Linguistic Cracking of Passphrases using Markov Chains

Peder Sparell, Mikael Simovits

Simovits Consulting AB, Saltmätargatan 8A, 113 59, Stockholm, Sweden
{peder.sparell, mikael}@simovits.com

Abstract. In order to remember long passwords, it is not uncommon users are recommended to create a sentence which then is assembled to form a long password, a passphrase. However, theoretically a language is very limited and predictable, why a linguistically correct passphrase according to Shannon's definition of information theory should be relatively easy to crack compared to brute-force.

This work focuses on cracking linguistically correct passphrases, partly to determine to what extent it is advisable to base a password policy on such phrases for protection of data, and partly because today, widely available effective methods to crack passwords based on phrases are missing.

Within this work, phrases were generated for further processing by available cracking applications, and the language of the phrases were modeled using a Markov process. In this process, phrases were built up by using the number of observed instances of subsequent characters or words in a source text, known as n-grams, to determine the possible/probable next character/word in the phrases.

The work shows that by creating models of language, linguistically correct passphrases can be broken in a practical way compared to an exhaustive search. In the tests, passphrases consisting of up to 20 characters were broken.

Keywords: Passphrases • Cracking • Markov chains

1 Introduction

Since password cracking constantly becomes more effective as the computing power becomes cheaper, usually recommendations say that passwords should be long and random. Humans, however, often have difficulties remembering such passwords, so one of these requirements are often ignored. It is usually argued that assembled sentences, passphrases, is a good solution and represents great security because these are long and still easy to remember. However, looking at it from an information theoretical point of view, a sentence of 20 characters should be as easy to crack as a random password of eight characters.

We will in this work propose a solution on how to make the cracking of passphrases more efficient by only testing linguistically correct phrases and building a phrase generator for cracking such passphrases.

The main objective has been to crack lowercase passphrases up to 16 characters of length, but tests has also been performed on lengths up to 20, using a normal workstation.

2 Related Work

Most password crackers available are able to put together phrases by combining several words from dictionaries, but this is usually done in a more brute-force manner by testing all possible combinations of words from the different lists. It is also possible to use lists of known and manually assembled phrases, providing an often far from adequate search scope. Practically, passphrase cracking is a somewhat inadequately applied area.

However, there is some research and theories regarding passphrases and their strength, or entropy. The most significant work done, and which this work to a large extent was based on, is Shannon's theories and definition of entropy in (Shannon CE, 1948), and his estimate of the entropy of the English language (Shannon CE, 1951). Shannon's focus was not on password cracking, but more generally on information and communication. His theories have been applied in a lot of areas though.

Although it has become common practice to measure password strength in entropy, it has been questioned as a suitable indicator. One example is (Ma, Campbell, Transgender, & Kleeman, 2010) who question the use of entropy in this context because it only gives a lower limit for how many guesses needed to crack a password. They also consider that it is not possible to determine the entropy of a password through a Markov Process according to Shannon's guessing experiment, because during a password attack there is no knowledge of any leading characters, as there is in Shannon's experiments. At password guessing you know only after the entire word is guessed if it was right or not.

In the conference paper "Effect of Grammar on Security of Long Passwords" (Rao, Jha, & Kini, 2013) an attempt to use grammar to crack passphrases is presented. The papers results show a slight increase in the number of cracked passwords using grammatical rules compared to other methods. It also discusses additional topics related to this work, such as user behavior regarding the selection of number of words in a passphrase and shortcomings of available cracking applications regarding passphrases.

In the paper "Password cracking using probabilistic context-free grammars" (Weir, Aggarwal, de Medeiros, & Glodek, 2009), context-free grammar is used in a successful attempt to crack passwords. The passwords that are processed are not limited to passphrases, the work is more focused on traditional passwords where grammar rather is used for word mutations.

In (Bonneau & Shutova, Linguistic properties of multi-word passphrases, 2012) patterns in human choice of passphrases are studied, based on Amazons now discontinued PayPhrase service. In that study, conclusions are drawn that a 4 words long passphrase probably has less than 30 bits of security because users tend to choose linguistically correct phrases.

3 Theory

A completely random password with 8 characters of length in lowercase letters from the English alphabet, according to Shannon's definition of entropy (Shannon C. E., 1948), has a total entropy of about 38 bits ($8 * \log_2(26)$). In case of a linguistically correct password, the probability of making a correct guess is affected since some letters are more frequent than others in a human language, and thus the entropy is also affected. If the frequency of n-grams is taken into account, i.e. how often certain letters follow others, entropy is further reduced.

There have been many attempts using different models to estimate the entropy of the English language, and according to NIST's (National Institute of Standards and Technology (NIST), 2013) interpretation of Shannon and his experiments, it is possible to estimate the entropy of the first letter to 4 bits and then reduce it for subsequent characters. For characters 2-8, an entropy of 2 bits per character should be used, and for the characters 9-20 entropy is increased with 1.5 bits per character. Character 21 and subsequent can be calculated with as low as 1 bit per letter.
Based on these assumptions, a linguistically correct passphrase of length 20 (entropy 36 bits) theoretically should be easier to crack than a randomly chosen password of length 8 (entropy 37.6 bits).

3.1 Markov Chains

To obtain a low entropy, or high linguistic correctness, in the modeling of a language, it must be based on a good language model, and the Markov model was one proposed by Shannon in his aforementioned work.
A Markov chain process is by definition a random process with different states, where the probability distribution between transitions to a new state is dependent only on the current state. It is not taking into account any previous states thus the process has no "memory". If the process meet this requirement it satisfies the "Markov property".
There is also a variant of Markov chain, called Markov chain of order *m*, which ironically takes *m* number of states into account. With a little determination it is possible to even get this Markov chain to satisfy the Markov property, by combining the *m* previous states and instead define these combinations as states. For example, if there are states A, B, and C, so instead of saying that A transitions to B which then transitions to C, new states AB and BC etc. are defined, and it is said that AB transitions to BC.

