How to Memorize a Random 60-Bit String

Marjan Ghazvininejad
Information Sciences Institute
Department of Computer Science
University of Southern California
ghazvini@isi.edu

Kevin Knight
Information Sciences Institute
Department of Computer Science
University of Southern California
knight@isi.edu

Abstract

User-generated passwords tend to be memorable, but not secure. A random, computer-generated 60-bit string is much more secure. However, users cannot memorize random 60-bit strings. In this paper, we investigate methods for converting arbitrary bit strings into English word sequences (both prose and poetry), and we study their memorability and other properties.

1 Introduction

Passwords chosen by users (e.g., "Scarlet%2") are easy to remember, but not secure (Florencio and Herley, 2007). A more secure method is to use a system-assigned 60-bit random password, such as 0010100010100...00101001. However, this string is hard to memorize. In this paper, we convert such strings into English phrases, in order to improve their memorability, using natural language processing to select fluent passphrases.

Our methods are inspired by an XKCD cartoon[1] that proposes to convert a randomly-chosen 44-bit password into a short, nonsensical sequence of English words. The proposed system divides the 44-bit password into four 11-bit chunks, and each chunk provides an index into a 2048-word English dictionary. XKCD's example passphrase is *correct horse battery staple*:

```
44-bit password   English phrase
---------------   --------------
10101101010    -> correct
10010110101    -> horse
01010101010    -> battery
10110101101    -> staple
```

The four-word sequence is nonsense, but it is easier to memorize than the 44-bit string, and XKCD hypothesizes that users can improve memorability by building an image or story around the four words.

In this paper, we investigate other methods for converting a system-generated bit string into a memorable sequence of English words. Our methods produce whole sentences, e.g.

```
Fox news networks are seeking
views from downtown streets.
```

as well as short poems, e.g.

```
Diversity inside replied,
Soprano finally reside.
```

We also move to 60-bit passwords, for better security. One source claims:

> As of 2011, available commercial products claim the ability to test up to 2,800,000,000 passwords a second on a standard desktop computer using a high-end graphics processor.[2]

If this is correct, a 44-bit password would take one hour to crack, while a 60-bit password would take 11.3 years.

Our concrete task is as follows:

[1] http://xkcd.com/936

[2] http://en.wikipedia.org/wiki/Password_cracking

Human Language Technologies: The 2015 Annual Conference of the North American Chapter of the ACL, pages 1569–1575,
Denver, Colorado, May 31 – June 5, 2015. ©2015 Association for Computational Linguistics

Method Name	Average Number of Words	Average Number of Characters	AVG LM Score	Capacity	Sample Passwords
XKCD	4	31.2	-62.42	1	fees wesley inmate decentralization
					photo bros nan plain
					embarrass debating gaskell jennie
First Letter Mnemonic	15	87.7	-61.20	$2 \cdot 10^{51}$	It makes me think of union pacific resource said it looks like most commercial networks .
					Some companies keep their windows rolled down so you don't feel connected to any community .
					Contains extreme violence and it was a matter of not only its second straight loss .
All Letter Method	11.8	70.8	-58.83	$3 \cdot 10^{56}$	Parking and utilities have been searching for a third straight road win .
					It was the same girl and now a law professor in the former east german town .
					I know a man who said he was chief of staffs in a real and deep conversation .
Frequency Method	9.7	55.5	-52.88	$6 \cdot 10^{14}$	Fox news networks are seeking views from downtown streets .
					The review found a silver tree through documents and artifacts .
					These big questions are bothering me a bit stronger .
Poetry	7.2	52.7	-73.15	10^6	Joanna kissing verified soprano finally reside
					Diversity inside replied retreats or colors justified
					Surprise celebrity without the dragging allison throughout

Table 1: Comparison of methods that convert system-assigned 60-bit strings into English word sequences. Average word lengths range from 4 (XKCD) to 15 (First Letter Mnemonic). Average character lengths include spaces. LM score refers to the log probability assigned by a 5-gram English language model trained on the Gigaword corpus. Capacity tells how many English word sequences are available for an individual 60-bit input string.

