of assurance. The lengths of the words are shown in this table.

<table>
<thead>
<tr>
<th>Length</th>
<th>Words</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>4879</td>
</tr>
<tr>
<td>6</td>
<td>7813</td>
</tr>
<tr>
<td>7</td>
<td>9623</td>
</tr>
<tr>
<td>8</td>
<td>10370</td>
</tr>
<tr>
<td>9</td>
<td>9976</td>
</tr>
<tr>
<td>10</td>
<td>8275</td>
</tr>
<tr>
<td>11</td>
<td>6262</td>
</tr>
<tr>
<td>12</td>
<td>4382</td>
</tr>
</tbody>
</table>

The average word length is 8.37 characters. We repeated the above experiments on this larger dictionary. The first 8 lines report those results.

<table>
<thead>
<tr>
<th>Hints</th>
<th>Partitions</th>
<th>Probability</th>
<th>Bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>1</td>
<td>0.00016</td>
<td>15.91</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-------------------</td>
<td>------</td>
<td>------------</td>
<td>--------</td>
</tr>
<tr>
<td>First</td>
<td>26</td>
<td>0.000422</td>
<td>11.210</td>
</tr>
<tr>
<td>Last</td>
<td>25</td>
<td>0.000406</td>
<td>11.266</td>
</tr>
<tr>
<td>Length</td>
<td>8</td>
<td>0.00130</td>
<td>12.910</td>
</tr>
<tr>
<td>First + Last</td>
<td>558</td>
<td>0.009061</td>
<td>6.786</td>
</tr>
<tr>
<td>First + Length</td>
<td>208</td>
<td>0.003378</td>
<td>8.210</td>
</tr>
<tr>
<td>Length + Last</td>
<td>183</td>
<td>0.002972</td>
<td>8.394</td>
</tr>
<tr>
<td>First + Last + Length</td>
<td>3347</td>
<td>0.054352</td>
<td>4.202</td>
</tr>
<tr>
<td>First + Mid + Last + Length</td>
<td>24,494</td>
<td>0.397759</td>
<td>1.330</td>
</tr>
</tbody>
</table>

In the second-to-last line, the attacker knows the first letter, last letter and length of the word. That partitions the dictionary into 3347 classes, so the probability of success is 0.054, or 4.2 bits of assurance. In the final line, the attacker also knows the middle character of the word, which gives success on 2 tries out of 5, for just 1.3 bits of assurance. Even large dictionaries succumb to hints about just a few characters.

For a final experiment, we obtained three files of common names from the U.S. Census Bureau at [www.census.gov/genealogy/names/](http://www.census.gov/genealogy/names/). The first file contained 88,799 last (family) names, the second contained 4275 female first names, and the third contained 1219 male first names. With no hints, the name lists contained 16.44, 12.06, and 10.25 bits of assurance, respectively. With the three hints of first letter, last letter and length, the bits of assurance dropped substantially to 4.26, 2.21 and 0.86, respectively. Adding the fourth hint of the middle character reduced the bits of assurance to 1.73, 0.89, and 0.27. That is, given those four hints, an attacker can guess a male first name with probability 0.83. Thus name lists seem just as vulnerable as the word lists, if not more so.

### 5 Markov Text

Chew and Baird [2003] suggest using Markov text as CAPTCHA challenges. Appendix A gives details on the experimental and theoretical analysis of Markov text for this purpose. This section summarizes the primary results in that appendix.

Section A.1 describes the training text (the King James Bible) and the algorithm used to generate challenges. Here are some sample 7-letter challenges:

```
famini tgseyi werean cecrent opletih atobre makefor ibeasst ohiare
```

These are not English words, and therefore do not yield to a dictionary attack. Yet they are vaguely familiar and often even pronounceable.

Like all challenges, Markov strings are vulnerable to lucky long shot attacks. Section A.2 estimates how vulnerable Markov text is to a global blind attacker who guesses a string using various strategies. Markov strings provide 4.70 bits of assurance per character against an attacker who uniformly guesses one of 26 letters, 3.86 bits against an attacker who guesses letters with the appropriate frequencies, and 2.97 bits against an attacker who guesses common letters. An engineering approximation and experiment together indicate that the average character gives about 2.07 bits of assurance against an attacker who guesses a string produced by the same Markov model.

Section A.3 describes how some Markov strings occur much more frequently than others, and are therefore vulnerable to more clever blind attackers. The most common string among 100,000 random 5-character challenges was “andth” (due in large part to the common word pair “and the”); it occurred 341 times. That observation reduces
the number of bits of assurance per character from 2.07 to just 1.64. Other attacks yield similar reductions in assurance.

One approach to the problem of challenges that have nonuniform probabilities is to give variable assurance. Section A.4 tells how steps in that direction lead to dead ends.

A second approach to the problem of nonuniform probabilities is to “groom” the objects to induce uniformity. Section A.5 proposes two different ways of grooming Markov text to thwart guessing attackers. Here is text generated by “limited successor grooming” where the subsequent character is randomly selected from among the 8 most likely following letters:

hitityamsis cumasottea clonginiev oiwofhimou gillbyhone ifuganarin

Each character in those strings carries almost 3 bits of assurance against certain attacks.

Section A.6 shows that ungroomed Markov text is vulnerable to contextual attacks. If an attacker can read the first two letters of a trigram, he is able to guess the next letter with probability about 0.375, which corresponds to just 1.414 bits of assurance. Groomed strings might be less vulnerable to some contextual attacks.

Appendix B describes experiments on Markov text generated using a training sample of dictionary words, rather than running text. Here is a random selection of 8-letter challenges made by that process:

pansonet andobhonz celtoundo centhous sumencri suserp uroce

Experiments in that appendix indicate that such “dictionary Markov text” often provides more assurance than Biblical Markov text. That section also examines training on lists of names.

6 CVC Syllables

Dictionary words are easy to read and to remember but are also easy to guess; random strings are hard to guess but are also hard to read and to remember. Markov text provides a tradeoff between those two extremes. In this section, we will investigate another intermediate point on that spectrum: “consonant-vowel-consonant” or “CVC” syllables. Because they contain three letters, we will sometimes refer to them as “CVC trigrams”.