3.2 N-grams

In linguistics the term n-grams is frequently used. These are sequences of *n* number of elements, which may consist of characters or words. A unigram is an n-gram of the first order and is simply a character or a word, a bigram is an n-gram of the second order, that is: a sequence of two characters or words. Correspondingly trigram, four-grams (4-grams), five-grams (5-grams) and so on are defined.
Often these n-grams are extracted from larger texts, corpora, where the number of occurrences are counted, to be used as statistical data. One way of modeling a language

is to use these statistical n-gram data in a process using Markov chains of (n-1):th order to predict subsequent characters/words. This model does not always produce perfect results, but with a good statistical basis, it can actually give surprisingly linguistically correct results.
The formal definition for an n-gram model is provided by this probability formula:

$$P(X_i = x_i \mid X_{i-(n-1)} = x_{i-(n-1)}, \ldots, X_{i-1} = x_{i-1})$$

Where the stochastic variable X_i is the character that is assessed, given the $n-1$ preceding characters, and n is equivalent to the n in "n-gram".
In the case of trigrams is thus the formula:

$$P(X_i = x_i | X_{i-2} = x_{i-2}, X_{i-1} = x_{i-1})$$

If the n-grams are on character level, i.e. characters has been selected as unit, a bigram (2-gram) consists of two characters, a trigram of three and so on.
It is also possible to use words as unit in the n-grams instead of characters. The big difference is that nonsense words are avoided, but the statistics need to be collected from a larger base source text.

With higher order of n-grams more linguistically correct words/sentences are generated, but its similarity to the source text increases, which thus means less variation in the results.

4 Implementation

4.1 General Architecture

This works implementation was based on two console applications, the first of which generated the n-gram statistics, and the other generated output in form of one phrase per line to a stdout stream. This was done in order for it to be possible to use the program either directly with a cracker application (which often accept stdin as input source), or to redirect the stream to a file for later use, reuse or use with a cracker who cannot read from stdin streams, which then makes it possible to make a dictionary attack with the generated file as wordlist.
A disadvantage saving to file is that the generated files may take considerably large amount of disk space, but since the generation/cracking ratio is large, that is: the time to generate the phrases versus the time to run them through a cracker, the reuse of files can be more effective.

4.2 Phase 1: N-gram Extraction

To implement generation of phrases using a Markov process, first thing needed is statistics showing the n-grams that occur within a larger text, and its number of occurrences in this text. A text source should be selected that are linguistically relevant for the passwords to be cracked. For example should not an old Bible or the works of Shakespeare be chosen, because their language is very different from our modern language which people usually use as their passwords basis. A corpus based on, for example news, blogs or the like is probably preferable.

With the text file as input, the file is now processed and scanned for n-grams, and the n-grams found are added to a sorted list that also shows their number of occurrences. When the file is read to the end, and the list is complete, it is sent as output to stdout. Depending on whether it is selected to generate n-grams on character level or word level, processing is somewhat different:

Character Level

In the case of character level, marking of sentence/phrase breaks, which usually manifests themselves in the form of the punctuation symbols ".", "!", "?", and ":" (usually with subsequent whitespace) is done by replacing them with a single period character without whitespace. Thereafter, the remaining whitespace (spaces, tabs, etc.) are replaced with the underscore character "_".

N-grams which consists of regular characters, and the "_" and "." are inserted as a new record with the occurrence value 1 in the list of observed n-grams, alternatively 1 is added to the record if it is already in the list.

Why sentence breaks are desired in the n-gram file is partly due to that the first character, according to NIST and Shannon, has the largest entropy, so it is important that the beginning of the phrase is guessed as well as possible. Choosing the start of the generated phrases by taking into account which n-grams that are specifically common at the start of sentences, increases the quality of the phrases. For example, the 7-gram "name_is" occurs relatively frequent in a text, but rarely at a sentence start.
Knowing common sentence breaks also makes it possible to make an adaptation of the Markov process, in order to end the generated phrases in a more intelligent way.
A similar reasoning applies to the whitespaces. Despite the generated phrases are not intended to contain spaces, they should nevertheless be taken into account in the process because, for example, it is far more unusual that a word starts with the letter "x" than that it occurs later in the word.

Word Level

If instead generation of n-grams on the word level is chosen, the procedure is similar, but some details are different of course. In this case, no consideration is given to whitespace since it is understood that the words are surrounded by them. However, punctuation characters are still considered in order to mark sentence breaks, and are replaced with the general self-defined marker, a single period character ".".

Because this work is limited to not to include special characters in its generated phrases, and thus neither in the n-grams, the extraction of words needs to take into account words like "we're" or "don't" that should finally be saved to "were" and "dont" instead of "we" and "re" and "don" and "t".

Corresponding to character level, every n-gram with its number of occurrences are saved in the list.

4.3 Phase 2: Phrase Generation

At this stage, lists of the n-gram statistics that are of interest can be created and saved. These lists are in phase 2 used to generate the actual phrases.

Loading the n-gram List
The n-gram list is loaded from the selected text file and stored in memory as a list of states, which are defined as the n-1 first characters/words of the n-grams, along with their sub-lists/children in form of a list of the next possible subsequent characters/words sorted in order of occurrence.

Starting States
At initiation, the phrase is empty and there is no previous state to evaluate, so it must be determined how the first n-1 characters should be chosen in order to continue the Markov chain. For this, the aforementioned marking of punctuation/sentence breaks are used. The possible startup states are derived from all the n-grams beginning with ".", and for each startup state, in order of occurrence, the function below is executed.

