STEALTHY CIPHERTEXT

A Thesis

Presented to

The Faculty of the Computer Science Department

San Jose State University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

by

Martina Simova

December 2005

ABSTRACT

STEALTHY CIPHERTEXT

by Martina Simova

To ensure the confidentiality of data, it can be encrypted before it is transmitted. However, most data today is unencrypted. As a result, encrypted data might attract unwanted attention, simply due to the fact that it is encrypted. To avoid such attention, we propose a method of converting ciphertext into data that can pass certain automated tests for English text. Out goal is to defeat automated detection methods, and we want to expand the encrypted data as little as possible in the process. Out technique can be considered as a form of steganography.

Table of Contents

Introduction	1
Test for Randomness	3
Simple Methods for Hiding Random Data	5
Sentence Substitution	5
Base-64 Encoding	7
Entropy of Base-64 Encoded Data	9
Weakness of Sentence Substitution and Base-64 Encoding	10
Word Substitution	11
Entropy of Data Generated by Word Substitution	12
An Example of Word Substitution	13
Reversal of the Word Substitution	14
Weakness of Word Substitution	15
Automatic Detection of English	16
Hidden Markov Models	16
Problems That Can Be Solved by HMM	20
HMM Challenges	21
HMM Test for "Englishness"	21
Training the Model	22
Using HMM to Identify English	28
Syntactical Substitution	30

Syntactical Substitution in More Detail 33

An Example of Syntactical Substitution 35

Reversal of Syntactical Substitution 35

Data Expansion 37

Entropy of Transformed Data 38

HMM Test of Syntactical Substitution 39

Summary of Results 40

Conclusion 41

Works Cited 43

List of Tables and Figures

Tables

Table 1.	Base-64 encoding table	8
Table 2.	Dictionaries available for syntactical substitution	38
Table 3.	Entropy of data	40
Table 4.	"Englishness" of the transformed data	40

Figures

Figure 1.	Generic Hidden Markov Model	19
Figure 2.	English text conforming to this pattern passes our HMM test for "Englishness	31
Figure 3.	Pattern employed in our syntactical substitution method	32

"

Introduction

Today, most data is stored or sent unencrypted. Unencrypted data does not attract much of attackers' attention--attackers expect confidential data to be encrypted. Encrypted data (ciphertext), on the other hand, may attract lots of unwanted attention. Because encrypted data looks random but unencrypted data is well structured, attackers can use automated tools to search for encrypted (random) data. Once attackers acquire encrypted data they might attempt to decrypt it. Even if the attackers are not able to decrypt the data, they can still perform traffic analysis.

Therefore, just obscuring data by encrypting it is not sufficient. We need to prevent attackers from knowing that an encrypted communication even occurred. The technique of making an encrypted message "invisible" to the attacker is related to the field of steganography (also known as information hiding). "Steganography is the art and science of writing hidden messages in such a way that no one apart from the intended recipient knows of the existence of the message" [10]. The method we discuss below can therefore be considered as a form of steganography.

To hide the existence of an encrypted message from the attacker we have developed a method that makes encrypted data look like unencrypted text. Our

goal is to avoid automated tools that might filter out random data. Note that we do not require that the text appear to be sensible and grammatical English to a human reader. The process of transformation of encrypted data into "English-like" data naturally results in the encrypted data's expansion. Therefore, we aim to minimize this expansion.

Our initial inspiration for developing such a data transformation method comes from the talk given by "Mystic" at Def Con 11. "Mystic presented an example where he encrypted the sentence 'This is a test' and then processed the ciphertext to produce a long paragraph about baseball. The tool simply used the encrypted bits as a key for selecting snippets of text, while following rules so that the resulting text was somewhat sensible. The process could be reversed by the receiver so that he could reconstruct the encrypted text from which he could recover the plaintext" [6]. The text generated by "Mystic's" method appears as sensible to a human reader. However, the data expansion associated with "Mystic's" method is enormous. Unlike "Mystic's", our goal is to develop a method which will transform encrypted data into data that will pass likely automated tests, while minimizing the expansion of the data. We are not aware of any comparable published approach.

Test for Randomness

To develop an efficient data hiding method, we first have to look at how encrypted data can be automatically detected. An easy and efficient method to detect encrypted (random) data is described in the paper "Playing Hide and Seek with Stored Keys" [5] by Adi Shamir and Nicko van Someren. The entropy of random data is higher than the entropy of nonrandom data and so Shamir's method identifies random data by measuring entropy. To identify random data embedded in nonrandom data, Shamir divides the data into segments, measures entropy of each segment, and then displays the locations of data with high entropy.

Obtaining an exact value of entropy is a complex process. However, the entropy of nonrandom data is significantly lower than the entropy of random data, and so to identify random data, we do not need to know the exact entropy value. By experimenting, Shamir discovered that "examining a sliding window of 64 bytes of data and counting how many unique byte values were used gave a good enough measure of entropy" [5]. Shamir's experiments showed that an average window of random data contains about 60 unique byte values, while an average window of nonrandom data contains only about 30 unique byte values [5].

As mentioned above, the goal of our project is to transform random data (which has high entropy) into a nonrandom looking data (which has low entropy). To measure the success of this transformation, we will use Shamir's method of a "sliding window" for measuring entropy, as we outline below.

