

# A Statistically Based Method of Guessing Passwords With Known Plaintext (Extended Abstract)

Michael de Mare \*  
Computer Science Department  
SUNY Institute of Technology

**Keywords:** computer intrusion, keys, passwords, security, cyberwarfare

## Abstract

Documents from an organization can be analyzed into tables of trigrams to generate probable passwords for that organization. Generating a password with a given probability where  $n$  characters have been chosen is as simple as looking for the most probable trigram that starts with the two last letters and using it to choose the next letter. Probable passwords of any given length can be generated.

## 1 Introduction

Guessing passwords is not useful only for intruding into systems. It can also be used to test the security of existing systems, to decrypt the conventionally encrypted private key of a public key cryptosystem or to retrieve access to data that is on a system, but unavailable because the keyholder is no longer available.

---

(C) Copyright 2004 SUNY Institute of Technology

\*Portions of this research were performed at TechnoProductions, Corp

## 2 Definitions

We define a text space to be a set of properly spelled words, which follow one another with given probabilities. An example of a text space would be a dialect of the English language.

We define a bigram to be a sequence of two letters in a character set paired with a probability that a given pair of letters will be the bigram in the given text space.

We define a trigram to be a sequence of three letters along with the probability that a sequence in the given text space will be this sequence of letters.

We abuse the definition of known plaintext. Traditionally, known plaintext means text that is encrypted that we know both the ciphertext and plaintext values for. In this paper we use it to denote text which is written in a given text space. The argument for calling it known plaintext is that often this plaintext is protected by the key we are trying to guess.

## 3 The Algorithm

This section describes the algorithm for generating probable passwords for key protected systems with known plaintext. The known plaintext is used to compute the probability of each possible trigram in the key space.

### 3.1 Trigrams

In World War II, cryptanalysis revolved around trigrams [1]. A trigram is a set of three letters with a probability in a given text domain. In available literature, the text domain is a language, but with the tendency of organizations to develop their own jargon, which is on the one hand relatively restricted (the Oxford English Dictionary contains 750,000 words, larger than anyone's vocabulary) and the tendency to have project and department names, it makes sense for the text domain to be extracted from documents from the organization from whom passwords must be guessed.

Given the most likely set of characters in the password (usually lowercase letters, passwords can be tried with digits alternately appended to them if the system requires the password to contain digits), create a table for which

each possible set of three letters has an entry. Also create a table of bigrams [1] which is like trigrams but are pairs of characters.

### **3.2 Computing the Probability of Bigrams and Trigrams**

To compute the probability of each bigram and each trigram, scan through the document remembering the previous character and the previous bigram. The next trigram is the previous bigram with the current character appended. Increment its count by one, along with the total count. The next bigram is the previous character with the current character appended. Repeat this until the end of all known plaintext is reached.

Compute the probability of each bigram and trigram by dividing the hits by the total count.

### **3.3 Generating Probable Passwords**

Sort the bigram and trigram tables by probability. For each starting letter in the character set try bigrams down to a minimum probability determined by computing resources. Then, using the two letters, try the trigrams with descending probability down to a level determined by computing resources whose first two letters are the previous two letters. Try this password. Continue adding a characters by trying the last letter of the trigram determined by the previous two letters down to a minimum probability recursively and trying these passwords<sup>1</sup>.

## **4 Probability of a Guess**

It is important to know and exploit the probability of the guesses you are generating. Once you know how to compute them, you can tune your program to discontinue when the probability gets too low, and move on to the next trigram, or if going all the way back, trigram. One can also sort guesses by probability.

---

<sup>1</sup>On systems requiring a digit in the password try appending each numerical digit to each attempt

## 4.1 Computing the Probability of a Guess

The first three letters have the probability given by the trigram. To compute the probability of one of these as a password, divide the probability of that trigram by the number of possible lengths of passwords.

For guesses longer than three characters, let the initial probability,  $P_3$  be the probability of the trigram. For  $i = 4 \dots n$  where  $n$  is the length of the guess. Let  $T_i$  be the probability of the trigram formed by the substring  $(i-2, i-1)$  and  $B_i$  be the probability of the trigram formed by the substring  $(i-1, i)$ . Let  $P_i = (B_i + T_i)P_{i-1}$ . Let  $C$  be the number of possible lengths of passwords<sup>2</sup>. The probability of a guess will be  $P_n/C$ . [4]

## 4.2 Using the Probability of a Guess

The best way to use the probability is to determine when to abandon a given path and move on to the next trigram or, if you are down to two letters in recursing out, bigram. The guesses will already be sorted by probability by the algorithm. Computing the probability is for the purpose of tuning your program to cover as much as possible with the limited computing resources available.