The Main Function
Once the phrase start is determined, one character or word is added at a time, according to the Markov model. Normally when Markov chains are used, they are used to find the most likely chain or a random chain, but here *all* possible chains are searched for.
Therefore the function was made recursive, so the current state call the function with all possible next steps, which calls all its possible next steps and so on. The depth of the recursion ends when the end condition is met, which is when the phrase consists of the number of letters and/or words that have been specified at startup.
The recursive function takes both current phrase and the state as the input, because the state can include spaces and points, while the phrase cannot.
The general pseudo code for this function is as follows:

```
If phrase length is within the desired range → output phrase
If maximum length is reached → return
Else:
    For each possible next step (char/word sorted desc. by no. of occurrences):
        If next step is "_" or "." → change state, let phrase remain the same.
        Else → add the step (the char/word) to the phrase, and update the state.
        Call recursively using new phrase and state
```

Threshold
If the probability for the next step to be a particular character is small, it follows from the multiplication principle that the probability for this combination of characters to appear anywhere in a phrase also is small. By directly ignoring these steps, instead of implementing a threshold for the phrases cumulative probabilities which filter out phrases only when the phrase is complete, the rest of its recursive chain is avoided in

order to obtain higher performance. This implementation of a threshold was tested in this work, but a proposal for future work is to investigate other methods.
The threshold is very dependent on the quality and especially the size of the source text used to produce n-gram statistics as well as the n-gram level. So in this work, finding a reasonable value to use for each source text and n-gram level used have been achieved by testing.

At phrase generation with n-grams on the character level, this value should in general be set higher than on the word level.

Lower Boundary for Phrase Length
In our implementation, because the exact length of a sought password is not always known, there can also be specified a lower boundary for the length of the phrases in addition to the maximum boundary. In this way, it is possible to specify a range of desired characters in the phrases.

Number of Words as Stop Condition
It is also possible to specify the desired number of words in the phrase, and thus provide an additional stop condition for the recursive function in the main algorithm. The motivation for this is that e.g. a company may have a policy that the passphrase must be created from at least four words. If the cracking attempt is intended to be directed toward a passphrase of such a company, and this information about the policy is known, it should be possible to use this information and thereby reduce the overall search scope.

Phrase Termination Using Sentence Breaks
Once the stop condition is met for a phrase, one additional step in the Markov chain is considered whether the phrase can end with a sentence break, and only accept it as a phrase if it can. Without this control, there may be many unnecessary phrases generated that are only linguistically correct up until the stop condition hit, but feels cut off, and are really "meant" to continue. For example, the word 'the', which is the most common word in the English language is not as widely used as the last word of a phrase, but a continuation of the phrase is generally expected after it.

5 Results

5.1 N-gram Statistics

The first part consists, as previously described, in analyzing the source text and create n-gram statistics to be saved to file. To create these statistics, three types of text sources were used. These are available on the website "The Leipzig Corpora Collection" (Universität Leipzig, 1998 - 2015).
The first is text taken from the English Wikipedia in 2010, containing 1 million sentences. The second is text taken from English news sites in 2013, containing 3 million

sentences. The third is text taken from general websites, blogs, etc. 2002, containing 1 million sentences.
Phrases in our tests has been generated based on these three sources, independent of each other, in order to determine what influence the source type has on the results.
Initial visual tests showed that on the word level, 3-grams provided a well-balanced mix of randomness and predictability. Using 2-grams, the number of nonsense phrases were many, and use of 4-grams generated a too small number of phrases, which also were more predictable and similar to the source text. On the word level, only 3-grams has been used in the tests below.

On the character level, corresponding tests were made which showed that 5-gram was the lowest order where reasonably understandable phrases were generated. Since many nonsense phrases after all was generated even in that case, 5-grams has been chosen to be used only when generating shorter phrases, and 8-grams when generating longer.

Because the source text with the text taken from news sites was the one that contained the greatest amount of text, it has been used as a primary source and was used both on the word level and on both character levels to fairly compare the differences between all methods.
In addition, some other n-grams files were created from the other source texts to compare the differences in the use of different source texts.

5.2 Testing Samples

To examine the effectiveness of the implementation, an independent test sample basis was needed. Therefore, a survey was sent out to different persons, where they were asked to create some lowercase phrases with english words (2 or more) assembled without spaces, and deliver them hashed with unsalted NTLM hash.
The distribution of the requested categories of the testing basis became as follows:

Table 1. Number of hashes in testing basis

Phrase length	**Number of hashes**
10	15
14	15
16 (4-6 words)	24
20 (6 words)	12
Total	**66**

5.3 Measurements / Calculations

Mainly, the stream of the generated phrases was directed to file for further processing by HashCat (hashcat.net, n.d.), since the comparisons of the results becomes clearer.
For comparison and an estimate of linguistically correctness of the phrases in the created lists, some calculations were made on each list:

The total number of phrases that should cover the English language according to NIST's interpretation of Shannon's theories of entropy, i.e. the number of possible outcomes, is given by 2^{H_t}, where H_t is the entropy of each phrase length calculated according to NIST's formula. This is the "target entropy".

For each created phrase list, we also define a "potential entropy," which is the entropy that *could* apply to the phrases in the list, based solely on the total number of generated phrases in the list. This is calculated by

$$H_p = log_2(X)$$

H_p is the potential entropy of the list, and X is the size of the set of generated phrases. If this value is less than the target entropy, this means that the phrase list is too small, and it does not cover all possible outcomes. If instead it is larger, it means that there are redundant phrases in the list, that it has more possible outcomes than the English language, and the list is thereby unnecessarily large.