1. Get a window of 64 bytes of data
2. Analyze the window by looking at each byte of the window and recording the number of occurrences of each byte value: 0 through 255 (in decimal) in a frequency array
3. Examine the frequency array, and see how many unique byte values occurred in the window
4. Slide the window right by 1 bit (i.e., get a new window)
5. Go to Step 2

We have empirically determined the number of unique byte values per window for random and nonrandom texts using our implementation of Shamir's method for detecting randomness. In our experiments on English texts from Brown corpus [9], a window of 64 bytes contains, on average, about 26 unique byte values, while a window of random data contains, on average, about 58 unique byte values. Thus, to deceive automated tools for randomness detection based on an entropy calculation, we need, at the very least, to transform random data

into data whose average sliding window of 64 bytes will contain about 26 unique byte values.

Simple Methods for Hiding Random Data

We know that to avoid automatic random data detection tools we must lower the entropy of the data. We will now proceed with developing techniques that will transform random data into nonrandom (or less random) data. Random data can be transformed into nonrandom data using various approaches. Each approach is associated with a different degree of random data's expansion and a different degree of "protection" from automated detection tools. Let us first focus on two obvious, simple methods of lowering the entropy:

1. Sentence substitution, i.e., replacing each group of bits of ciphertext with a sentence
2. Base-64 encoding

Sentence Substitution

A simple way of transforming encrypted data into a normal looking text is to replace each group of *n* bits of encrypted data by a full English sentence. Such a

method certainly hides the ciphertext well. The resulting data is a text consisting of English sentences, which is, as we explained earlier, nonrandom and thus has low entropy. The text might look strange to a human because sentences are unrelated to each other. However, using automated tools, it would be difficult to determine that the generated text is actually derived from ciphertext and is not just plain English.

The main problem with the method of sentence substitution is the enormous data expansion. Let us assume, as an example, that we have a dictionary of short English sentences, containing at least 65,536 sentences, each sentence being on average 30 characters long. Having at least 65,536 sentences in the dictionary allows us to replace at most 16 bits of encrypted data with a sentence from a dictionary. (The maximum number of bits *n* that can be replaced by a sentence must be such that $2^n \leq$ dictionary size so that we can recover the ciphertext from the transformed text). By replacing each group of 16 bits of encrypted data with a sentence that is on average 30 characters long, the data size gets expanded by about 1400% of its former size.

Base-64 Encoding

Another method of lowering the entropy of random data is base 64-encoding. Base-64 encoding converts each 3 bytes of data into 4 printable characters. Each base-64 encoded character is represented by only 6 bits, and so we have a total of $2^6 = 64$ printable characters in base-64 encoding.

To perform base-64 encoding, we first group data into blocks of 3 bytes (24 bits), then divide each block into 4 groups of 6 bits. Each 6-bit group can then be represented by a printable ASCII character. The rules for converting groups of 6 bits into printable characters are in Table 1 below.

If the last input block consists of only two bytes, then we append two zero bits to obtain a block of 18 bits. We then convert these 18 bits to three base-64 characters. We must also append an "=" character, so that the final output block will contain four base-64 characters. If the last input block consists of only 1 byte, then we append four zero bits, obtaining a 12-bit block. These 12 bits are then converted into two base-64 characters. To end up with an output block of four base-64 characters, two "=" characters must be appended.

Value	Char		Value	Char		Value	Char		Value	Char
0	A		16	Q		32	g		48	w
1	B		17	R		33	h		49	x
2	C		18	S		34	i		50	y
3	D		19	T		35	j		51	z
4	E		20	U		36	k		52	0
5	F		21	V		37	l		53	1
6	G		22	W		38	m		54	2
7	H		23	X		39	n		55	3
8	I		24	Y		40	o		56	4
9	J		25	Z		41	p		57	5
10	K		26	a		42	q		58	6
11	L		27	b		43	r		59	7
12	M		28	c		44	s		60	8
13	N		29	d		45	t		61	9
14	O		30	e		46	u		62	+
15	P		31	f		47	v		63	/

Table 1. Base-64 encoding table [1]

An example of base-64 encoding:

We would like to base 64-encode these 3 bytes of random data:

1 0 1 0 1 1 0 1	0 1 0 1 0 1 1 1	0 0 1 1 0 1 1 1

The 3 bytes are separated into 4 blocks of 6 bits each:

1 0 1 0 1 1	0 1 0 1 0 1	0 1 1 1 0 0	1 1 0 1 1 1

Their corresponding decimal representations are:

43	21	28	55

These decimal numbers are then, using the base-64 encoding table above, translated into the printable sequence:

r	V	c	3

Since in base 64-encoding each 3 bytes of data are converted into 4 bytes, the data size gets expanded by only about 33% of its former size.

Entropy of Base-64 Encoded Data

When we measured the entropy of base-64 encoded data, we found that the entropy of base-64 encoded data is higher than the entropy of nonrandom data but lower than the entropy of random data. In our experiments, we base-64 encoded nonrandom English texts and measured on average about 36 unique byte values per 64 byte window. We also base-64 encoded random data and recorded on average about 42 unique byte values per window of 64 bytes. Recall that, by our measurements, nonrandom English texts contain about 26

unique byte values per 64 byte window and random data contains about 58 unique byte values per window of 64 bytes.