Ebbinghaus [1885] introduced CVC trigrams in his pioneering study of human memory that commenced in 1879; he referred to them as “nonsense syllables”. In their simplest English form, we classify the 26 letters of the Roman alphabet as the five vowels a, e, i, o, and u, while the remaining 21 letters are consonants. We then generate a trigram as a consonant-vowel-consonant or CVC trigram, such as het, som, zih, qox, and cat. We assume that a CAPTCHA challenge would consist of either two or three CVC triples, such as hetsom or zihqoxcat.

One can be quite elaborate in forming CVC trigrams. Ebbinghaus [1885, Chapter III, Section 11, Footnote 1] writes:

The vowel sounds employed were a, e, i, o, u, ã, õ, ù, au, ei, eu. For the beginning of the syllables the following consonants were employed: b, d, f, g, h, j, k, l, m, n, p, r, s, (= sz), t, w and in addition ch, sch, soft s, and the French j (19 altogether); for the end of the syllables f, k, l, m, n, p, r, s, (= sz) t, ch, sch
He discarded three-letter German words to avoid mnemonic associations that do not clutter true nonsense syllables; he was left with about 2300 syllables. (Ebbinghaus uses “I” in the last sentence because he was the only subject in his experiments.)

As a brief step in the direction of well-considered CVC trigrams, we evaluated CVC trigrams assembled from the following components:

- Initial Consonant: b c d f g h j k l m n p r s t v w y z
- Vowel: a e i o u
- Final Consonant: b c d f g j k l m n p r s t v x z

We employed the five standard vowels, but excluded some consonants using the "looks funny" test. As initial consonants, we excluded q and x; q looks strange without a following u and vowel, and x looks strange at the start of a syllable (we apologize for our xenophobia). We similarly ignored h, q, w and y as final consonants; q still looks strange without the following u, and ih, iw and iy are all rare digrams. We believe that this simple policy results in plausible CVC syllables. It yields pairs such as

sixbeg pembus letsv dégyov dedvox nefbak kojnud zunzib tafbis

and triples such as

mabjertot jaylabcav lidfaggas mazhilcoz nubnabcux pifgoxkam lirbagius

Each of those sequences was produced by a program, and selected without human editing.

We assume that CVCs used as CAPTCHA challenges would be employed as pairs or triples. How much assurance is in such a challenge against a blind attacker? A uniform selection from 19 initial consonants provides 4.25 bits of assurance, 5 vowels provide 2.32 bits, and 17 final consonants provide 4.09 bits, for a total of 10.66 bits per CVC trigram, and an average of 3.55 bits per character. A CVC pair thus provides 21.31 bits, and a CVC triple provides 31.97 bits of assurance. Thus a CVC triple of 9 letters provides a tad more assurance than 6 characters of the random strings chosen from a 32-symbol alphabet that we saw in Section 3:

3frem 7j6hwu ldsu5b dwazdr jx62z7 7n62lg 9lyy57y dgtycs gjwqbq

We conjecture that even though the CVC triples are 50% longer, the average user will find the CVC triples easier to read, easier to remember (for non-touch typists), and more "friendly".

CVC challenges, like all strings, succumb to lucky long shot attacks. But how do they fare against imperfect OCR? If an attacker knows that a string is generated in this fashion, and recognizes some of the characters, we can still add the assurance left in the unrecognized characters: 4.25 bits for an initial consonant, 2.32 bits for a vowel, and 4.09 bits for a final consonant, or 3.55 bits on the average. Since the CVC letters were generated uniformly and independently, knowledge of the surrounding characters does not aid an attacker.

Some people find CVC doubles and triples to be awkward impersonators of English words. A friend who wishes to preserve his anonymity once generated random pronounceable passwords by combining 4 CV pairs based on Japanese syllables, like
“yokohama”. With the same (non-Japanese) initial consonants and vowels as described earlier, his scheme generates words like:

\[
pahimotucusefutuvicebagiehevurouhuzedaravufelatotosiruka
\]

As before, a selection from 19 consonants provides 4.25 bits of assurance against a blind attacker, and a selection of 5 vowels provides 2.32 bits, for a total of 6.57 bits for the digram, and an average of 3.28 bits per character, a slight decrease from the 3.55 bits per CVC character.

7 Evaluating a Challenge

How should a CAPTCHA designer select a challenge? Is it better to use a one-syllable word from the EZ-Gimpy dictionary or twelve randomly chosen alphanumeric characters? This optimization problem has many variables. Section 7.1 examines dimensions dealing with assurance, Section 7.2 examines psychological dimensions, and Section 7.3 brings the issues together.

7.1 Assurance Against Various Attacks

We have considered three fundamentally different forms of attack. Global attacks use guessing but no OCR, while lucky long shot attacks use OCR but no guessing. Contextual attacks combine both OCR and guessing.

**Global Attacks.** Dictionaries and (ungroomed) Markov text are both subject to “blind” attacks that use no OCR. If a blind attacker knows that the challenge is chosen from a dictionary, Theorem 2 quantifies how guessing a word is more effective than guessing a sequence of random letters. Similarly, if a blind attacker knows that the word is generated from (ungroomed) Markov text, Section A.3 shows how he gains advantage by guessing a common string.

**Lucky Long Shot Attacks.** We use this term to characterize a guesser who uses imperfect OCR that recognizes a fraction of the letters. If the OCR happens to recognize all letters, then the attack succeeds. If the OCR recognizes each letter with probability \(p\), then it recognizes all \(N\) letters with probability \(p^N\). One way to defend against this attack is by decreasing \(p\) (presumably by further degrading the image); the other way is to lengthen the challenge. Lemma 3 allows us to evaluate an attacker who combines imperfect OCR with uniform guesses about unrecognized characters.