In cases when a cracking attempt using a phrase list is successful, that is: it contains one or more of the searched for phrases, an approximation is made of the linguistic correctness of the list by first calculating the efficiency by:

$$e = \frac{h_c}{h_{tot}}$$

h_c is the number of cracked hashes, and h_{tot} is the total number of hashes/sought passphrases.

To make an estimate of how large the search scope would have to be to crack all of the sought phrases, provided this efficiency, we calculate:

$$X_e = \frac{X}{e}$$

To finally get a comparable value to the target entropy, we calculate the 2-logarithm of that search scope:

$$H_{est} = \ log_2(X_e)$$

H_{est} is referred to hereinafter as the "estimated entropy".
This can be summarized with the formula:

$$H_{est} = log_2\left(\frac{X}{e}\right)$$

This value thus gives an approximation of, and a comparable value, how linguistically correct the phrase list is.
If for example all hashes are broken by a phrase list, then $e = 1$, and thus $H_{est} = H_p$.
If as well the list contains the same amount of phrases as the total possible, $H_p = H_t$ the phrase list is perfect because the target entropy is reached.
If instead half of the hashes are cracked, we obtain $e = \frac{1}{2} = 0.5$, and the phrase list could thus been twice as effective. This would give:

$$H_{est} = log_2\left(\frac{X}{0.5}\right) = log_2(X) - log_2(0.5) = \ H_p + 1$$

If the phrase list would also contain double the number of phrases compared with the English language, thus $H_p = H_t + 1$, and we get $H_{est} = H_t + 2$, meaning that the phrase list could have been $2^2 = 4$ times more linguistically correct.

5.4 Cracking using Phraselists

For the tests in this work different files with phrase lists were created, so the files can then be reused for any additional cracking attempts, and the analysis of the overall effectiveness became simpler and clearer. The results from using these lists are shown in **Table 2** below.

The input data values used for each list can be derived from the selected file names. Example: The file name "L16W4T5N8CWiki" reveals that:

- length of the phrases in the list/file is 16 (L16)
- phrases consists of exactly 4 words (W4)
- the threshold value is set to 5 (T5)
- n-gram file used is 8CWiki, which in turn reveals that:
  - n-grams of order 8 has been used
  - n-grams on the character level has been used (C=char, W=words)
  - text source is the one from Wikipedia (Wiki, Web or News)

5.5 Observations

Phrase Length 10

Since 10 is a "short" length in context of this work, 5-grams was used at the character level. If 8-grams instead would be selected here, only the 3 last characters would change in the Markov process, and it would hardly be any Markov process worthy of the name.

For this length of phrases, it may be interesting to note the difference between word level and character level. When using word level, there cannot with this source text basis be generated more than 15.2 million phrases, even if no filtering of the n-grams occurs (threshold = 0), on character level with some filtering (T=100) there was 54 times more phrases generated.

Examples of how the entropy values in the table can be used for comparisons using calculations on the list on row 3:

If the potential entropy is compared with the estimated entropy, a difference of 24.2 to 21.3 = 2.9 is obtained. To get the efficiency, the calculation $2^{2.9} = 7.5$ can now be done. This means that the list would be able to contain 7.5 times more correct phrases. If instead the target entropy and the estimated entropy are compared, a difference of 24.2 to 21 = 3.2 is obtained. This means that when taking into account that the number of phrases in the file is larger than the target amount, the list would totally be $2^{3.2} = 9.2$ times more efficient with regard to both file size and linguistic correctness.

A further comparison between lists on the word level, is that they cracked the same amount of passphrases although the list with T = 0 contains about 6 times as many phrases. This makes the estimated entropy larger for this list, meaning that it is less effective.

Table 2. The results, using different phrase lists

Phrase List	Time generation	Time HashCat	Cracked hashes	Possible outcomes (English)	Phrases Generated	Target entropy	Pot. entropy	Estimated entropy
Phrase Length 10								
L10T100N5CNews	4.2 h	46 s	7/15	2^{21} (2.1 milj.)	$2^{29.6}$ (825 milj.)	21	29.6	30.7
L10T0N3WNews	1.5 h	1 s	2/15	2^{21} (2.1 milj.)	$2^{23.9}$ (15.2 milj.)	21	23.9	26.8
L10T1N3WNews	16 min	0 s	2/15	2^{21} (2.1 milj.)	$2^{21.3}$ (2.6 milj.)	21	21.3	24.2
Phrase Length 14								
L14T3000N5CNews	1.7 h	6 s	0/15	2^{27} (134 milj.)	$2^{26.4}$ (90 milj.)	27	26.4	-
L14T1N8CWiki	23.0 h	4 min 32s	2/15	2^{27} (134 milj.)	$2^{30.8}$ (1865 milj.)	27	30.8	33.7
L14T1N3WNews	20.2 h	31 s	2/15	2^{27} (134 milj.)	$2^{28.9}$ (505 milj.)	27	28.9	31.8
L14W-5T0N3WNews	24.0 h	45 s	2/15	2^{27} (134 milj.)	$2^{28.2}$ (312 milj.)	27	28.2	31.1
Phrase Length 16								
L16W4T5N8CWiki	7.3 h	31 s	1		$2^{28.8}$ (479 milj.)			
L16W5-6T5N8CWiki	34.7 h	3 min 37 s	0		$2^{30.3}$ (1 355 milj.)			
Total, group 1	42 h	4 min 8 s	1/24	$<2^{30}$ (<1 100 milj.)	$2^{30.8}$ (1 834 milj.)	<30	30.8	35.4
L16W4T0N3WNews	9.2 h	6 s	4		$2^{26.0}$ (67.5 milj.)			
L16W5T0N3WNews	79.5 h	10 min 16 s	1		$2^{29.7}$ (887 milj.)			
L16W6T1N3WNews	58.7 h	1 min 32 s	0		$2^{29.7}$ (851 milj.)			
Total, group 2	147.4 h	11 min 54 s	5/24	$<2^{30}$ (<1 100 milj.)	$2^{30.7}$ (1 806 milj.)	<30	30.7	33.0
L16W5T0N3WWeb	16.6 h	13 s	1		$2^{27.6}$ (199 milj.)			
Phrase Length 20								
L20W6T40N8CWiki	12.8 h	18 s	0/12	$<2^{36}$ (<69 000 milj.)	$2^{27.9}$ (244 milj.)	<36	27.9	-
L20W6T1N3WWeb	52.2 h	2 min 14 s	0/12	$<2^{36}$ (<69 000 milj.)	$2^{29.4}$ (727 milj.)	<36	29.4	-
L20W6T0N3WWeb	960 h	29 min 9 s	1/12	$<2^{36}$ (<69 000 milj.)	$2^{33.0}$ (8 500 milj.)	<36	33.0	36.6
L20W6T1N3WNews	550.5 h	39 min 3 s	1/12	$<2^{36}$ (<69 000 milj.)	$2^{31.0}$ (2 131 milj.)	<36	31.0	34.6