- **Input:** A random, system-generated 60-bit password.
- **Output:** An English word sequence with two properties:
  - It is memorable.
  - We can deterministically recover the original input 60-bit string from it.

This implies that we map 2^{60} distinct bit strings into 2^{60} distinct English sequences. If a user memorizes the English word sequence supplied to them, then they have effectively memorized the 60-bit string.

2 Password Generation Methods

We now describe our baseline password generation method, followed by four novel methods. In Section 3 we experimentally test their memorability.

2.1 XKCD Baseline

Our baseline is a version of XKCD. Instead of a 2048-word dictionary, we use a 32,7868-word dictionary. We assign each word a distinct 15-bit code.

At runtime, we take a system-assigned 60-bit code and split it into four 15-bit sequences. We then substitute each 15-bit segment with its corresponding word. By doing this, we convert a random 60-bit code into a 4-word password.

The first row of Table 1 shows three sample XKCD passwords, along with other information, such as the average number of characters (including spaces).

2.2 First Letter Mnemonic

XKCD passwords are short but nonsensical, so we now look into methods that instead create longer but fluent English sentences. We might think to guarantee fluency by selecting sentences from an already-existing text corpus, but no corpus is large enough to contain 2^{60} ($\sim 10^{18}$) distinct sentences. Therefore, we must be able to synthesize new English strings.

Bit Sequence	Mapped Character	Bit Sequence	Mapped Character
0000	e	1000	r,x
0001	t	1001	d,j
0010	a	1010	l,k
0011	o	1011	c,v
0100	i	1100	u,b
0101	n	1101	m,p
0110	s,z	1110	w,y
0111	h,q	1111	f,g

Table 2: Mapping function between 4-bit sequences and English letters in the First Letter Mnemonic method.

In our first sentence generation method (*First Letter Mnemonic*), we store our input 60-bit code in the first letters of each word. We divide the 60-bit code into 4-bit sections, e.g., ‘0100-1101-1101-...’. Every 4-bit sequence type corresponds to an English letter or two, per Table 2. We build a word-confusion network (or “sausage lattice”) by replacing each 4-bit code with all English words that start with a corresponding letter, e.g.:

```
0100    1101    1111         ... 0011
----    ----    ----             ----
income  my      frog         ... octopus
is      miner   feast        ... of
inner   priest  gratuitous   ... oregon
 ...     ...     ...              ...
```

This yields about 10^{74} paths, some good (*is my frog...*) and some bad (*income miner feast...*). To select the most fluent path, we train a 5-gram language model with the SRILM toolkit (Stolcke, 2002) on the English Gigaword corpus.[3] SRILM also includes functionality for extracting the best path from a confusion network.

Table 1 shows sample sentences generated by the method. Perhaps surprisingly, even though the sentences are much longer than XKCD (15 words versus 4 words), the n-gram language model (LM) score is a bit better. The sentences are locally fluent, but not perfectly grammatical.

We can easily reconstruct the original 60-bit code by extracting the first letter of each word and applying the Table 2 mapping in reverse.

2.3 All Letter Method

Most of the characters in the previous methods seem “wasted”, as only the word-initial letters bear information relevant to reconstructing the original 60-

[3] https://catalog.ldc.upenn.edu/LDC2011T07

Bit Sequence	Mapped Characters
0	e, o, i, h, r, c, u, f, g, b, v, x ,q
1	t, a, n, s, d, l, m, w, y, p, k, j, z

Table 3: Mapping function between bits and English characters in the All Letter Method.

bit string. Our next technique (*All Letter Method*) non-deterministically translates *every* bit into an English letter, per Table 3. Additionally, we non-deterministically introduce a space (or not) between each pair of letters.

This yields $4 \cdot 10^{84}$ possible output strings per input, $3 \cdot 10^{56}$ of which consist of legal English words. From those $3 \cdot 10^{56}$ strings, we choose the one that yields the best word 5-gram score.

It is not immediately clear how to process a letter-based lattice with a word-based language model. We solve this search problem by casting it as one of machine translation from bit-strings to English. We create a phrase translation table by pairing each English word with a corresponding "bit phrase", using Table 3 in reverse. Sample entries include:

```
din ||| 1 0 1
through ||| 1 0 0 0 0 0 0
yields ||| 1 0 0 1 1 1
```

We then use the Moses machine translation toolkit (Koehn et al., 2007) to search for the 1-best translation of our input 60-bit string, using the phrase table and a 5-gram English LM, disallowing re-ordering.