From these results we see that the value of entropy for base-64 encoded data is not close to the entropy of random data. However, its value is not close to the entropy of nonrandom data either. Due to this distinction, base-64 encoded data might attract an attacker's attention. The attacker might not only try to find random data by looking for data with about 58 unique byte values per window of 64 bytes but he might also be detecting base-64 encoded data by searching for data with either 36 or 42 unique byte values per window of 64 bytes. Once the attacker captures base-64 encoded data, he can easily perform base-64 decoding on them. For these reasons, we concluded that base-64 encoding of data is not a sufficient method of preventing random data's detection by an attacker.

Weakness of Sentence Substitution and Base-64 Encoding

We can see that the two simple methods discussed above for reducing the randomness of data are far from satisfactory. The method of sentence substitution results in good protection from automated detection tools. Unfortunately, this method is associated with an enormous data size expansion.

On the other hand, when using base 64-encoding to transform random data, the data size expansion is very small. However, the base-64 encoded data's entropy is not close to the entropy of normal English text.

Thus, we need a better data transformation method. This better method must minimize the data size expansion. It must also reduce the randomness of the data so that the resulting data can not be detected by measuring entropy. One such method that we have developed replaces *n* bits of random data with an English word.

Word Substitution

One way to make random data look like a nonrandom text is to replace blocks of bits of random data with words from an English dictionary. More specifically, we can take a block of *n* bits of random data, calculate its decimal value *D* and replace this block of random data by the D^{th} word from a dictionary. The size of the block of random data that can be replaced by a word from a dictionary depends on the size of the dictionary used. The larger the dictionary, the larger the value of *n* (the number of bits in a block) can be. For example, if we have a dictionary containing at least 32,768 words, we can replace blocks of size $n = 15$

bits (2^{15} = 32,768), if we have a dictionary containing at least 131,072 words, we can take blocks of size n = 17 bits (2^{17} = 131,072).

Consequently, the amount of data expansion associated with this data hiding method depends on the size of the dictionary used. For example, if we use dictionary of at least 131,072 words, then assuming the average length of a word in the dictionary is 7.4 characters, the data size expansion is by about 295% of its former size (each group of 17 bits becomes, on average, 7.4 bytes plus a space, i.e., 67.2 bits). Similarly, when we use a smaller dictionary for ciphertext hiding, say a dictionary of at least 1024 words, then the blocks of bits of ciphertext to be transformed can be at most 10 bits long, resulting in the data size expansion of 572% of this former size.

Entropy of Data Generated by Word Substitution

To determine the effectiveness of our word substitution method, we have measured the entropy of the hidden (transformed) data. Our measurements show that on average, a window of 64 bytes of data transformed by our technique contains about 24 unique byte values. We concluded that the entropy of the transformed data is very close to the entropy of English data and so it is unlikely that the transformed data will attract any attacker's attention provided

that the attacker is relying only on this one statistical test. (Recall that a window of 64 bytes of structured data contains about 26 unique byte values.)

An Example of Word Substitution

A random data (in hexadecimal):

FE CC 50 EF 5B D1 F5 60 47 E9 D2 4C 65 40 2E 22 A2 76 3B BF

Might get transformed into a text looking like this:

Louvre bandwagon dynasty penurious acerbity malignant glom boson bonito crossroad Mennonite maverick air

The text generated by our word substitution method will look strange to a human because the words are unrelated to each other. However, we are primarily concerned with protection of ciphertext from automated detection tools. Above, we determined that the entropy of the text generated by our word substitution method is close to the entropy of normal English texts. Therefore it would be difficult to determine using automated randomness detection tools that the generated text is actually derived from ciphertext.

Reversal of the Word Substitution

A crucial property of any data hiding method is its reversibility. To recover the data hidden by our word substitution method the receiver of the transformed data only needs to know which dictionary was used to hide the data.

The reversal of our word substitution method above, i.e., the ciphertext recovery, is implemented as follows:

1. Based on the dictionary used to hide the data, determine *n* (the number of bits of ciphertext hidden in each word)

   n = floor($\log_2$ (size of dictionary))

   (where floor(F) is defined as the largest integral value that is less than or equal to F, and $\log_2$ (E) returns the base 2 logarithm of E)
2. Read in a word of the received data
3. Find the word in the dictionary and record its position *P* in the dictionary
4. The value of *P* is a decimal representation of the value of *n* bits, so convert *P* into a binary number of *n* bits
5. If not all words from the transformed message analyzed, go to step 2

When we want to recover only a certain portion of the ciphertext, it is not necessary to decode the entire data. Assuming that each word of transformed data hides n bits of ciphertext, we can recover the b^{th} bit of ciphertext as follows:

1. Determine which word of the transformed data hides the b^{th} bit of ciphertext, i.e., find the w^{th} word of the transformed data such that:

   $(w - 1) \cdot n < b \leq w \cdot n$
2. Read in the w^{th} word of the transformed data, find it in the dictionary and record its position P in the dictionary
3. The value of P is a decimal representation of the value of n bits, so convert P into a binary number of n bits
4. If (b mod n) = 0, then the desired bit b is the n^{th} (last) bit of the recovered n bits, otherwise, the desired bit b is the (b mod n)th bit of the recovered n bits

Weakness of Word Substitution

The word substitution approach defeats the automated tools that search for encrypted data by measuring entropy. Also, the data size expansion associated with this method is reasonable. However, even though the transformed text is a sequence of English words, it does not look like a properly structured English text. That is because the words from the dictionary are chosen only on the basis

of their location in the dictionary and no English syntax rules are followed. Therefore, if the attacker analyzed the transformed data using a tool that takes into account the structure of English, the text generated by word substitution should score poorly. We examine this case below.