At the character level, a threshold was chosen that causes many nonsense phrases, hence the large number of phrases. On this list a difference is seen between the list's estimated and its potential entropy of 1.1, which means that it contains a large number of correct phrases, about one half as $2^{1.1} = 2.1$, but it also contains many nonsense phrases since the difference between the estimated and the target entropy is large.
The two phrases cracked by the word level lists were included among the seven who were broken by the character level list. Number of unique cracked passphrases amounted to 7 at phrase length 10.

Phrase Length 14

For phrase length 14 it has been selected to compare the word level of order 3 with the character level of order 5 and 8. Number of words per phrase is also introduced here to filter out unwanted phrases.
At this phrase length it is no longer efficient to use 5-grams since the threshold had to be raised significantly to obtain performance, and a lot of real phrases are missed because of this. No hashes were cracked with the 5-gram phrase list.

The list of phrases based on 8-grams, however, was more successful, and two hashes were cracked.

Similarly did the lists on word level crack two hashes each. These were not the same hashes which were cracked by the 8-gram list. However, they were not unique among themselves.

So, a total number of 4 unique passphrases with the length of 14 was cracked.

Phrase Length 16

The criteria on the test basis was that the phrases with length 16 should consist of 4-6 words, why the lists were limited to this. In some cases here, to reduce file sizes, these lists were split into groups of lists that contain phrases with different number of words per phrase. In these cases, the total row is what is most interesting, since the number of sought phrases with respective number of words per phrase is not known.
Interestingly, however, is that in one group of lists, it was the 4 word per phrase list that cracked the largest number of phrases despite the smaller total number of generated phrases. This may suggest that the test group to a greater extent created phrases with only four words.

That the phrases are limited to 4-6 word unfortunately leads to that the target entropy no longer is exactly correct, but it is actually slightly lower than 30.

There was also a comparison with various text sources at the word level with phrases containing five words. The hashes that was cracked by these two lists were unique, which shows that the choice of the source text is clearly influencing.

The phrase that was cracked by the list on character level, was also cracked by the one on word level. Totally there were 6 unique cracked phrases with the length of 16.

Phrase Length 20

At the phrase length of 20, three lists were generated based on word level 3-grams, and one list based on the character level of 8-grams, all with 6 words per phrase for additional filtration since this was one of the criteria on the hashes from the test group. Again, due to this criterion, the target entropy is actually lower than 36.

The threshold values had at first to be increased compared to the generation at length 16, to keep time and file sizes at a reasonable level.

On the two first generated lists it is seen that the number of generated phrases are far below the total target search scope. This is also reflected by that the lists' potential entropy is lower than the target entropy.

When the phrase generation was allowed to work for a while there also were results on the 20-character phrases. The bottom two lines show the successful attempts, and that they took weeks to complete. That the potential entropy is lower than the target entropy indicates that, despite the successful cracks, there were too few phrases generated to even cover all of the 20-character phrases of the English language.

A total of 2 phrases consisting of 20 characters were cracked.

5.6 Efficiency

The different entropy values calculated in the above paragraph is shown in Appendix A, **Fig. 1** in a bar graph for clarity.

The aim is that the bar for the estimated entropy should be as low as possible, but no lower than the bar for the target entropy. All three bars should be as equal as possible. Remember that the entropy axis has a logarithmic scale, so one higher unit on the scale corresponds to a halving of efficiency since the logarithm base is 2.

This chart and the numbers in the results table (**Table 2**) suggests that the most efficient phrase list / file group tested in this work is the File Group 2 for the phrase length 16, with its 2.3 bits difference between target entropy and estimated entropy. This suggests that the best order and level of n-grams to successfully generate a so linguistically correct list of phrases as possible could be word level of order 3. More tests may be needed to verify this.

5.7 Time Consumption

The time consumption of each list generation is shown in a bar chart in Appendix A, **Fig. 2**. Only the time of the actual phrase generation is shown, because that is the dominant part of the total time consumption. The n-gram extraction and the actual cracking times are negligible in comparison.

According to this implementation the lists with n-grams on the word level generally took longer time to generate, but they seemed to also be more effective according to the previous graph.

5.8 Brute-Force Comparison

Since there is no other widely available effective cracking attack on these password lengths, what in this work is called the "estimated entropy" of each phrase list has in Appendix A, **Fig. 3** been compared to the entropy of an exhaustive, or brute-force attack.
It can be seen that the method used in this work for cracking is, although it can be further developed and optimized, far more efficient than the brute-force attack. The scale is logarithmic.

