Hiding Secrets in Plain Text

4th year project technical milestone report

by Konrad Komorowski
supervised by Dr. Jossy Sayir
CUED Signal Processing and Communications Laboratory

January 16, 2014

Abstract

The objective of the project is to create a secure steganographic communications system. The system should translate any message into an innocuous stream of English text secured by a private key. Only the knowledge of the key should allow to detect and retrieve the message. A working proof of concept of the system, which does not yet implement secrecy, has been created. The next main areas of focus will be implementation of secrecy and improvement of the language models used in the generation of output text.

1 Aim

The aim of the project is to create a communications system that will allow to broadcast a secret message enciphered with a private key as a paragraph of pseudo-randomly generated English text. The system should also translate this paragraph back into the original message using the same key. The sentences should appear to be randomly drawn from the set of all possible English sentences – in other words will seem innocuous for an enemy monitoring communications between two parties. The introduction of a private key means that the secrecy of the message does not rely on the enemy's unfamiliarity with the system, but on a private piece of information exchanged between the two parties.

2 Overview of steganography

The project falls in the field of *steganography*, which can be defined as follows:

> Steganography is the art and science of encoding hidden messages in such a way that no one, apart from the sender and intended recipient, suspects the existence of the message. [1]

An example stegosystem providing some *innocuousness* and *secrecy* can be constructed in the following way:

> A message, the *plaintext*, may be first encrypted by traditional means, producing a *ciphertext*. Then, an innocuous *covertext* is modified in some way so as to contain the *ciphertext*, resulting in the *stegotext*. [1]

There are various ways in which ciphertext may be embedded in covertext: *a)* by breaking the lines of covertext in such a way that the first words of every line form ciphertext – this works if stegotext is not reformatted in transmission; *b)* by replacing standard characters with their redundant Unicode lookalikes according to ciphertext – this works if stegotext is transmitted using Unicode; *c)* by modifying noise in covertext according to ciphertext – this works if a rasterised version of stegotext is transmitted *exactly* (i.e. digitally, not in print).

The methods outlined above provide only some degree of innocuousness. They largely rely on the enemy's unfamiliarity with the system and may be detected using statistical tests, e.g. identifying unusual distributions of Unicode characters or image noise.

3 The interval algorithm

The interval algorithm [2] provides a means of sampling from an arbitrary stochastic process (target) given an input sequence from a different arbitrary stochastic process (source). The algorithm is very similar to arithmetic coding – it also uses successive refinements of the $[0, 1)$ interval to map sequences.

The sequence mapping algorithm starts by mapping the empty sequence $[\,]$ to the interval $[0, 1)$. For any sequence, its interval is partitioned into subintervals corresponding to the sequence extended by a single term. The size of each interval is chosen to be proportional to the probability of the last term occurring given the preceding sequence. The order of the subintervals does not matter, but needs to be consistent if reverse mapping from an interval to a sequence is to be performed. By the chain rule of probability, the size of the resulting interval is equal to the probability of observing its sequence. Consequently, only sequences of non-zero probability can be mapped. For a more detailed overview of arithmetic coding see [3].

The interval algorithm works by continuously mapping the observed input sequence to successive intervals defined by the stochastic model of the source. In parallel, the intervals are reverse mapped to an output sequence according to the stochastic model of the target. Terms of the output sequence can be generated as long as its interval is a superinterval of the input sequence interval. See Fig. 1 for an example of the procedure.