Automatic Detection of English

To defeat tools for automatic detection of English texts we need a method that will transform random data into an "English-like" text. To determine the properties that text generated by that method should have in order to be classified as English we developed a tool that measures the "Englishness" of text. Our tool tests for "Englishness" of text using Hidden Markov Models. We chose Hidden Markov Models for their ability to draw out statistically significant information, without requiring many a priori assumptions on the data.

Hidden Markov Models

Markov models are mathematical representations of stochastic processes. Stochastic processes generate random sequences of outcomes according to

certain probabilities. In Markov models, the probability of observing an output depends only on the current state and not on previous states.

A Hidden Markov Model (HMM) is a model in which we observe an output sequence, but we do not know the sequence of underlying states the model went through to generate the observations, that is, the actual states of the model are "hidden". The goal is to recover the state information from the observed data [3]. A Hidden Markov Model is really a statistical tool for understanding some deterministic process, where the deterministic process cannot be observed directly [8].

Notational conventions for HMM:

T = the length of the observation sequence

N = the number of states in the model

M = the number of observation symbols

$Q = \{q_0, q_1, \ldots, q_{N-1}\}$ = the states of the Markov process

$V = \{0, 1, \ldots, M - 1\}$ = set of possible observations

$O = (O_0, O_1, \ldots, O_{T-1})$ = observation sequence

$X = (X_0, X_1, \ldots, X_{T-1})$ = state sequence

Distributional parameters

A = the state transition probability distribution

B = the observation symbol probability distribution

π = the initial state distribution

Where:

The matrix A is an N × N stochastic matrix (i.e., elements of each row sum to 1) A = $\{a_{ij}\}$, such that:

a_{ij} = P(state is q_j at t + 1 | state is q_i at t)

The matrix B is an N × M matrix B = $\{b_j(k)\}$, such that elements of each row sum to 1 and the observation symbol probability distribution in state j:

$b_j(k)$ = P(observation is k at t | state is q_j at t)

The matrix π is a 1 × N matrix $\pi = \{\pi_i\}$, such that elements of its row adds up to 1 and:

π_i = P(state is q_i at 0)

Using the above notation, a Hidden Markov Model is a five-tuple (Q, V, A, B, π) [4]. When the set of possible observations and the states of the Markov process are fixed, then the HMM can be defined by $\lambda = \{A, B, \pi\}$.

Figure 1 shows a generic Hidden Markov Model. The X_i denotes the hidden states. "The Markov process—which is hidden behind the dashed line—is determined by the initial state X_0 and the A matrix. We are only able to observe the O_i, which are related to the actual states of the Markov process by the matrices B and A" [7].

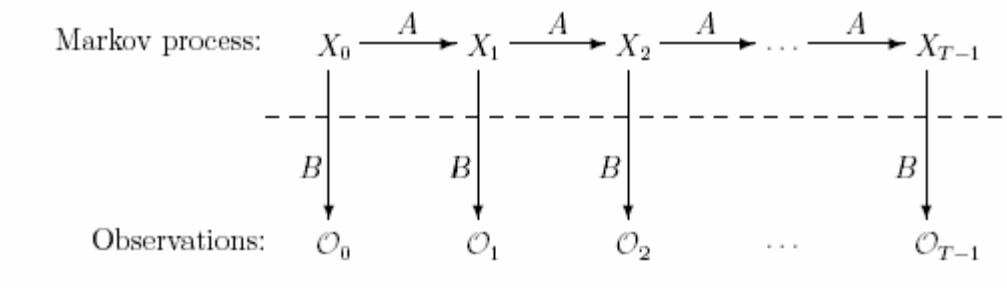

Figure 1. Generic Hidden Markov Model [7]

Problems That Can Be Solved by HMM

Problem 1

We want to compute the probability that the model generates the observation sequence, i.e., given observations $O = (O_0, O_1, \ldots, O_{T-1})$ and model $\lambda = (A, B, \pi)$, we want to compute $P(O \mid \lambda)$.

Problem 2

We want to find the optimal state sequence that generates the observation sequence. This can be thought of as uncovering the hidden part of the model or finding the best "explanation" for the data. Given observations $O = (O_0, O_1, \ldots, O_{T-1})$ and model $\lambda = (A, B, \pi)$, we want find the optimal state sequence $X = (X_0, X_1, \ldots, X_{T-1})$

Problem 3

In this problem, we want to train the HMM to best fit the observation sequence, i.e., given observations $O = (O_0, O_1, \ldots, O_{T-1})$, determine model parameters $\lambda = (A, B, \pi)$ that maximize $P(O \mid \lambda)$.

HMM Challenges

Experiments have shown that HMMs are effective and highly accurate in practice. However, a few challenges must be overcome when using HMMs. “The structure of the HMM is not always obvious” [2]. In order to construct the states and transitions that an HMM can take, we need to have some prior knowledge of the domain. Often, in practice trial and error is used to construct the states and transitions of HMM.