Table 1 shows that these sentences are shorter than the mnemonic method (11.8 words versus 15 words), without losing fluency.

Given a generated English sequence, we can deterministically reconstruct the original 60-bit input string, using the above phrase table in reverse.

2.4 Frequency Method

Sentence passwords from the previous method contain 70.8 characters on average (including spaces). Classic studies by Shannon (1951) and others estimate that printed English may ultimately be compressible to about one bit per character. This implies we might be able to produce shorter output (60 characters, including space) while maintaining normal English fluency.

Our next technique (*Frequency Method*) modifies the phrase table by assigning short bit codes to frequent words, and long bit codes to infrequent words. For example:

```
din ||| 0 1 1 0 1 0 1 0 0
through ||| 1 1 1 1
yields ||| 0 1 0 1 1 1 0 1
```

Note that the word *din* is now mapped to a 9-bit sequence rather than a 3-bit sequence. More precisely, we map each word to a random bit sequence of length $\lfloor \max(1, -\alpha \times log\ \mathrm{P}(word) + \beta) \rfloor$. By changing variables α and β we can vary between smooth but long sentences ($\alpha = 1$ and $\beta = 0$) to XKCD-style phrases ($\alpha = 0$ and $\beta = 15$).

Table 1 shows example sentences we obtain with $\alpha = 2.5$ and $\beta = -2.5$, yielding sentences of 9.7 words on average.

2.5 Poetry

In ancient times, people recorded long, historical epics using poetry, to enhance memorability. We follow this idea by turning each system-assigned 60-bit string into a short, distinct English poem. Our format is the *rhyming iambic tetrameter couplet*:

- The poem contains two lines of eight syllables each.
- Lines are in iambic meter, i.e., their syllables have the stress pattern 01010101, where 0 represents an unstressed syllable, and 1 represents a stressed syllable. We also allow 01010100, to allow a line to end in a word like *Angela*.
- The two lines end in a pair of rhyming words. Words rhyme if their phoneme sequences match from the final stressed vowel onwards. We obtain stress patterns and phoneme sequences from the CMU pronunciation dictionary.[4]

Monosyllabic words cause trouble, because their stress often depends on context (Greene et al., 2010). For example, *eighth* is stressed in *eighth street*, but not in *eighth avenue*. This makes it hard to guarantee that automatically-generated lines will scan as intended. We therefore eject all monosyllabic words

[4] http://www.speech.cs.cmu.edu/cgi-bin/cmudict

from the vocabulary, except for six unstressed ones (*a*, *an*, *and*, *the*, *of*, *or*).

Here is a sample poem password:

```
The le-gen-da-ry Ja-pan-ese
 ↓   ↑   ↓  ↑  ↓ ↑   ↓   ↑
Sub-si-di-ar-ies ov-er-seas
 ↓   ↑  ↓  ↑   ↓   ↑  ↓  ↑
```

Meter and rhyme constraints make it difficult to use the Moses machine translation toolkit to search for fluent output, as we did above; the decoder state must be augmented with additional short- and long-distance information (Genzel et al., 2010).

Instead, we build a large finite-state acceptor (FSA) with a path for each legal poem. In each path, the second line of the poem is reversed, so that we can enforce rhyming locally.

The details of our FSA construction are as follows. First, we create a finite-state transducer (FST) that maps each input English word onto four sequences that capture its essential properties, e.g.:

```
create -> 0 1
create -> 0 1 EY-T
create -> 1r 0r
create -> EY-T 1r 0r
```

Here, `EY-T` represents the rhyme-class of words like *create* and *debate*. The *r* indicates a stress pattern in the right-to-left direction.

We then compose this FST with an FSA that only accepts sequences of the form:

```
0 1 0 1 0 1 0 1 X X 1r 0r 1r 0r 1r 0r 1r 0r
```

where X and X are identical rhyme classes (e.g., EY-T and EY-T).