Time is not taken into consideration here, only the total search scope. With GPU support, the speed in guessed hashes per second can be far much higher for a brute-force attack, but this works phrase cracking has a major advantage in intelligence.

5.9 LinkedIn Results

A test on the 2012 leaked LinkedIn hashdump was also made, to see how the method would stand up in the real world. Since this test was not the main objective of this work, basically the same generated phrase lists were used as in the tests above, which of course originally were customized for the test groups hashes (with specified number of words and length). Also, this works implementation only handles lowercase, which is not the case of real world hash dumps.
However, using password statistics from www.adeptus-mechanicus.com on already cracked LinkedIn passwords, one can determine the password length distribution, as well as that about 20% of the passwords are only lowercase. With that information it is possible to estimate the target scope for each phrase length.
So, when using non-customized phrase lists from earlier tests we got these results:

Table 3. Crack results for LinkedIn

Phrase length	Cracked hashes	Target scope (estimated total no. of lowercase passwords)
10	18 269 (11%)	168 000
14	1 878 (10%)	18 300
16	378 (6%)	6 100
20	6 (9%)	68

Note that the estimated target scope above may include non-linguistically correct passwords and phrases not targeted by our previously created files or not at all even targeted by this work.

6 Conclusions

6.1 Conclusions and discussion

It has in this work been investigated if the Markov process is a possible model to represent a language in regard to effectively crack long passphrases.
There were successful attempts to some extent to break these long passphrases, which in itself can be seen as impressive and a good result. However, the goal to get close to Shannon and NIST's definition of entropy for these lengths of phrases was not quite achieved.

Probably the results could be closer to the target entropy after some further experimenting with variables and inputs to the phrase generation. That the type and size of the source text as well as the selected threshold value can have significant effects is seen in many of the tests.

Conclusion must be drawn that this seems to be an efficient way to crack passphrases, since it after all worked to some degree as desired and there is also potential to improve the efficiency even more. Moreover, there is a great lack of available alternative methods to crack as long passphrases that this work covers.

One can also infer that today it is practicable, with some optimization of the implementation as well as the input values, generating phrase lists of phrases up to approximately length 20. At higher lengths than that, file sizes would begin to grow to non-manageable sizes for a standard computer, even if the phrase lists would be completely linguistically correct and without any nonsense phrases.
Of course it is possible to avoid storing these files by either modifying the application and save the phrases with a more advanced and more efficient data structure in a database or the like, or if it is possible to increase the speed of the phrase generation, it would be possible to more effectively direct input to the cracking application.

The results further show that if a password policy would be based solely on passphrases, there should also be a requirement that the phrases should be at least longer than 20 characters long, and as in many other password policies, a requirement to include both lowercase, uppercase, numbers and special characters.

7 Future work

7.1 Performance optimization

Despite the great care taken to make the main program efficient, it is possible to further improve efficiency. Every little time saving that can be made in the main algorithm gives a big difference in the overall result, since the recursive main function is called a huge number of times per run.
One example is that the present application uses much of the data type "string" to represent strings. Since many of these strings have a known expected size, "char arrays" could be used more extensively. This is a simpler data type, for which at initialization can be assigned only so much memory which is needed.

Another interesting area of optimization is examination of the profit that can be made by adapting the program for utilizing GPU processors using OpenCL programming. Even the simple use of conventional threading in the application would improve performance to some extent.

7.2 The Delimitations

This work was limited to working with the English language. The application that was developed is able to handle all languages which use the same alphabet as English, depending solely on the source text file used. Some locality adjustments has to be made in order to include other languages, like including the special characters that are required for the language, eg å ä ö in Swedish.
The work was also limited to only handle phrases composed of lowercase characters. One useful improvement would be to include at least uppercase, but preferably also numbers, and perhaps even special characters. For example, uppercase could be used in beginning of sentences or words. For the application to be useful in real-life contexts, including additional character sets is a necessary measure.

7.3 Experimentation with input values

Of course it is possible to change the input values to the application, such as orders of n-grams, number of words per phrase or phrase lengths, and set them differently to optimize each cracking attempt. The program was built for this, and it is up to the prospective user to experiment to their preference, taste and requirements. To the extent possible, generation of phrases at word level without threshold, or creation of n-gram files from a larger text source, would probably increase efficiency.
A deeper study on the optimal size and type of source text needed to get optimum n-gram files would also be interesting.

An additional area to examine is the implementation of the threshold value. Perhaps it would make a difference if a threshold was defined for a cumulative probability summed over the entire phrase instead of completely filtering out the n-gram that has few occurrences in the source text.
Regardless of choice of implementation for the threshold, a way to calculate which value is optimal for each n-gram file used would be useful, in order to fine tune the threshold without having to test manually.