HMMs do not always produce comprehensible results--in some cases, the distribution and transition probabilities do not provide an intuitive descriptions of what has been learned [2]. This problem arises from an incorrect initial HMM "setup". It is dificult to determine the appropriate number of states, transitions and the form of the probability distributions.

HMM Test for "Englishness"

To develop our test for "Englishness" we have to solve two HMM problems--Problem 3 and Problem 1 as defined above. We first use Problem 3 to train HMM on properly structured English texts and thereby obtain a model for English. Once trained, we use Problem 1 to determine how closely a given text

conforms to the model. In other words, to determine whether the given text "looks" like English, from the perspective of our HMM.

Training the Model

The process that we apply for obtaining a model can be described as:

1. Initialize the model $\lambda = (A, B, \pi)$

   A is an N × N matrix, B is an N × M matrix, and π is a 1 × N matrix. All three matrices must be initialized in a way so that the elements of each row add up to 1 and the elements of each matrix are not uniform. The easiest way is to initialize $a_{ij} \approx 1 / N$, $b_j(k) \approx 1 / M$, and $\pi_i \approx 1 / N$

2. Repeat for a desired number of iterations or while $P(O \mid \lambda)$ increases:

i) For $i = 0, 1, \ldots N - 1$ and $t = 0, 1, 2, \ldots, T - 1$ compute:

$$\alpha_t(i) = P(O_0, O_1, \ldots, O_t, X_t = q_i \mid \lambda)$$

The value of $\alpha_t(i)$, which is the probability of the observation sequence up to time t, and where at time t the Markov process is in state q_i is computed as follows:

for $i = 0, 1, \ldots N - 1$

$$\alpha_0(i) = \pi_i \, b_i (O_0)$$

for $i = 0, 1, \ldots N - 1$ and $t = 1, 2, \ldots, T - 1$

$$\alpha_t(i) = [\Sigma_{j=0 \text{ to } N-1} \, \alpha_{t-1}(j) \, a_{ji}] \, b_i (O_t)$$

ii) For $i = 0, 1, \ldots N - 1$ and $t = 0, 1, 2, \ldots, T - 1$ compute:

$$\beta_t(i) = P(O_{t+1}, O_{t+2}, \ldots, O_{T-1} \mid X_t = q_i, \lambda)$$

We compute the value of $\beta_t(i)$, which is the probability of the observation sequence after time t, and where at time t the Markov process is in state q_i as follows:

for $i = 0, 1, \ldots N - 1$

$$\beta_{T-1}(i) = 1$$

for $i = 0, 1, \ldots N - 1$ and $t = T - 2, T - 1, \ldots, 0$

$$\beta_t(i) = \Sigma_{j=0 \text{ to } N-1} \, a_{ij} \, b_j \, (O_{t+1}) \, \beta_{t+1}(j)$$

iii) For $i = 0, 1, \ldots N - 1$, $j = 0, 1, \ldots N - 1$ and $t = 0, 1, 2, \ldots, T - 2$ compute:

$$\gamma_t(i, j) = P(X_t = q_i, X_{t+1} = q_j \mid O, \lambda)$$

The value of $\gamma_t(i, j)$, which is the probability of being in state q_i at time t and transiting to state q_j at time t + 1, is computed as follows:

for $i = 0, 1, \ldots N - 1$, $j = 0, 1, \ldots N - 1$ and $t = 1, 2, \ldots, T - 2$

$$\gamma_t(i, j) = (\alpha_t(i) \, a_{ij} \, b_j \, (O_{t+1}) \, \beta_{t+1}(j)) / P(O \mid \lambda)$$

iv) For $i = 0, 1, \ldots N - 1$, $j = 0, 1, \ldots N - 1$ and $t = 1, 2, \ldots, T - 2$ compute:

$$\gamma_t(i) = P(X_t = q_i \mid O, \lambda)$$

We calculate the value of $\gamma_t(i)$, which is the probability of being in state q_i at time t, as:

for $i = 0, 1, \ldots N - 1$, $j = 0, 1, \ldots N - 1$ and $t = 1, 2, \ldots, T-2$

$$\gamma_t(i) = \Sigma_{j=0 \text{ to } N-1} \gamma_t(i, j)$$

v) Re-estimate π:

for $i = 0, 1, \ldots N - 1$

$$\pi_i = \gamma_0(i)$$

vi) Re-estimate A:

Re-estimated a_{ij}, which is the probability of transiting from state q_i to state q_j is the ratio of the expected number of transition from state q_i to state q_j to the expected number of transitions from q_i to any state. a_{ij} is re-estimated as:

for $i = 0, 1, \ldots N - 1$, $j = 0, 1, \ldots N - 1$

$$a_{ij} = [\ \Sigma_{t=0 \text{ to } T-2} \gamma_t(i, j)\ ] / [\ \Sigma_{t=0 \text{ to } T-2} \gamma_t(i)\ ]$$

vii) Re-estimate B:

Re-estimated $b_j(k)$ is the probability of observing symbol k when a model is in state q_j. This is computed as the ratio of the expected number of times the model is in state q_j with observation k to the expected number of times the model is in state q_j. $b_j(k)$, is re-estimated as:

for $i = 0, 1, \ldots N - 1$ and $k = 0, 1, \ldots M - 1$

$$b_j(k) = [\ \Sigma_{t=\{0,1,\ldots,T-2\} \text{ and O at } t = k} \gamma_t(i)\ ] / [\Sigma_{t=0 \text{ to } T-2} \gamma_t(i)\ ]$$