**Contextual Attacks.** These attacks combine partial OCR and clever guessing: the attacker knows the value of some characters, and guesses the remaining characters. Random strings have no context, and therefore give attackers no information. CVC syllables have only a positional context: the attacker knows the set of characters valid in each position. CAPTCHA researchers have long realized that dictionaries are vulnerable to contextual attacks, and the tables in Section 4 derived using Theorem 4 quantify that vulnerability. Section A.6 shows that Markov text is vulnerable to contextual attacks.
7.2 Psychological Issues

These issues are beyond the scope of this mathematical paper, but we survey them briefly for completeness. Card, Moran and Newell [1983] provide an overview of this area.

Readability. Chew and Baird [2003, Section 3.2] cite psychophysical studies that show that linguistic context is helpful for human readers trying to recognize characters.

Memorability. A touch typist might enter the characters of a challenge without his eyes leaving the CAPTCHA image. A hunt-and-peck typist, though, must move his eyes from the image to the keyboard in order to enter the challenge. Card, Moran and Newell [1983, Section 2.1] describe “the model human processor” involved in this task. Their example on page 36 is directly relevant to the CAPTCHA task. They allege that the “sequence of nine letters below is beyond the ability of most people to repeat back”:

B C S B M I C R A

Try it and see how you fare. They then offer this list, which differs only slightly

C B S I B M R C A

and observe that “the average American college sophomore” [of 1983] will “chunk” it into the three easily recalled components CBS, IBM and RCA. For similar reasons, we suspect that dictionary words, Markov text and CVC syllables are all more easily remembered than random alphanumeric strings.

Intimidation. Many users are intimidated by CAPTCHAs. Much of the discomfort is generated by having to take a test and by the visual degradations. We conjecture that intimidation is also a function of the challenge string, and that it (roughly) increases among this sequence of strings

hi cozy cwms death sixbeg i5e2 perspicacity dandhe pifgoxxam shk3aj7

Quantifying this intuition remains an important open psychological problem.

7.3 Selecting a Challenge

We will now take a more complete look at the four types of CAPTCHA challenges that we have studied. We intend the comments below not as a complete answer for all applications, but rather as a starting point for particular engineering decisions.

Dictionary Words. These are psychologically comfortable, but vulnerable to both global and contextual attacks. They should probably not be used when much assurance is required.

Markov Text. Such challenges provide much of the comfort of real words, and are not quite as vulnerable. In our experiments, each character carried about 2.07 bits of assurance against guesses from the same model. Analyses in Appendix A show substantial vulnerabilities to both global (1.6 bits of assurance per character) and contextual attacks (1.4 bits per character); we believe that additional analyses (especially taking advantage of additional information about the value of unrecognized characters) will show that Markov text is even more vulnerable. Such challenges should therefore probably not be used when much assurance is required. If Markov text is employed, the grooming sketched in Section A.5 should be seriously considered. The dictionary
Markov text described in Appendix B also appears to improve both assurance and psychological friendliness.

**Random Alphanumeric Characters.** Such challenges offer maximum assurance; they have minimal vulnerability to global and contextual attacks. Short strings are vulnerable to lucky long shot attacks. Unfortunately, these are not psychologically friendly.

**CVC Syllables.** Although we admit to a bias, we believe that these are strong all-around contenders. Each CVC character achieves about 2/3 the assurance of a random character, so 9 CVC characters (three CVC syllables) have about the same assurance as 6 random characters against both global and contextual attacks. Those 9 characters will be more protected against a lucky long shot attack, and will almost certainly be easier to memorize. We believe that they will have other psychological advantages as well.

### 8 Conclusions

This paper has described a theory of assurance and used it to analyze several known types of CAPTCHA challenges. It also introduced two new types of challenges (the CVC syllables of Section 6 and the “dictionary Markov text” of Appendix B) and used “grooming” to decrease the vulnerability of Markov text. Section 7 shows how this information might be useful in choosing a challenge to deploy in a production-quality CAPTCHA.

Almost all of the results in this paper immediately apply to “automated password generators” such as the Linux “apg” program. NIST [1993] provides a Federal Information Processing Standard for programs that produce passwords from Markov text after training on dictionaries. Van Vleck [2005] gives Java and C implementations and a brief history of such programs: Daniel Edwards wrote a digram-based password generator for the MIT CTSS timesharing program in 1965. A web implementation can be found at www.winguides.com/security/password.php. NIST [1993, Section 2.4] characterizes the security of the generated passwords in these words:

> Although numbers and special characters are not permitted, the password space, which is a function of the number of characters in the password, is very large. Approximately 18 million 6-character, 5.7 billion 8-character, and 1.6 trillion 10-character passwords can be created by the program. Users should select a password space commensurate with the level of security required for the information being protected.

Each of the statements about size of space is just as true as it is misleading. Each statement implies that a character carries slightly over 4 bits of assurance. Our analyses of Markov text in Appendix B.1 confirm that number for a letter-guessing attacker. However, the assurance drops to about 2.49 bits per character against an attacker who generates random Markov text, and could drop to 2.14 bits per character against more clever attackers. Grooming could certainly help to increase that assurance. Our analyses of partial OCR apply to an attacker who learns information about a subset of the letters, perhaps by watching a user type a password or by partially intercepting a transmitted password. Our analyses also apply to passwords generated as random strings, dictionary words and CVC trigrams.
Many problems remain open. Quantifying the psychological issues sketched in Section 7.2 is high on that list.

This paper has applied the theory of assurance to the textual challenge strings used in the most common class of CAPTCHAs. Baird and Bentley [2005] apply a preliminary version of this theory to “Implicit CAPTCHAs” in which the user clicks on a navigation aid, potentially without even realizing that he has thereby passed a test. They make statements of the form that a simple test “provides 5 bits of assurance against a blind attacker, or 3 bits of assurance against an attacker that can group words”. Rui, Lie, Kallin, Janke and Paya [2005, p. 54] report that a facial recognition CAPTCHA based on many mouse clicks “is robust at a rate of 2 out of a million”. We would hope that they might now rephrase that as “it provides 19 bits of assurance” against a specified class of attacker. We hope that this theory of assurance can broadly applied to CAPTCHAs and in the larger field of Human Interactive Proofs (HIPs).