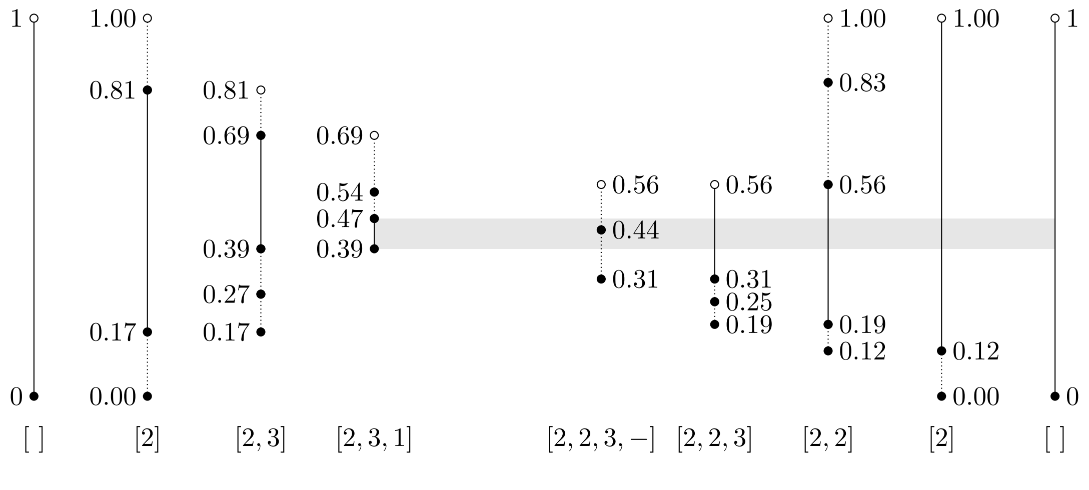

Figure 1: Example run of the interval algorithm. On the left, an input sequence $[2, 3, 1]$ from the source process gets mapped to $[0.39, 0.47)$. On the right, this interval allows to generate up to 3 terms of an output sequence from the target process, i.e. $[2, 2, 3]$, which corresponds to $[0.31, 0.56)$. Note that it is not possible to generate more terms of the output sequence, as neither $[0.31, 0.44)$ nor $[0.44, 0.56)$ is a superinterval of $[0.39, 0.47)$.

4 Stegosystem based on the interval algorithm

The algorithm introduced in section 3 can be used to create a stegosystem as shown below:

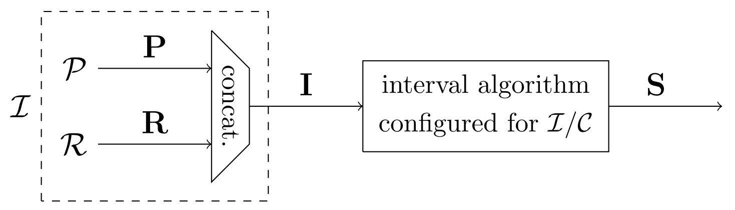

Figure 2: $\mathcal{P}$ is the source of plaintext and $\mathcal{R}$ is any arbitrary stochastic process, for example a sequence of i.i.d. bits. Both $\mathcal{P}$ and $\mathcal{R}$ have known stochastic models. $\mathcal{C}$ is the stochastic model of the covertext. The process $\mathcal{I}$ is the concatenation of the finite sequence output by $\mathcal{P}$ (the secret message $\mathbf{P}$ of length N) followed by the infinite output of $\mathcal{R}$. $\mathcal{I}$ is then used as input to the interval algorithm, generating an output sequence according to $\mathcal{C}$ – the stegotext $\mathbf{S}$ of arbitrary length M. M is chosen to be large enough so that $\mathbf{P}$ can be recovered in full from $\mathbf{S}$.

The stegotext sequence $\mathbf{S}$ is generated according to the covertext source $\mathcal{C}$, so the stegotext is innocuous (i.e. statistically indistinguishable from covertext) by construction. However, the system does not implement any kind of key-based secrecy – upon knowledge of the system, both the enemy and recipient can decode the plaintext secret message $\mathbf{P}$.

The recovery of $\mathbf{P}$ from $\mathbf{S}$ deserves more explanation. Let us denote by $\mathbf{I}$ the input sequence from $\mathcal{I}$ that was used to generate the output sequence $\mathbf{S}$. Let us also denote the intervals corresponding to particular sequences (according to their stochastic models) by lowercase bold serif letters, i.e. $\mathbf{p}$ for $\mathbf{P}$, $\mathbf{i}$ for $\mathbf{I}$, etc.