It remains to map an arbitrary 60-bit string onto a path in the FSA. Let k be the integer representation of the 60-bit string. If the FSA contains exactly 2^{60} paths, we can easily select the kth path using the following method. At each node N of the FSA, we store the total number of paths from N to the final state—this takes linear time if we visit states in reverse topological order. We then traverse the FSA deterministically from the start state, using k to guide the path selection.

Our FSA actually contains 2^{79} paths, far more than the required 2^{60}. We can say that the information capacity of the English rhyming iambic tetrameter couplet is 79 bits! Some are very good:

```
Sophisticated potentates
misrepresenting Emirates.

The supervisor notified
the transportation nationwide.

Afghanistan, Afghanistan,
Afghanistan, and Pakistan.
```

while others are very bad:

```
The shirley emmy plebiscite
complete suppressed unlike invite

The shirley emmy plebiscite
complaints suppressed unlike invite

The shirley emmy plebiscite
complaint suppressed unlike invite
```

Fortunately, because our FSA contains over a million times the required 2^{60} paths, we can avoid these bad outputs. For any particular 60-bit string, we have a million poems to choose from, and we output only the best one.

More precisely, given a 60-bit input string k, we extract not only the kth FSA path, but also the $k + i \cdot 2^{60}$ paths, with i ranging from 1 to 999,999. We explicitly list out these paths, reversing the second half of each, and score them with our 5-gram LM. We output the poem with the 1-best LM score. Table 1 shows sample outputs.

To reconstruct the original 60-bit string k, we first find the FSA path corresponding to the user-recalled English string (with second half reversed). We use depth-first search to find this path. Once we have the path, it is easy to determine which numbered path it is, lexicographically speaking, using the node-labeling scheme above to recover k.

3 Experiments

We designed two experiments to compare our methods.

The first experiment tests the memorability of passwords. We asked participants to memorize a password from a randomly selected method[5] and recall it two days later. To give more options to users,

[5] In all experiments, we omit the First Letter Mnemonic, due to its low performance in early tests.

Method	Participants	Recalls	Correct Recalls
XKCD	16	12	58.3%
All Letter Method	15	9	33.3%
Frequency Method	15	10	40.0%
Poetry	16	13	61.5%

Table 4: Memorability of passwords generated by our methods. "Recalls" indicates how many participants returned to type their memorized English sequences, and "Correct Recalls" tells how many sequences were accurately remembered.

Method Name	User preference
XKCD	5%
All Letter Method	39%
Frequency Method	37%
Poetry	19%

Table 5: User preferences among passwords generated by our methods.

we let them select from the 10-best passwords according to the LM score for a given 60-bit code. Note that this flexibility is not available for XKCD, which produces only one password per code.

62 users participated in this experiment, 44 returned to recall the password, and 22 successfully recalled the complete password. Table 4 shows that the Poetry and XKCD methods yield passwords that are easiest to remember.

In the second experiment, we present a separate set of users with passwords from each of the four methods. We ask which they would prefer to use, without requiring any memorization. Table 5 shows that users prefer sentences over poetry, and poetry over XKCD.

4 Analysis

Table 4 shows that the Poetry and XKCD methods yield passwords that are easiest to memorize. Complete sentences generated by the All Letter and Frequency Methods are harder to memorize. At the same time Table 5 shows that people like the sentences better than XKCD, so it seems that they overestimate their ability to memorize a sentence of 10-12 words. Here are typical mistakes (S = system-generated, R = as recalled by user):

```
(S) Still looking for ruben sierra could
    be in central michigan
(R) I am still looking for ruben sierra
    in central michigan

(S) That we were required to go to
    college more than action movies
(R) We are required to go to
    college more than action movies

(S) No dressing allowed under canon law
    in the youth group
(R) No dresses allowed under canon law
    for youth groups
```

Users remember the gist of a sentence very well, but have trouble reproducing the exact wording. Post-experiment interview reveal this to be partly an effect of overconfidence. Users put little mental work into memorizing sentences, beyond choosing among the 10-best alternatives presented to them. By contrast, they put much more work into memorizing an XKCD phrase, actively building a mental image or story to connect the four otherwise unrelated words.