Three-letter CVC syllables are among the simplest models of random English text. One could use a more complex model involving consonant sequences and vowel pairs to generate random syllables such as “splaint”. Starting with such a base, one could use affix analysis to generate random challenge words such as “desplainify”. Dan Bentley of Google has suggested increasing the effective size of a dictionary by intentionally introducing misspellings, so the word “suggest” might render challenge strings such as “soggest”, “sugest”, “sojast”, and many others. Much of the research on spelling correction might be inverted to transform a “spell checker” into a “mispelling generator”. Our simple CVC syllables only scratch the surface of word models beyond Markov text.

It is tempting to generate challenge strings that contain both upper- and lower-case letters, and check to ensure that the respondent capitalizes only the selected letters. This could potentially add one bit of assurance per letter. Unfortunately, many humans find case distinctions awkward, especially in a context like a CAPTCHA that is seen only rarely. Furthermore, while it is straightforward to distinguish between a and A, the nine upper-case letters C, K, O, S, U, V, W, X and Z can look like their lower-case counterparts, depending on font and whether a CAPTCHA varies the size of individual letters. The most promising path therefore appears to be to stochastically capitalize only the first letter of a word (because that is most natural to readers of English), and perhaps for the 17 letters that do not have similar forms. This idea merits investigation.

Section A.5 describes two ways in which Markov text might be groomed. Preliminary experiments indicate that the methods might be effective. That area is ripe for further work.

We spoke of letter segmentation and character recognition as orthogonal tasks, but in fact they can be related. If an attacker knows that a challenge is generated as Biblical Markov text, and he can’t decide whether an area represents the single letter w or the digram vv, then it can be useful to know that the digram vv never occurs in the training text. On the other hand, if the challenge is from a dictionary, then vv occurs only rarely, in words such as savvy and skinny. This example shows how contextual information can aid segmentation as well as character identification.

Throughout this paper we have used a very simple model of imperfect OCR: the process succeeds (and knows so) with probability p, or fails (and gathers no additional information) with probability 1−p. It would be interesting and useful to analyze contextual attacks in which OCR gives a set of probabilities for various characters. Such
a model will open the door for the contextual attacks based on Bayesian analyses sketched in Section A.7.

With the exception of uniform alphanumeric challenges and the CVC syllables of Ebbinghaus, most of the work that we have described applies predominantly to English. How are these results related to the haphazard nature of English spelling? French spelling is much more regular; it is often said that if a native speaker of French can pronounce a word, then he can spell it. What are the implications of that for Markov text or nonsense syllables in French? How robust is a French dictionary to an attack with partial OCR? Apart from issues of transforming other alphabets, how should CAPTCHA challenges for various languages be implemented in the Latin alphabet? Random Romaji syllables for Japanese users should consist of consonant-vowel pairs from a well-known table of kana (different from the CV pairs at the end of Section 4.3), perhaps with a final n. Random words for Arabic and Hebrew speakers might exploit the triconsonantal structure of those Semitic languages. Analyzing the assurance of internationalized CAPTCHA challenges is an interesting open problem.

Acknowledgments

The authors are grateful for the helpful feedback of many of the participants at the HIP 2005 Workshop, especially Henry Baird, Andrei Broder, Kumar Chellapilla, Dan Lopresti, and Patrice Simard. For helpful comments on this paper, we are grateful to Henry Baird, Dan Bentley, and Brian Kernighan.

Appendix A Details on Markov Text

This appendix studies in detail challenges that are generated as order-2 Markov text. The results are of this appendix are summarized in Section 5.

A.1 The Training Text and The Markov Model

Markov text is described by Shannon [1948, Sections 2-4]. Our experiments use the King James Bible as training text. We chose that text because it is widely available, has a vocabulary and style familiar to many readers of English, yet has not (as far as we know) been used to generate challenges for any commercial CAPTCHAs; we do not wish to assist attackers. We stripped book, chapter and verse markings from the text, removed all non-alphabetic characters (including spaces), and converted upper case letters to lower case. The result was a string of $N=3,222,345$ lower-case letters, and therefore $N–2$ trigrams.

Table 1. Statistics on digrams from the King James Bible

<table>
<thead>
<tr>
<th>Digram</th>
<th>Common Successors</th>
<th>Other</th>
</tr>
</thead>
<tbody>
<tr>
<td>th</td>
<td>5.13</td>
<td></td>
</tr>
<tr>
<td>e</td>
<td>62.7</td>
<td></td>
</tr>
<tr>
<td>a</td>
<td>11.1</td>
<td></td>
</tr>
<tr>
<td>i</td>
<td>5.7</td>
<td></td>
</tr>
<tr>
<td>o</td>
<td>5.4</td>
<td></td>
</tr>
<tr>
<td>y</td>
<td>3.3</td>
<td>11.8</td>
</tr>
<tr>
<td>he</td>
<td>4.04</td>
<td>r</td>
</tr>
<tr>
<td>-----</td>
<td>------</td>
<td>-----</td>
</tr>
<tr>
<td>an</td>
<td>2.38</td>
<td>d</td>
</tr>
<tr>
<td>nd</td>
<td>2.02</td>
<td>t</td>
</tr>
<tr>
<td>ha</td>
<td>1.56</td>
<td>t</td>
</tr>
</tbody>
</table>

Table 1 shows the five most common digrams and the five most common letters following each. The first line says that 5.13 percent of the trigrams begin with the digram “th” (that is, 165,246 of the 3,222,343 trigrams), and of those digrams, 62.7% are followed by the letter e, 11.1% are followed by a, and so on. The final column says that 11.8% of the time, the digram is followed by one of 20 other letters (the letter x is the only letter that never follows “th”).

The main data structure in one algorithm to generate Markov text is the complete version of the fragmentary Table 1. The algorithm starts with an arbitrary digram. At each state, it chooses the next character according to the probabilities from the structure. That new character is printed, and becomes the second letter in the new digram. Every specified number of letters, we print a newline to form challenge words of a fixed length.

