# Decoding Running Key Ciphers

**Victor Redko**
260659220

**Marie Payne**
260686859

## Abstract

Artificial intelligence and natural language processing techniques can be used to decode substitution ciphers given the plaintext and keytext follow conventions of the English language. Groupings of n letters for n as large as 6 can be used in decoding methods comprising the Viterbi algorithm and in classifiers such as Multinomial Naive Bayes, logistic regression and support vector machines. We evaluate these methods with a baseline comparison model.

## 1 Introduction

In cryptography, a letter substitution cipher is constructed by inputting a plaintext and a key into a substitution function to produce a ciphertext. The original plaintext can be recovered by inputting the ciphertext and the key into the inverse of the function. A common choice for the substitution scheme is the *tabula recta,* where c = (p + r) mod 26 for same-case letters in the English alphabet, where c is the ciphertext, p is the plaintext and r is the keystream. The function is iterated over every letter in the plaintext, presuming the key is truncated if longer than the plaintext or wrapped around if shorter. If the plaintext or key contain patterns that can be deduced, then they can be decoded through the ciphertext by the means of statistical attacks. In the case that the key is perfectly random, never reused, and kept secret, this results in an unbreakable one-time pad. A variation of the substitution cipher, the running key cipher, uses a stream of characters from, for example, a book to create a long non-repeating key. The plaintext is typically preprocessed to remove occurences of whitespace and punctuation.

Since we know the plaintext and keystream are both obtained from the English language, our goal is to determine the most likely plaintext and keystream belonging to a subset of English that produced the given ciphertext.

## 2 Related Work

N-grams were initially proposed as a method of statistical attack for substitution ciphers and further explored by Bauer and Tate, (Craig Bauer, 2002). Comparisons were made between different values of N up to 6, and tested on ciphertexts of length 1000 produced from the Project Gutenberg corpus with an average accuracy of 30%. This provided a foundation with which to base other methods on, but doesn't produce viable plaintexts as a single concept.

Griffing (Griffing, 2006) employed the Viterbi algorithm, a dynamic programming method for finding the most likely sequence of hidden states or the Viterbi path, in tandem with Hidden Markov models. This method achieves a median 87% accuracy rate on ciphertexts of length 1000. A shortcoming with this work is that it is computationally intensive, with results searching through $26^5$ states at every position in the ciphertext. It also doesn't perform any comparisons between sources or baselines.

Corlett and Penn (Eric Corlett, 2010) produced a method that employs the A* search and Viterbi algorithms to decode various ciphertexts sourced from websites. Since this method is exact, it will determine the plaintext and keystream with a 100% accu-

racy, however it is computationally involved and can lead to exponential time consumption.

Reddy and Knight (Sravana Reddy, 2012) used a method with Gibbs sampling of n-grams up to 6 to produce iterations of probable plaintext/keystream pairs. They tested their methods on the Wall Street Journal and Project Gutenberg corpuses with varying results depending on which source the keytext and plintext came from. On average, their method consistently produced accuracies of 93% on ciphertexts of length 1000. A shortcoming of this work is that it was hard to reproduce given the small level of detail in their paper, unfortunately.

## 3 Methods

In this paper, we implement an n-gram solution and perform tests with Multinomial Naive Bayes, logistic regression and support vector machine classifiers, as well as implement the Viterbi algorithm and compare its performance based on different corpora and ciphertext length used.

### 3.1 Data Preprocessing

Our implementation takes the corpus as a txt input and removes all punctuation and whitespace characters through regex matching before converting to lowercase. Each sentence gets stored as its preprocessed form linked to its raw, human readable form. The spaces need to be rejected in order to properly determine the n-grams of the training corpora.

### 3.2 N-Grams

Each corpora is divided as a ratio of 80-20 training-testing for optimal results. All possible n-gram combinations are computed based on the alphabet, then linked to a probability based on the training source text. These probabilities are then fed into the different classifiers for performance comparison as well as being used in the Viterbi calulations.

### 3.3 Viterbi Algorithm

The Viterbi algorithm is based on a Hidden Markov Model (HMM) representation of the text, where the states represent the letters at each position, the transition probabilities are interpreted as the n-gram frequencies, and the emissions are determined based on those frequencies rather than aligned emission. The most probable 'path' of letters is computed by saving the most probable state sequence for the previously given letters. Both smoothing and unsmoothed HMM results were used for comparison.

## 4 Results

## 5 Future Work

Though the results shown were promising, further comparisons could be done through implementing Gibbs sampling or through a simple supervised learning approach, where the relations between the plaintext and keystream were learned based on the ciphertexts. Since the sample space for ciphertext generation is small, this could be in scope.

## 6 Conclusion

We compared the performance of a sampling of different methods that decode running key ciphers against a baseline random guesser, classifers, and with different source texts for training and testing sections.

## 7 Statement of Contribution

Victor contributed the code for the data preprocessing, n-gram generation, cache, random guesser, classifier tests, HMM and Viterbi algorithms.

Marie contributed the initial proposal, final report, and the code for the argument parsing and data preprocessing.

## References


Christian N.S. Tate Craig Bauer. 2002. A statistical attack on the running key cipher. *Cryptologia*, 26(4):274.

Gerald Penn Eric Corlett. 2010. An exact a* method for deciphering letter-substitution ciphers. *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, 48(1):1040.

Alexander Griffing. 2006. Solving the running key cipher with the viterbi algorithm. *Cryptologia*, 30(4):361.

Kevin Knight Sravana Reddy. 2012. Decoding running key ciphers. *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics*, 50(1):80.