The method of tuning is as follows. In the algorithm for generating probable passwords, the algorithm states that one tries combinations down to a minimum probability determined by computing resources. The current probability can be computed using the technique in the previous subsection. A little experimenting may be useful on your machine to determine what minimum probability is feasible, but given the high redundancy of language, it is likely you will get a fairly high probability without massive computing resources.

## 5 Defeating this Algorithm

Passwords are chosen by people. The people choosing the passwords need them to be easily remembered. As a result a mnemonic, project name or concatenation of words will be chosen<sup>3</sup>. Mnemonics and project names will

---

<sup>2</sup>Most systems impose both a minimum and a maximum number of characters in a password

<sup>3</sup>Most systems will not allow dictionary words to be chosen

turn up in the probable passwords, since they have the statistics of the documents the user generates or reads. People do not tend to remember random letters and digits.

There are rules which can make the job of finding probable passwords more difficult by a small factor. Let  $N$  be the largest allowed password.

## 5.1 Embedded digits

Let  $L$  be the length of the password. When embedded digits are required by the system, each guess needs to be tried with each possible digit in each possible place in the password. Since people will tend to use the minimum requirement, it suffices to only check for one embedded digit. There are 10 digits, so this increases the difficulty of guessing a password by a maximum of  $10L$ . Since this is a small constant factor, this does not defeat this algorithm.

## 5.2 Mixed case

When a system requires passwords to have at least one uppercase character, it is simply a factor of  $L$  to scan through the guess trying each letter as an uppercase.

## 5.3 Long Passwords

Requiring long passwords will lower probabilities to the point of making this algorithm inefficient. It presents the problem, however, of causing people to forget their passwords and/or, use well known quotes which may be guessed from their literary tastes.

## 5.4 Delaying Hash Results for Wrong Passwords

This does not help, as the implementation may be multithreaded. In fact, an efficient implementation might be for the program to generate a list of passwords to try, fork a process for each user and test the users in parallel.

## 5.5 Challenge/Response

Challenge/Response systems, which ask a user a question and require that it be answered correctly, is secure from this sort of attack. It poses some problems, however,

- This can not protect encryption keys.
- A database of questions to randomly ask the user must be constructed at significant difficulty.
- The user now has to remember the answers to a large number of challenges rather than one password, increasing the likelihood of not being able to get legitimate access.
- An intruder may continue trying to get access until it finds a challenge it thinks it can correctly respond to.

## 5.6 PINS

A PIN is a Personal Identification Number of a specific length.

As a supplement to passwords, a computer-issued PIN which may be carried on one's person until memorized is an effective countermeasure. Experience with telephones is that people can remember seven digits. Since PINS are punched on the numeric keypad, much like a telephone keypad, motor skill memory will help supplement a person's memory, ie. people may punch in number without remembering it. Cards carrying the PIN should be destroyed immediately after memorization.

## 5.7 Low Cost Devices

Magnetic card readers, such as those used for credit cards are a cost-effective supplement to passwords. For installations with a high user to access-point ratio, barcode readers are also cost-effective, because although bar code readers are at the time of writing more expensive, business-card blanks are less. A cautionary note on the use of barcodes is that the number should not be printed, to prevent sharp-eyed intruders from copying them down.

## 6 Open Problems

The biggest open problem is how to defeat this algorithm and still have a user friendly access mechanism without resorting to thumbprint or retina scans which provide their own set of problems. PINs seem promising as do low-cost card readers. Both are a supplement, as opposed to a stand-alone solution.

Further research on this would be to determine ideal password lengths and rules which makes this algorithm too inefficient to use.

Another open problem is whether using  $n$ -grams with  $n > 3$  improves the algorithm.

## 7 Conclusions

This technique can be used to either test the security of keys or passwords in systems or to attempt to break them. It is most likely to guess passwords when used on a group of passwords where the goal is to guess as many passwords as possible. When used in this fashion, a list of probable passwords should be generated and tried on all the candidates.

## References

- [1] W. F. Friedman. *Military Cryptanalysis*. Aegean Park Press.
- [2] S. Kullback. *Information Theory and Statistics*. Dover Publications, Inc, 31 Easy 2nd St, Mineola, NY 11501, 1959.
- [3] R. J. A. Little and D. B. Rubin. *Statistical Analysis with Missing Data*. John Wiley and Sons, 1987.
- [4] S. Ross. *A First Course in Probability*. Prentice-Hall, Inc, Upper Saddle River, NJ 07458, fifth edition, 1998.