This data structure makes it easy to compute the probability of generating a particular string as it is computed. Let us suppose, for illustration, that with probability 1 we start with the digram “th”. We initially assign p the value 1, run the process, and multiply p by the probability of the branch taken at every step. If the process chooses the next letter as e, then the string “the” results with probability 0.627, and the current digram becomes “he”. If the random process chooses the successor m (with probability 0.083), then the result is the string “them”, and the probability p becomes 0.0520. We will find such probabilities a handy tool in later analyses.

A.2 Evaluating Assurance

How much assurance do we gain if a CAPTCHA subject correctly identifies a Markov challenge? If the attacker notices that the challenges are all letters, and therefore guesses a letter uniformly, then each letter in a challenge carries $\lg 26 \approx 4.70$ bits of assurance against that attacker. But what if the attacker observes the letter frequencies? For this Bible, the most frequent letter is e at 12.73%, the second most frequent is t at 9.81%, and the least frequent is x at 0.04%. Bentley and Mallows [2005, Section 4.1] prove the following.

**Lemma 5.** Suppose that a CAPTCHA chooses a challenge according to a probability vector $p = (p_1, p_2, \ldots, p_N)$, and that the attacker also chooses a string according to $p$. Then the attacker’s probability of success is the mean probability $\Sigma_i (p_i)^2$.

**Proof.** Each term represents the probability of the CAPTCHA choosing that value times the probability of the attacker also choosing it. QED.

If we apply Lemma 5 to the letter frequencies in this Bible, we find that this attacker’s probability of success on any letter is 0.0688, which corresponds to 3.86 bits of assurance.
Bentley and Mallows [2005, Section 4.1] also consider the case of an attacker who knows the values of the probability vector \( p \). Rather than choosing a random element of that vector, a clever attacker should choose the most likely element.

**Lemma 6.** Suppose that a CAPTCHA chooses a challenge according to a probability vector \( p = (p_1, p_2, ..., p_N) \) in which the probabilities are ordered \( p_1 \geq p_2 \geq \ldots \geq p_N \). An attacker maximizes his probability of success by selecting one of the most likely elements, and the probability of success is \( p_j \).

**Proof.** Selecting any element \( j \) with \( p_j < p_i \) decreases the probability of success. QED.

This lemma applies to guessing a single letter of Markov text: guessing the most likely letter \( e \) succeeds with probability 0.1273, which corresponds to 2.97 bits of assurance. But because successive letters of Markov text are not independent, this lemma does not apply to the more general problem. When presented with a 7-letter challenge, a foolish attacker might misapply the lemma to guess “eeeee”. But while \( e \) is a common letter, and “ee” is a relatively common digram (about 0.47% of the digrams), “eee” is a very rare trigram (it occurs only with Biblical word pairs such as “thee even”). In particular, the probability of the digram “ee” being followed by the letter \( e \) is just 0.27%, so the probability of the string “eeeeee” is roughly 0.0047\( \times 0.00275 \approx 6.7 \times 10^{-10} \), corresponding to about 50.4 bits of assurance.

A more reasonable assumption is that a blind attacker generates random challenges from the same Markov model. We can therefore calculate the probability of generating a string as sketched above. This table shows several seven-character strings in the order they were generated (so the starting digram for “tseywif” was the “gi” at the end of “famingi”) and the probability of computing each string.

<table>
<thead>
<tr>
<th>Challenge</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>famingi</td>
<td>0.0000009547</td>
</tr>
<tr>
<td>tseywif</td>
<td>0.0000000010</td>
</tr>
<tr>
<td>erandav</td>
<td>0.0000056477</td>
</tr>
<tr>
<td>erecent</td>
<td>0.000019876</td>
</tr>
</tbody>
</table>

On the average, how many bits of assurance does each Markov character yield? We ran several small experiments in which we generated many characters, and computed the mean probability of generating each character. The average tended to cluster around 0.238. Our final experiment generated ten million random characters, and computed the average probability of generation as 0.23785, which by Lemma 5 is the attacker’s probability of success, and corresponds to 2.0718 bits of assurance per character.

To calculate that number exactly, we could use basic principles of Markov chains to compute the steady-state probabilities of this chain. As an approximation, we instead assumed that the steady-state probability of each digram was its fraction in the training text. If we let \( d_i \) denote the fraction of the \( i \)th digram in the training set, and \( t_{ij} \) denote the probability of transitioning from digram \( i \) to character \( j \), then we approximate the average probability by generalizing Lemma 5 to

\[
\sum d_i \sum (t_{ij})^2
\]

Using that approximation, the expected probability of a letter is 0.23772, which corresponds to 2.0723 bits of assurance. Because that approximation is close to the experimental value, assigning each Markov character roughly 2.07 bits of assurance seems a
plausible engineering approximation. A seven-character Markov challenge therefore gives roughly 14.5 bits of assurance against an attacker who generates a random string from this Markov model.

A.3 Global Attacks

We will assume in this section that each Markov challenge is of a fixed length and is assigned the same assurance, about 2.07 bits per character. A blind attacker who has no additional information about the particular challenge can exploit the fact that some Markov strings occur much more frequently than others. To explore such an attack, we generated 100,000 Markov strings, each 5 characters in length. We then viewed the first character of each word as a one-character word, the first two characters of each word as a two-character word, and so on through all five characters in the word. This table shows the number of distinct words of the various lengths:

<table>
<thead>
<tr>
<th>Length</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Distinct Words</td>
<td>26</td>
<td>518</td>
<td>4707</td>
<td>22306</td>
<td>49947</td>
</tr>
</tbody>
</table>