To obtain a model for English language that can be used in our test for "Englishness", we experimented with an HMM to determine whether English can be distinguished according to some statistics. First we prepared the observation sequence. For training purposes, we used as an input English texts from Brown corpus [9]. We read *T* words from the Brown corpus (where *T* is the length of observation sequence). We restricted our observation symbols to be: noun, verb, adjective, adverb, pronoun, conjunction, interjection, preposition, and period. Therefore we determined for each of the words that we read from the Brown corpus, what word group it belongs to (i.e., noun, verb, adjective,….). We perform this classification by comparing each of the words to their definitions in a dictionary. The implementation of the word classification is not as trivial as it might appear. The dictionary contains the words in their basic forms only. As a result, word forms such as plural, past tenses, 3rd person verbs, -ing verbs, -er, -est adjectives and proper nouns are not listed in the dictionary. Therefore such words must first be converted by our tool to their basic form in order to be found in the dictionary. The result of this clasification is an observation sequence of length T consisting of word types (i.e., noun, verb, adjective,….).

Once we obtain the observation sequence, we use HMM to see if the words in the observation sequence can be partitioned according to some statistics. To obtain sensible, consistent statistics, we experimented with various lengths of observation sequences (T), varied the number of states in the model (N), the

number of observation symbols (M), and the number of iterations needed for the model to converge, as follows:

T = 50,000 to 100,000

M = 8 or 9 (the 9^{th} symbol being period)

N = 2, 3, or 4

iterations = 1,000 to 5,000

We considered the following models:

Model 1: *2 states, 8 observation symbols*

We found that training a model with 2 states and 8 observation symbols does not produce any sensible results. The statistics vary from test to test and as a result such a model is not useful.

Model 2: *2 states, 9 observation symbols*

We found that training a model with 2 states and 9 observation symbols does not produce any sensible results either. The statistics vary from test to test and as a result such a model is not useful.

Model 3: *3 states, 8 observation symbols*

In this model, the observation symbols were divided into the following three hidden states:

State 1: noun

State 2: verb, preposition, adverb, conjunction

State 3: adjective, pronoun, interjection

Model 4: *3 states, 9 observation symbols*

At the end of the training process we found the observation symbols were separated into three hidden states:

State 1: noun

State 2: verb, preposition, adverb, conjunction, period

State 3: adjective, interjection, pronoun

Model 5: *4 states, 8 observation symbols*

In this case, the observation symbols were divided into the following four hidden states:

State 1: verb, adverb, pronoun

State 2: noun

State 3: adjective, interjection

State 4: preposition, conjunction

Model 6: *4 states, 9 observation symbols*

At the end of the training process we found the observation symbols were separated into these four hidden states:

State 1: verb, adverb, pronoun

State 2: noun

State 3: adjective, interjection

State 4: preposition, conjunction, period

We empirically determined that the Models 4 and 6 can be used in solving Problem 1 of HMM--determining the probability of observing a certain sequence, given a model. In the case of our test for "Englishness" the Models 4 and 6 can be used to determine the probability that an observed sequence is English.

Using HMM to Identify English

After training our model on English texts, we can use it to determine the probability that a given text is English. In other words, given the model $\lambda = (A, B, \pi)$, and an observation sequence O, we want to find probability $P(O \mid \lambda)$. The following procedure was used:

1. Prepare the observation sequence, using as input the text that we want to test for "Englishness". The observation sequence is prepared by

classifying each word of the input as belonging to a certain word group (e.g. noun, verb, adjective, …).

2. Initialize the matrices A, B, and π to the values that matrixes A, B, and π converged to during the training of the model being used.
3. Compute $\alpha_t(i)$ as follows:

   for i = 0, 1, … N - 1

   $$\alpha_0(i) = \pi_i \, b_i \, (O_0)$$

   for i = 0, 1, … N - 1 and t = 1, 2, …, T-1

   $$\alpha_t(i) = [\Sigma_{j=0 \text{ to } N-1} \, \alpha_{t-1}(j) \, a_{ji}] \, b_i \, (O_t)$$

4. Compute $P(O \mid \lambda)$:

   $$P(O \mid \lambda) = 1 - \Sigma_{i=0 \text{ to } N-1} \, \alpha_{T-1}(i)$$

We verified, using a variety of texts, that our technique reliably classifies a given text as English or not English. For example, when using Model 4 (as discussed above), which uses 3 states and 9 observation symbols, our test for "Englishness" gives the following results:

- For English texts, $P(O \mid \lambda)$ is, on average, 0.97 and in no case worse than 0.94.
- For texts consisting of English words in random order, $P(O \mid \lambda)$ is, on average, 0.68 and never more than 0.72.

From those results we see that our word substitution method described above is highly vulnerable to this HMM test. We explained earlier that the word substitution method would pass Shamir's test for entropy. It would, however, fail this more sophisticated HMM test which incorporates some of the structure of English.

We will now proceed with developing a ciphertext hiding technique, which transforms encrypted data into a text that can pass the HMM test for "Englishness".