References

1. BlandyUK. (2015). *Hashcat GUI*. Retrieved May 2015, from HashKiller: http://www.hashkiller.co.uk/hashcat-gui.aspx
2. Bonneau, J. (2012). Statistical metrics for individual password strength. *Twentieth International Workshop on Security Protocols.* Cambridge, UK. Retrieved from http://www.jbonneau.com/doc/B12-SPW-statistical_password_strength_metrics.pdf
3. Bonneau, J., & Shutova, E. (2012). Linguistic properties of multi-word passphrases. *USEC '12: Workshop on Usable Security.* Kralendijk, Bonaire, Netherlands. Retrieved from http://www.jbonneau.com/doc/BS12-USEC-passphrase_linguistics.pdf
4. Brown, P. F., Della Pietra, V. J., Mercer, R. L., Della Pietra, S. A., & Lai, J. C. (1992). An Estimate of an Upper Bound for the Entropy of English. *Computational Linguistics, 18*(1), 31-40. Retrieved from http://www.aclweb.org/anthology/J92-1002?CFID=488028382&CFTOKEN=96498617
5. Garsia, A. (n.d.). *Shannon's Experiment to Calculate the Entropy of English*. Retrieved Mars 2015, from UC San Diego, Department of mathematics: http://www.math.ucsd.edu/~crypto/java/ENTROPY/
6. Gosney, J. M. (2012). Password Cracking HPC. *Passwords^12 Security Conference.* Oslo, Norway. Retrieved 2015, from http://passwords12.at.ifi.uio.no/Jeremi_Gosney_Password_Cracking_HPC_Passwords12.pdf
7. *Hashcat/oclHashcat*. (n.d.). Retrieved May 2015, from Hashcat - Advanced password recovery: https://hashcat.net/
8. Kozlowski, L. (n.d.). *Shannon entropy calculator*. Retrieved Mars 2015, from http://www.shannonentropy.netmark.pl/
9. Ma, J., Yang, W., Luo, M., & Li, N. (2014). A Study of Probabilistic Password Models. *2014 IEEE Symposium on Security and Privacy* (pp. 689-704). Washington DC, USA: IEEE Computer Society. doi:10.1109/SP.2014.50
10. Ma, W., Campbell, J., Tran, D., & Kleeman, D. (2010). Password Entropy and Password Quality. *Fourth International Conference on Network and System Security*, (pp. 583-587). Melbourne, Australia. doi:10.1109/NSS.2010.18
11. MacKay, D. J. (2003). *Information Theory, Inference, and Learning Algorithms.* Cambridge, UK: Cambridge University Press. Retrieved April 2015, from http://www.inference.phy.cam.ac.uk/itprnn/book.pdf
12. National Institute of Standards and Technology (NIST). (2013). *NIST Special Publication 800-63-2 - Electronic Authentication Guideline.* Retrieved from http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-63-2.pdf
13. Open Wall. (n.d.). *John the Ripper password cracker*. Retrieved May 2015, from Open wall: http://www.openwall.com/john/
14. Rao, A., Jha, B., & Kini, G. (2013). Effect of Grammar on Security of Long Passwords. *CODASPY '13 Third ACM Conference on Data and Application Security and Privacy*, (pp. 317-324). Retrieved from https://www.cs.cmu.edu/~agrao/paper/Effect_of_Grammar_on_Security_of_Long_Passwords.pdf
15. Samuel, E. (2012). *LinkedIn Hashdump and Passwords*. Retrieved August 2015, from Adeptus Mechanicus: http://www.adeptus-mechanicus.com/codex/linkhap/linkhap.php
16. Shannon, C. E. (1948). A Mathematical Theory of Communication. *The Bell System Technical Journal*, 379–423, 623–656. Retrieved from http://cm.bell-labs.com/cm/ms/what/shannonday/shannon1948.pdf

17. Shannon, C. E. (1951). Prediction and Entropy of Printed English. *The Bell System Technical Journal*, 50-64. Retrieved from http://languagelog.ldc.upenn.edu/myl/Shannon1950.pdf
18. Shay, R., Gage Kelley, P., Komanduri, S., Mazurek, M., Ur, B., Vidas, T., . . . Faith Cranor, M. (2012). Correct horse battery staple: Exploring the usability of system-assigned passphrases. *SOUPS '12 Proceedings of the Eighth Symposium on Usable Privacy and Security*. Retrieved from https://cups.cs.cmu.edu/soups/2012/proceedings/a7_Shay.pdf
19. Teahan, W. J. (2010). *Artificial Intelligence - Agent Behaviour I.* bookboon.com. Retrieved from https://books.google.se/books?id=4Hy2QzVK1wAC&lpg=PA83&ots=Az7b6aGBm2&hl=sv&pg=PA83#v=onepage&q&f=false
20. Universität Leipzig. (1998 - 2015). Deutscher Wortschatz. *The Leipzig Corpora Collection*. Leipzig, Deutschland. Retrieved Mars 2015, from http://corpora.uni-leipzig.de/download.html
21. Weir, M., Aggarwal, S., de Medeiros, B., & Glodek, B. (2009). Password Cracking Using Probabilistic Context-Free Grammars. *30th IEEE Symposium on Security and Privacy*, (pp. 391-405). Berkeley, CA, USA. doi:10.1109/SP.2009.8