The next table shows the five most common words and the least common word out of the 100,000 total words of each length:

```
<table>
<thead>
<tr>
<th>e</th>
<th>12598</th>
<th>th</th>
<th>5091</th>
<th>the</th>
<th>2979</th>
<th>ther</th>
<th>435</th>
<th>andth</th>
<th>341</th>
</tr>
</thead>
<tbody>
<tr>
<td>t</td>
<td>9846</td>
<td>he</td>
<td>3926</td>
<td>and</td>
<td>1771</td>
<td>ethe</td>
<td>435</td>
<td>ndthe</td>
<td>204</td>
</tr>
<tr>
<td>h</td>
<td>8964</td>
<td>an</td>
<td>2361</td>
<td>tha</td>
<td>716</td>
<td>andt</td>
<td>426</td>
<td>ofhe</td>
<td>201</td>
</tr>
<tr>
<td>o</td>
<td>7353</td>
<td>nd</td>
<td>1950</td>
<td>eth</td>
<td>715</td>
<td>ndth</td>
<td>360</td>
<td>there</td>
<td>121</td>
</tr>
<tr>
<td>n</td>
<td>6977</td>
<td>ha</td>
<td>1722</td>
<td>hat</td>
<td>563</td>
<td>ofth</td>
<td>340</td>
<td>thall</td>
<td>108</td>
</tr>
<tr>
<td>q</td>
<td>16</td>
<td>bg</td>
<td>1</td>
<td>aav</td>
<td>1</td>
<td>aaco</td>
<td>1</td>
<td>aaceb</td>
<td>1</td>
</tr>
</tbody>
</table>
```

If a trainee is presented a 5-character challenge from this training text, his optimal strategy is to guess “andth”, and he achieves success (according to this limited experiment) with probability 0.00341, which corresponds to just 8.2 bits of assurance, which is below the expected 10.35. As a larger test, we generated one million Markov strings of length 7. The most common string was “andther”, which occurred 284 times. Eight other strings occurred more than 200 times; in decreasing order of frequency, they were: “sandthe”, “themand”, “mandthe”, “eandthe”, “dandthe”, “oughthe”, “andthes” and “handthe”.

The attacks have so far identified common challenges by generating many Markov strings and observing which ones occur frequently. An alternative attack starts with the most common digram (“th”), and at each stage always takes the most common next step in the Markov chain. For the King James training set, this results in an infinite cycle of “thereandthereand”. As before, we interpret the probability at each branch as giving the corresponding number of bits of assurance. For seven-letter strings, the assurance varies from 10.25 bits for “ndthere” to 12.90 bits for “ereandt”. Notice that 12 bits of assurance would yield about 250 instances in every million challenges, which is consistent with the observed values mentioned in the preceding paragraph.
A.4 Variable Assurance

Fixed assurance for variable probabilities encourages attackers to guess likely challenges. One might therefore consider varying the assurance, and there are many ways to do so. Suppose that the CAPTCHA administrator knows that a potential attacker employs the strategy sketched in the previous section of guessing the most likely word, which, in this case, is “andth”. The administrator should therefore give no assurance for that answer because the probability of that attacker making that guess is $p=1$, and $-\lg p = 0$. But if the attacker knows that twist, then he will in turn guess the runner-up "ndthe", and so on and so on.

Another approach is to assign little assurance to common strings and much assurance to rare strings. One implementation is to delete some of the trailing characters in an uncommon string such as “tseywif”, but keep all the characters in a common string such as “erandav”. Unfortunately, this opens another door for an attacker: if the string is short, the attacker knows to generate rare trigrams, while if it is long, he knows to generate common trigrams.

Variable assurance can be natural in a context in which assurance is accumulated (such as providing several forms of identification to authenticate a user), but is usually unnatural for “single-step” CAPTCHAs.

We have considered many approaches to assigning variable assurance to nonuniform probabilities, and we have found serious flaws in each. The next section shows how to avoid the problem by avoiding nonuniform probabilities.

A.5 Grooming

Bentley and Mallows [2005] introduce a technique called “grooming” to induce uniformity into nonuniform phenomena. We can use that idea to remove the biggest targets from the sights of our attackers. We will call this “common-string grooming”. The sample of 100,000 random 5-character Markov strings in Section A.3 yielded 49,947 distinct words. Among those, 32,910 occur just a single time. We could therefore ignore all duplicated words, and present as challenges words such as

```
yheri warem ushy tieda snate paria onnmai gerth eciph ceint
```

This scheme might combine the best of both worlds: text that is vaguely recognizable yet still hard to guess. While five uniform lower-case letters yield $5 \times \lg 26 \sim 23.5$ bits of assurance, this scheme would yield $\lg 32,910 \sim 15.0$ bits of assurance (roughly 3 bits per character) and increased readability. We also studied the one million Markov strings of length 7 that we described earlier. Grooming that set leaves 624,284 unique strings. Since $\lg 624,284 \sim 19.25$, that yields roughly 2.75 bits of assurance per character.

Our next approach is “limited successor grooming”. This method yields 3 bits of assurance per character against a local attacker with OCR that has identified the two preceding characters. Among the letters following each digram, consider only the 8 most common, and choose one of those uniformly. That uniform choice from 8 characters gives $\lg 8 \sim 3$ bits of assurance. Similarly, choosing one of the 4 most common characters gives 2 bits of assurance per character. One might fear that while regular Markov text tends to look like English, groomed Markov text might not. For instance, grooming to 4 characters might look unnatural by cutting off later characters. Then
again, grooming to 8 characters might look unnatural by giving equal weight to later characters. The following 7-letter challenges were generated either as “pure” Markov text, or with grooming to 4 or 8 characters; can you identify which is which?

hecketu heretto hantara thimst edioes areopro
andundd thouse theaetd tledan onganci fivilie ontohl rtohins omentot butpunc phribut vadotti

The first and fourth columns are ungroomed, the second and fifth columns are groomed to 4 characters, and the third and sixth columns are groomed to 8 characters. We do not note any differences in “feel” among the three methods. In generating 10,000 characters groomed to 4, we always generated 2 bits of assurance per character. If a digram does not have 8 following characters, we uniformly select one of its successors. This process generated an average of 2.93 bits of assurance per character.