Syntactical Substitution

To transform random data into a more "English-like" text, some English syntax rules need to be followed when choosing a proper word from a dictionary to replace the bits of random data. Thus when developing a method that transforms data into text that looks like English, we first determined what rules and patterns of English language our transformed data should follow. When looking at how English sentences are structured we observed certain patterns. For example, we saw that often in a sentence we have a noun followed by a verb, followed by an adverb. Or we also often found an adjective being followed

by a noun, followed by a verb. We took into account these patterns when designing a technique for transforming encrypted data into an "English-like" text.

We aimed for simplicity of our data transformation method, and so we empirically determined that, when analyzed by our tool for measuring "Englishness", a text passes as English when it consists of words in this order:

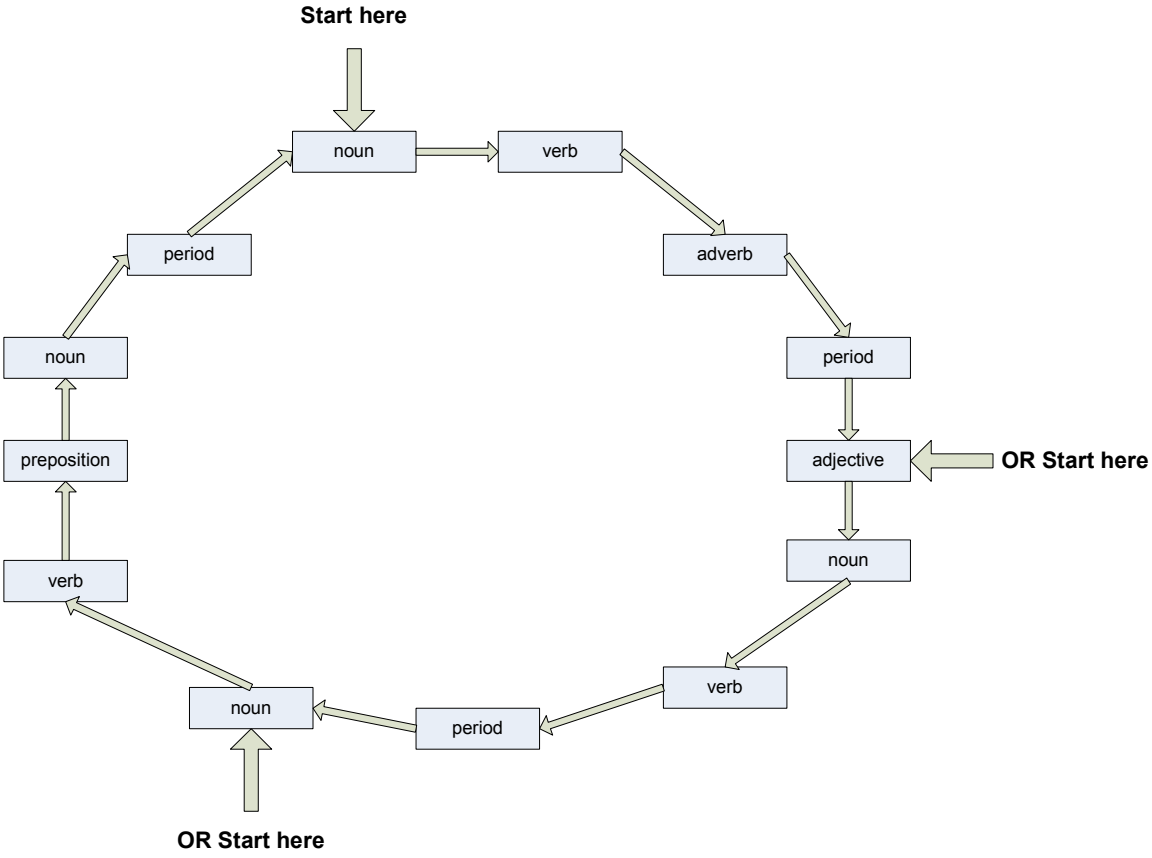

Figure 2. English text conforming to this pattern passes our HMM test for "Englishness"

We employed the chart above to develop a method that transforms random data into "English-like" text by replacing groups of bits in a way that the resulting text follows the pattern below.