To generate $\mathbf{S}$, we need $\mathbf{i} \subseteq \mathbf{s}$. If we simply used $\mathbf{P}$ as $\mathbf{I}$, then $\mathbf{p} = \mathbf{i}$. But $\mathbf{s}$ as a superinterval of $\mathbf{p}$ would not be enough to identify $\mathbf{P}$. Consult Fig. 1 and imagine that $\mathbf{I} = \mathbf{P} = [2, 3, 1]$. $\mathbf{i} = [0.39, 0.47)$ would generate $\mathbf{S} = [2, 2, 3]$. However, $\mathbf{s} = [0.31, 0.56)$ could have come from $[0.39, 0.47)$ (i.e. $[2, 3, 1]$), or $[0.47, 0.54)$ (i.e. $[2, 3, 2]$), or in fact any interval mapped from an extension of $[2, 3, 1]$ or $[2, 3, 2]$. This is why we need the process $\mathcal{R}$ to extend $\mathbf{P}$ to $\mathbf{I} = \mathbf{P} + \mathbf{R}$ such that the following inequality holds: $\mathbf{i} \subseteq \mathbf{s} \subseteq \mathbf{p}$ (i.e. keep generating $\mathbf{S}$ using $\mathcal{I}$ until $\mathbf{s} \subseteq \mathbf{p}$).

This can pose another problem – $\mathbf{s}$ might be *too much* smaller than $\mathbf{p}$, i.e. $\mathbf{s}$ may allow us to keep decoding from the model $\mathcal{P}$ past the actual length of $\mathbf{P}$. We need extra information to know when we reach the end of sequence $\mathbf{P}$. This can be achieved by for example storing length N using universal coding at the start of $\mathbf{P}$, or by appending a unique EOF symbol at the end of $\mathbf{P}$.

5 Stochastic process models of plaintext and covertext

Both plaintext and covertext can be viewed as a sequence of atomic elements. The elements can be either letters or full words, extended by punctuation and special symbols such as EOF or start/end of sentence markers, in total comprising the alphabet $\mathbb{E}$. The interval algorithm requires the ability to evaluate the probability of $(k+1)$th element $e_i \in \mathbb{E}$ given the preceding sequence $\mathbf{E}_{kj} \in \mathbb{E}^k$. Taking a frequentist approach to probability, this number can be expressed by the following equation (where the + operator denotes concatenation of sequences):

$$P(e_i|\mathbf{E}_{kj}) = \frac{\text{Freq}(\mathbf{E}_{kj} + [e_i])}{\sum_{i'=1}^{|\mathbb{E}|} \text{Freq}(\mathbf{E}_{kj} + [e_{i'}])} \tag{1}$$

Computing Eq. 1 exactly is difficult because the size of the frequency table increases exponentially with the length of sequence k. Such a model would also easily overfit the data, even if to calculate the frequencies we used the corpus of all sentences ever written. The space complexity of the problem can be reduced by assuming that e_i is independent of symbols more than $n-1$ positions before it, i.e. that the sequence is generated according to an nth order Markov chain. Sequence substrings of length n are commonly called ngrams in natural language processing literature. Eq. 1 can be then simplified to Eq. 2, where $\mathbf{E}_{kj}^{(n)}$ consists of $\min(n, |\mathbf{E}_{kj}|)$ last elements of $\mathbf{E}_{kj}$.

$$P(e_i|\mathbf{E}_{kj}) \approx P(e_i|\mathbf{E}_{kj}^{(n-1)}) = \frac{\mathrm{Freq}(\mathbf{E}_{kj}^{(n-1)} + [e_i])}{\sum_{i'=1}^{|\mathbb{E}|} \mathrm{Freq}(\mathbf{E}_{kj}^{(n-1)} + [e_{i'}])} \tag{2}$$

The frequency table of ngrams will be incomplete even with the above simplification – not every sequence from $\mathbb{E}^n$ will be found in the corpus at least once, even for modest n. As a result, either the numerator or both the numerator and denominator of Eq. 2 can be 0. Both cases pose a problem since each sequence to be mapped to an interval needs to have a well-defined, non-zero probability.