Appendix A – Result charts

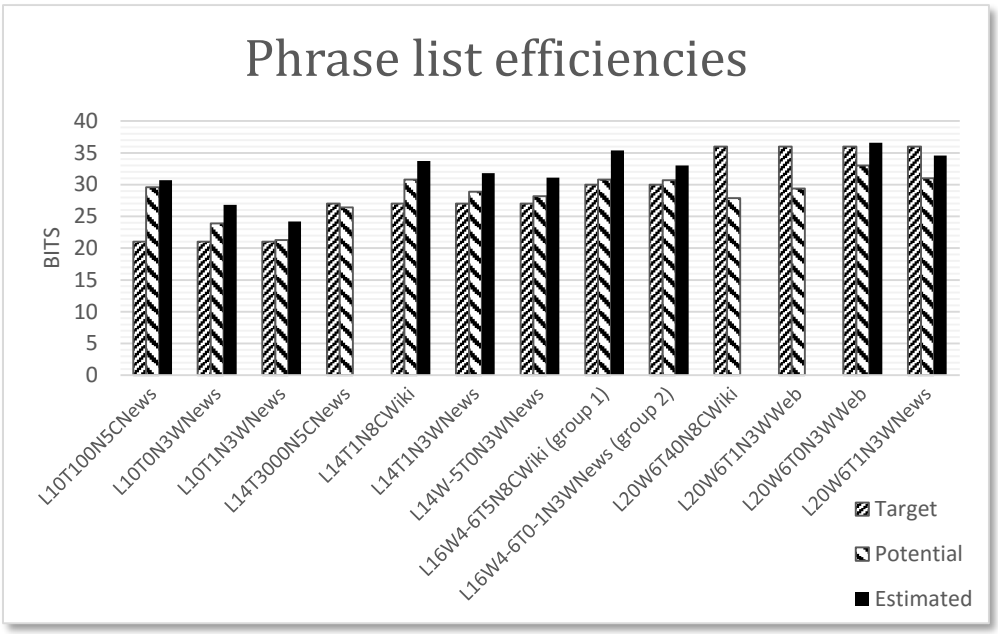

Fig. 1 Graph of phrase list efficiency

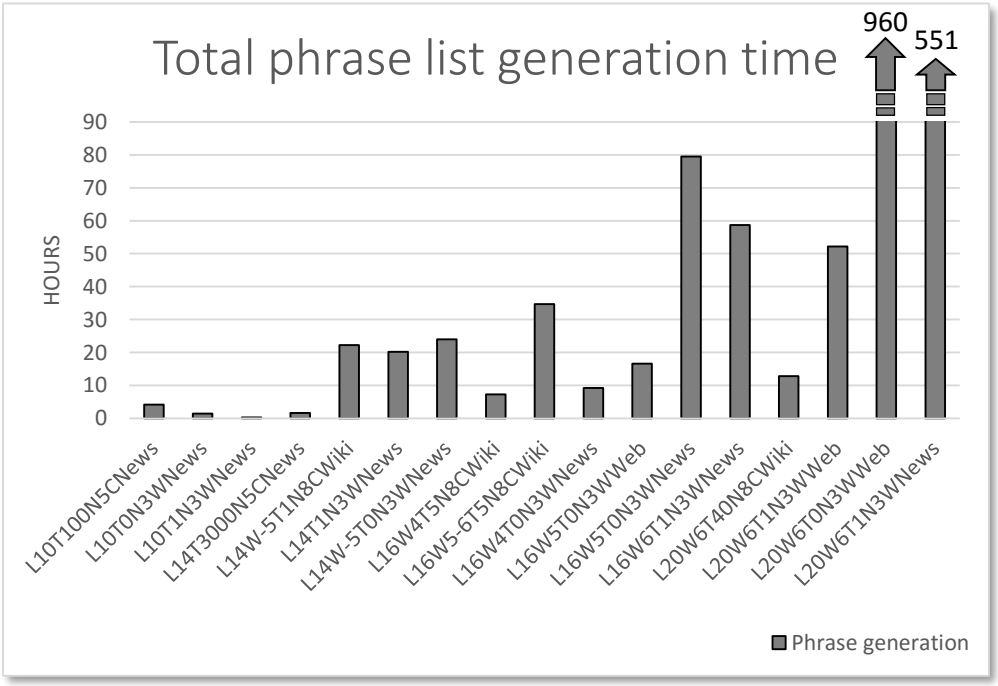

Fig. 2. Chart of time consumption of each phrase list

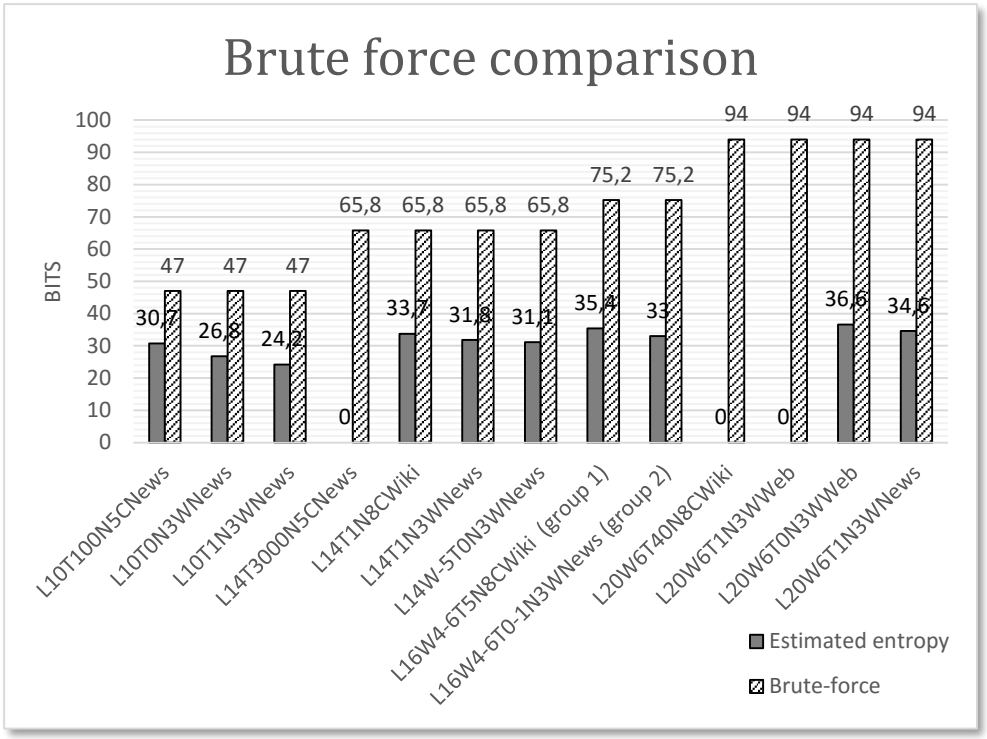

Fig. 3. Chart of the phrase lists efficiency compared to an exhaustive search