The table below repeats the experiment in Section A.3 with Markov text groomed to 8 successors. We generated 100,000 5-character strings, and viewed the first character of each as a 1-character string, the first two characters as a two-character string, and so forth. Here are the five most common strings and the least common string of each length.

| a 10064 | an 1257 | tal 176 | niq 63 | nique 24 |
| t 9858 | te 1249 | and 174 | avsh 46 | eavsh 22 |
| i 8637 | ta 1213 | thi 171 | thyp 36 | niqua 17 |
| e 8581 | th 1177 | tes 171 | thno 34 | iniq 17 |
| o 7875 | al 1168 | esh 166 | isro 34 | eniq 16 |
| z 41 | gy 1 | apt 1 | aack 1 | aaacap 1 |

In each length, the groomed strings are distributed more uniformly than their ungroomed counterparts. Among the groomed five-letter strings, only 14 occurred 10 or more times, while 993 strings ungroomed strings occurred that often. The most frequent groomed string (“niqie”) occurred 24 times, while its ungroomed counterpart (“andth”) occurred 341 times. Of the groomed 5-character strings, 37,237 were unique, while just 32,910 of the ungroomed strings were unique.

We have suggested two ways in which Markov text might be groomed, and preliminary experiments are optimistic. This area is ripe for further work.

A.6 Contextual Attacks

We saw in Section 4 how a partially sighted attacker with imperfect OCR can use contextual guessing. How does Markov text fare against contextual attacks?

Table 1 in Section A.1 reports key statistics on the five most common digrams and the letter that most frequently follows each digram. Suppose that a clever attacker has OCR that can read a digram in a Markov challenge, can’t read the following character, and has no other information. An obvious strategy is to guess the most likely following character. Such an attack will likely succeed on the digram “th”, because it is followed by e 62.7% of the time. The same attack is less likely to succeed on the digram “he”, because its most common successor is r at just 13.3%. Weighting all digrams by their frequency in the training text, we find that the average probability of success is 0.37516, so the next character carries, on the average, only \(-\log 0.37516 = 1.414\) bits of assurance. If imperfect OCR gives the leading characters in a string, we can use the
same strategy of following the most likely Markov branches to give both a likely string and an \textit{a posteriori} estimate of its probability.

Imperfect OCR might give attackers a variety of information, which they should exploit as far as possible. The most information is available when an unrecognized letter has two recognized letters on each side. (Because the challenge is generated as order-2 Markov text, letters further away make no difference.) Developing algorithms for additional cases remains an interesting open problem; Section A.7 sketches how a Bayesian attack might proceed.

An attack on Markov text constructed by “limited successor” grooming can exploit preceding characters to narrow the choice of the next character, but that character is eventually selected uniformly. Additional information might be gleaned from following characters. We suspect that “common-string” grooming provides some defense against contextual guessing so long as the attacker does not know the set of unique strings that were randomly generated. On the other hand, when the attacker knows the unique challenges, he can use the dictionary attacks described in Section 4. The dictionary of 32,910 unique 5-letter strings described in Section A.5 provides 15.0 bits of assurance against a blind attacker. Knowing the first character partitions the dictionary into 26 equivalence classes, and reduces the assurance to 10.3 bits; knowing the first two characters gives 514 equivalence classes and reduces the assurance to 6.0 bits. That is, an attacker who knows the first two characters of such a challenge can succeed with an \textit{a priori} probability of 0.0156.

A.7 Bayesian Attacks

Given a probability structure such as a Markov chain or a dictionary, one can derive the probability distribution of any character, given the other known characters. For an order-2 Markov model, for instance, the probability distribution for each character depends only on the immediately preceding digram of characters. To express this structure, we write $P(x | y)$ for the probability that a particular character is $x$, conditional on some information $y$ about other characters. These functions characterize the CAPTCHA’s generation mechanism.

We assume that the imperfect OCR produces for each character a probability distribution over the alphabet, representing its guess as to what the character might be. We write $Q(r | x)$ for the number that the OCR generates, representing its posterior probability that the character is $r$, when really it is $x$. So for each $x$, the sum $\Sigma_r Q(r | x)$ is unity. We assume that in computing these probabilities, the OCR assumes \textit{a priori} that all characters in the alphabet are equally likely.

Now suppose that the attacker $A$ has additional information from another source; perhaps he knows the number of characters, or is confident that he knows some of the characters. We write $y$ for this information. An application of Bayes’s theorem will then give $A$’s posterior probability that the character being read is $r$, when really it is $x$, and other information $y$ is known accurately by the attacker:

$$A(r | x, y) = \frac{P(x | y) Q(r | x)}{\Sigma_s P(x | y) Q(s | x)}$$
Appendix B  Markov Text from a Dictionary

Our experiments on Markov text in Appendix A follow the lead of Chew and Baird [2003] in using a body of “real” English text as a training sample. Chew and Baird trained on the million-word Brown Corpus, and we used the (roughly) 800,000-word King James Bible. The text generated from the resulting models therefore has many properties of written English, including its letter frequencies and digram and trigram frequencies. Unfortunately, we saw that such text tends to contain common words and even word pairs, and therefore huge targets for clever guessers such as “and the”.

This section describes an alternate version of Markov text, in which we use a dictionary as the training text. Our hope was that a dictionary would yield nonsense words that “looked like” English, while being more statistically diverse. In particular, our training text was the larger of the two dictionaries described in Section 4. It contains 61,580 English words, each of which contains between 5 and 12 lower-case letters. Before we submitted the words to our generator program, we prepended two underscores and appended an exclamation mark, so the word “table” became “__table!”.

The program starts each word with the (unprinted) digram “__”, and finishes each word when a “!” was generated. In the rest of this section, we will describe how the resulting “dictionary Markov text” compares to the “Biblical Markov text” of Appendix A.

Section A.2 begins by considering an attack based on letter frequencies. The most common, second most common and least common letters in that Bible are e, t and x with frequencies 12.73%, 9.81% and 0.04%, respectively. In the more uniform dictionary, the highs are lower and the lows are higher: the corresponding letters are e, i and x with frequencies 11.01%, 8.77% and 0.32%. For the Bible, the probability of a letter-guessing attacker’s success is 0.0688, which corresponds to 3.86 bits of assurance; for the dictionary, the probability drops slightly to 0.0628, and the bits of assurance rises slightly to 3.99.