5 Future Directions

Actually, we can often *automatically* determine that a user-recalled sequence is wrong. For example, when we go to reconstruct the 60-bit input string from a user-recalled sequence, we may find that we get a 62-bit string instead. We can then automatically prod the user into trying again, but we find that this is not effective in practice. An intriguing direction is to do automatic error-correction, i.e., take the user-recalled sequence and find the closest match among the 2^{60} English sequences producible by the method. Of course, it is a challenge to do this with 1-best outputs of an MT system that uses heuristic beam search, and we must also ensure that security is maintained.

We may also investigate new ways to re-rank n-best lists. Language model scoring is a good start, but we may prefer vivid, concrete, or other types of words, or we may use text data associated with the user (papers, emails) for secure yet personalized password generation.

6 Related Work

Gasser (1975), Crawford and Aycock (2008), and Shay et al. (2012) describe systems that produce meaningless but pronounceable passwords, such as "tufritvi" . However, their systems can only assign $\sim 2^{30}$ distinct passwords.

Jeyaraman and Topkara (2005) suggest generating a random sequence of characters, and finding a mnemonic for it in a text corpus. A limited corpus means they again have a small space of system-assigned passwords. We propose a similar method in Section 2.2, but we automatically synthesize a new mnemonic word sequence.

Kurzban (1985) and Shay et al. (2012) use a method similar to XKCD with small dictionaries. This leads to longer nonsense sequences that can be difficult to remember.

7 Conclusion

We introduced several methods for generating secure passwords in the form of English word sequences. We learned that long sentences are seemingly easy to remember, but actually hard to reproduce, and we also learned that our poetry method produced relatively short, memorable passwords that are liked by users.

Acknowledgments

We would like to thank James Bedell, Aliya Deri, Tomer Levinboim, Jonathan May, Nima Pourdamghani and the anonymous reviewers for their very helpful comments. This work was supported in part by DARPA contract FA-8750-13-2-0045.

References

Heather Crawford and John Aycock. 2008. Kwyjibo: automatic domain name generation. *Software: Practice and Experience*, 38(14):1561–1567.

Dinei Florencio and Cormac Herley. 2007. A large-scale study of web password habits. In *Proceedings of the 16th international conference on World Wide Web*, pages 657–666. ACM.

Morrie Gasser. 1975. A random word generator for pronounceable passwords. Technical report, Electronic Systems Division, Air Force Systems Command, USAF.

Dmitriy Genzel, Jakob Uszkoreit, and Franz Och. 2010. Poetic statistical machine translation: rhyme and meter. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pages 158–166. Association for Computational Linguistics.

Erica Greene, Tugba Bodrumlu, and Kevin Knight. 2010. Automatic analysis of rhythmic poetry with applications to generation and translation. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 524–533. Association for Computational Linguistics.

Sundararaman Jeyaraman and Umut Topkara. 2005. Have your cake and eat it too–infusing usability into text-password based authentication systems. *In Proceedings of ACSAC*.

Philipp Koehn, Hieu Hoang, Alexandra Birch, Chris Callison-Burch, Marcello Federico, Nicola Bertoldi, Brooke Cowan, Wade Shen, Christine Moran, Richard Zens, et al. 2007. Moses: Open source toolkit for statistical machine translation. In *Proceedings of the 45th Annual Meeting of the ACL, Demo and Poster Sessions*, pages 177–180. Association for Computational Linguistics.

Stanley A Kurzban. 1985. Easily remembered passphrases: a better approach. *ACM SIGSAC Review*, 3(2-4):10–21.

Claude E. Shannon. 1951. Prediction and entropy of printed English. *Bell System Technical Journal*, 30(1):50–64.

Richard Shay, Patrick Gage Kelley, Saranga Komanduri, Michelle L Mazurek, Blase Ur, Timothy Vidas, Lujo Bauer, Nicolas Christin, and Lorrie Faith Cranor. 2012. Correct horse battery staple: Exploring the usability of system-assigned passphrases. In *Proceedings of the Eighth Symposium on Usable Privacy and Security*, page 7. ACM.

Andreas Stolcke. 2002. SRILM-an extensible language modeling toolkit. In *INTERSPEECH*, pages 901–904.