If we want to be able to process any input sequence that is in $\mathbb{E}^k$, we can introduce a *back-off* symbol to the plaintext model. If the next term in the input sequence e_i has an undefined or 0 probability, we choose the subinterval corresponding to the back-off symbol and re-evaluate the probability of e_i using a lower order model. Finally, a 1st order model has to return a valid non-zero probability, or else the term would not be in the alphabet.

To keep generating the stegotext sequence, it suffices to be able to evaluate the probability of at least one term given the preceding sequence. If no terms are available, the decision to back-off the model can be made implicitly. However, in order to increase the variety of possible output sequences, it might be worth introducing a back-off symbol in the stegotext model as well.

5.1 Sources of ngram frequencies for words and letters

The Google Books Team has released a database of word ngram frequencies (up to $n = 5$) that contains a total of 468 billion ngrams from 4.5 million English books. [4] Ngrams include part-of-speech (POS) annotations for words and special markers for the beginning and end of a sentence. They are grouped by years and never span sentence boundaries. The cutoff for including an ngram in the database is ≥ 40 occurrences in the whole corpus.

Mark Davies has compiled databases of word ngram frequencies (up to $n = 4$ and $n = 5$) from the Corpus of Contemporary American English (COCA). [5,6] The ngrams include POS-annotated words and punctuation, but no special markers. Free version of the database contains 1 million most frequent ngrams for each n up to $n = 5$. Paid version contains all ngrams up to $n = 4$, i.e. a total of 155 million ngrams.

Freek Dijkstra has calculated single character frequencies from 120 most popular Project Gutenberg books. [7]

6 Progress made

6.1 Implementation

I have implemented a proof of concept system as described in section 4, i.e. without any secrecy. I used single letter frequencies for the plaintext model, and the free COCA ngrams database for the covertext model. I used " ." as the EOF symbol in plaintext.

The interval algorithm and both plaintext and covertext models were implemented in Mathematica. The COCA database was processed using Python and stored in a PostgreSQL database. Single letter frequencies were stored in the source code.

6.2 Results

An example plaintext "`TERMINATE.`" generates the following stegotext, which successfully translates back to the same input sequence:

> that he was preparing to leave for the airport and the extent of the government and the press

The output sounds English, but lacks features such as start or end of a sentence or punctuation. The stegotext is long due to low plaintext compression rate (here: underestimating the probability of plaintext sequence leads to a small input interval) and low entropy of the stochastic model of covertext (a long sequence is needed to achieve small output interval).

7 Future work

7.1 Secrecy

The next most important aspect of the project is secrecy. The simplest way to address it is to use already developed cryptographic techniques. For example, compress plaintext to a sequence of bits (e.g. using arithmetic coding), encipher the sequence with a secret key (e.g. using a stream cipher), and finally feed in the enciphered sequence to the interval algorithm in order to generate stegotext. If the entropy of the key is higher than the entropy of plaintext, it can be proved that perfect secrecy is achieved.

A more interesting way to encipher plaintext is by doing interval arithmetic. Showing that perfect secrecy can be achieved by for instance shifting the plaintext interval by a certain number could be a novel result in the field of information theory.

After having implemented procedures for secrecy, a measure of it should be developed as well. An interesting concept of deniable encryption could also be pursued – the system given a random key might be able to generate plausible plaintext from stegotext. Different schemes of encryption, such as public-key cryptography, can be explored as well.