Section A.2 then approximates the assurance of Biblical Markov text. Experiment and approximation together reveal that the average probability of an attacker’s success is 0.238, which corresponds to 2.07 bits of assurance against a guesser generating such text. Similar analyses of dictionary Markov text give an average probability of 0.174, which corresponds to 2.53 bits of assurance. This fact was encouraging: each character of dictionary Markov text potentially offers about 22% more assurance than Biblical Markov text!

We generated 100,000 dictionary Markov words to study their attributes. Because the training words were all between 5 and 12 characters long, we had high hopes that the generated words would have similar lengths. Of the 100,000 words, 30% contained fewer than 5 characters, while 19% of the words were longer than 12 characters. The longest word contained 66 characters (and a whopping 230 bits of assurance):

\[\text{adisororcreverierbonessioncalguighticathadosycentoescondowinculum}\]

This table shows the distribution of lengths of the 51% of the words between 5 and 12 characters long.

<table>
<thead>
<tr>
<th>Length</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>Words</td>
<td>10188</td>
<td>9023</td>
<td>7552</td>
<td>6345</td>
<td>5515</td>
<td>4623</td>
<td>3994</td>
<td>3349</td>
</tr>
</tbody>
</table>
We chose to focus on words of length 8, which occur with probability about 0.0635. We modified the generation program to reject a word once it passed a certain length, and also to reject short words. When the program generated 100,000 words of length 8, we found that each character had an average 2.49 bits of assurance against a guesser generating such random text (or just a tad less than the 2.53 bits of the original text, which included the extremely long words). Here is a random sample of those 8-character words:

```
palimard reemboz liancobs dirooting explipar defatine rathwass
```

A simple experiment reversed each word in the dictionary before processing. Piping the output through the same reversal program gives plausible English words, but the generation is from “right-to-left” rather than “left-to-right”. We had hoped that this might have more variability, and therefore more assurance. In fact, it seemed to yield slightly less assurance. Words of unbounded length carried 2.48 bits of assurance per character against a random guesser (reduced from 2.53), and words of length 8 carried 2.43 bits per character (reduced from 2.49). Although the change was in the wrong direction, the fact that the directions differ suggests that right-to-left generation might increase assurance on other training sets.

We next turned to the global attacks studied in Section A.3. The most common among the 100,000 words was “comation”, which appeared 9 times. Four words occurred 8 times (“stiously”, “ingeness”, “extrable” and “distiver”), and one word occurred 7 times (“comently”). Beyond those, 6 words occurred 6 times, 16 words occurred 5 times, 52 words occurred 4 times, 195 words occurred 3 times, and 1563 words occurred twice. Of the 100,000 words, 95,917 were unique. This suggests that these strings are not highly susceptible to blind global attacks.

In Section A.3 we described how a program implemented a global attack by starting with the most common digram, and then always taking the most common next letter. Unfortunately (or not), applying this attack to this dictionary yields the 2-character “word” that consists of the common starting and ending digram “st”. We therefore applied the process by hand to yield the 6-character string “stical”, which is generated with probability 0.0001387, which corresponds to 2.14 bits of assurance per character against such a clever guesser.

The fact that almost 96% of the generated words were unique suggests that 8-letter dictionary Markov challenges are particularly appropriate for the “grooming” described in Section A.5. Because so few words are repeated, an efficient implementation of grooming might keep a dictionary of repeated words, which are then discarded if they happen to be randomly generated. One can also apply the “limited successor grooming” of that section to dictionary Markov text; here are challenges of length 8 generated by grooming to 8 successors:

```
pansonet andobphon celfdozo centhous sumency subrousse purocele
```

Section A.6 showed that the contextual attack of guessing the most likely letter following a given digram succeeds with probability 0.375, for about 1.414 bits of assurance. For the dictionary Markov text, that probability drops to 0.288, corresponding to 1.796 bits of assurance, for an increase of about 27%. We suspect that dictionary Markov text will be more robust than Biblical Markov text against other contextual attacks as well.

How many dictionary Markov words are there of length 7? What is the probability that a dictionary Markov word will be 66 characters long? We have constructed a program that uses dynamic programming to answer these questions and others. The
Dynamic programming states correspond to the digrams of the Markov states, and the program proceeds systematically from strings of length \( N \) to strings of length \( N+1 \).

The program records the number of strings of length \( N \) that end in the digram “st” (for instance), or the probability that a string of length \( N \) ends in that digram. If \( S \) is the number of states (bounded above in this case by \( 26^2 \)) and \( L \) is the number of letters (26 in this case), then the resulting code uses \( O(S) \) space and runs in \( O(NSL) \) time, both of which we conjecture are optimal.

The Biblical Markov text in Appendix A and the dictionary Markov text of this appendix are near two endpoints of a spectrum. One could also use as training text the original words from a body of text with a space between each word, and then select as challenges words of an appropriate length. Placing two spaces between words resets the Markov process, and converts the training text to a weighted dictionary of the words in the document. Including only the unique words “unweights” the dictionary, and thereby gives less weight to common words. Many of these choices might produce useful CAPTCHA challenge strings, especially when combined with grooming schemes.

Instead of a dictionary of words, one might train on a file of names, such as those described at the end of Section 4. Here are 8-character names produced in that fashion, without human selection. Can you guess whether a particular name was produced by training on the female first names, male first names, or last names?

- Cargilyn Ryjoline
- Vicorlee Chelinne
- Chiredry Micharia
- Ferrilla Jakelmed Lemannie
- Jefferne Judichen Javerick
- Brolyre Samellas
- Letherpe Bondreno Debraund
- Siesimes Wajalain
- Polninel Fingengs

The first row represents female first names, the second row represents male first names, and the third row represents last names. Preliminary experiments indicate that the (ungroomed) 8-character last names provide about 2.67 bits of assurance per character.

References