7.2 Richer ngram models

The quality of the output can be improved if a richer ngram model is employed. The Google Books Ngrams database contains many more ngrams than the free COCA ngrams database used so far. The former should result in more varied, i.e. natural, stegotext. Higher entropy of the covertext model would decrease expected length of the output sequence. Additionally, start and end of sentence markers should make it possible to produce structured sentences.

Similarly, the single letter frequency plaintext model can be replaced with a better one. One option is to scan the Project Gutenberg books database to extract higher-order letter ngram frequencies. It is also possible to process the Google Books Ngrams database to do the same thing.

7.3 Rate of the system

Rate of the system could be defined and investigated. Han & Hoshi [2] state asymptotic bounds on the performance of the interval algorithm to generate output sequences of finite length. It might be possible to argue that performance of the stegosystem can be bounded by considering the analysis from [2] with the input and output reversed.

7.4 Communicating the length of plaintext

It is necessary to somehow communicate the length of plaintext N in order to be able to correctly decode it. The difference between storing N at the beginning of plaintext (or ciphertext) and appending an EOF symbol at the end of plaintext (or ciphertext) should be investigated. The two methods might result in different performance and/or security of the system.

7.5 More advanced language models

A sentence can be represented not only as a linear sequence of characters of words. For example, a *wh-clause* (i.e. one that uses pronouns such as *what*, *where*, etc.) can be separated by more than a few words, so it can't be adequately represented by ngrams. It is possible to use grammatical rules to parse a sentence into a tree, so that the relationship between particular words is exposed.

The problem is that the tree representation would have to be both parsed and generated in a consistent and linear manner. This is a very interesting problem in the field of algorithms and linguistics, but most likely is outside the scope of this project.

7.6 Finite precision arithmetic coding

Interval mapping in arithmetic coding is usually performed using finite precision arithmetic, because infinite precision arithmetic does not scale well on current hardware. In my implementation I am representing the intervals exactly using rational numbers. I have run tests that show that the algorithm can easily transform a sentence of plaintext into a paragraph of stegotext. However, it will likely not scale to input sizes in the order of a book. Finite precision interval mapping would solve this problem, but also introduce other issues such as complications due to interval rounding. The required changes to the interval algorithm and the stegosystem might be very substantial and outside the scope of the project.

References

[1] Wikipedia. Steganography — Wikipedia, the free encyclopedia. `http://en.wikipedia.org/w/index.php?title=Steganography&oldid=590219361`, 2014. Online; accessed January 13, 2014.

[2] Te Sun Han and Mamoru Hoshi. Interval algorithm for random number generation. *IEEE Transactions on Information Theory*, 43(2):599–611, March 1997.

[3] Thomas M. Cover and Joy A. Thomas. Shannon-Fano-Elias coding. In *Elements of Information Theory*, chapter 5.7, pages 127–130. John Wiley & Sons, Inc., 2nd edition, 2006.

[4] Jean-Baptiste Michel, Yuan Kui Shen, Aviva Presser Aiden, Adrian Veres, Matthew K. Gray, William Brockman, The Google Books Team, Joseph P. Pickett, Dale Hoiberg, Dan Clancy, Peter Norvig, Jon Orwant, Steven Pinker, Martin A. Nowak, and Erez Lieberman Aiden. Quantitative analysis of culture using millions of digitized books. *Science*, 331(6014):176–182, January 2011.

[5] Mark Davies. The corpus of contemporary American English as the first reliable monitor corpus of English. *Literary and Linguistic Computing*, 25(4):447–464, October 2010.

[6] Mark Davies. N-grams: based on 450 million word COCA corpus. `http://www.ngrams.info/`, 2013. Online; accessed October 18, 2013.

[7] Freek Dijkstra. Letter distribution — Exterior Memory. `http://www.macfreek.nl/memory/index.php?title=Letter_Distribution&oldid=4995`, 2014. Online; accessed January 14, 2014.