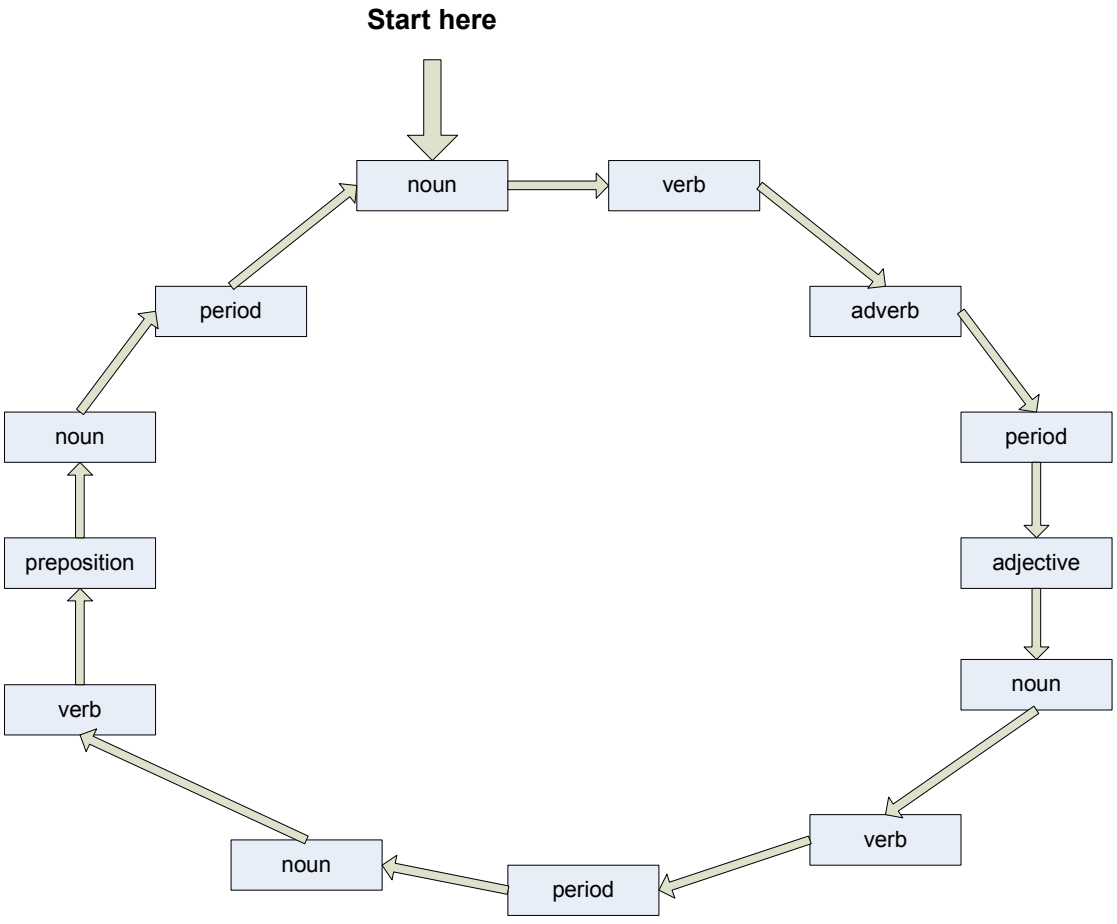

Figure 3. Pattern employed in our syntactical substitution method

That is, our method starts by replacing bits of random text by a noun, if there are still bits of encrypted data left, then it replaces the next group of random bits with

a verb, if there are still bits of encrypted data left, it replaces the next group of bits by an adverb followed by a period, and so on. When transforming random data in this way, we need to be able to find the word of the desired word group in the dictionary. For this purpose, our method uses several dictionaries--each of the dictionaries contains words belonging only to a certain word group, i.e., we have a dictionary of nouns, a dictionary of verbs, a dictionary of adjectives, a dictionary of prepositions, a dictionary of adverbs, a dictionary of pronouns, a dictionary of interjections, and a dictionary of conjunctions.

Syntactical Substitution in More Detail

In this section we show how the ciphertext is transformed into the pattern:

[noun] [verb] [adverb].

1. Read in x bits of ciphertext, where

   x = floor($\log_2$ (size of dictionary of nouns))

   Calculate D = decimal value of those x bits

   Output D^{th} word from dictionary of nouns
2. If unprocessed bits of ciphertext left, read in next y bits of ciphertext, where

   y = floor($\log_2$ (size of dictionary of verbs))

Calculate D = decimal value of those y bits

Output D^{th} word from dictionary of verbs

Else, output period

3. If unprocessed bits or ciphertext left, read in next z bits of ciphertext, where

$$z = \text{floor}(\log_2 (\text{size of dictionary of adverbs}))$$

Calculate D = decimal value of those z bits

Output D^{th} word from dictionary of adverbs

4. Output period
5. Make the first letter of the sentence capital.

The transformation of ciphertext into patterns other than "*[noun] [verb] [adverb].*" is implemented similarly. The number of bits of random data replaced each time depends on the size of dictionary that is being used. For example, if the dictionary of nouns contains E entries, then we can transform n bits of encrypted data into a noun, where

$$n = \text{floor}(\log_2 (E))$$

An Example of Syntactical Substitution

Random data (in hexadecimal) such as:

FE CC 50 EF 5B D1 F5 60 47 E9 D2 4C 65 40 2E 22 A2 76 3B BF

Might get transformed into a text that looks like this:

Inverter cicatrize creamily. Insectile curfew refreshen. Cineole earn ex hemiparasite. Galley glide nohow. Agrarian.

While this text would not pass human analysis, we show below that it does pass both Shamir's entropy test and our HMM test for English.

Reversal of Syntactical Substitution

In order to recover the hidden data, the sender and receiver must agree on which dictionaries are to be used. The receiver of the data must also know the pattern that was used to hide the ciphertext. The decryption method is implemented as follows:

1. Read in a word of the received data
2. Based on the pattern and the dictionaries used, determine, the number of bits of random data *n* that this word hides

3. Find the word in the dictionary for that word type and record its position *P* in the dictionary
4. The value of *P* is a decimal representation of the value of *n* bits, so convert *P* into a binary number of *n* bits
5. If all words from the transformed message have not been analyzed, go back to Step 1

To recover only a certain portion of the ciphertext, it is not necessary to decode the entire data. The b^{th} bit of ciphertext can be recovered as follows:

1. Determine which word of the transformed data hides the b^{th} bit of the ciphertext by adding one by one the numbers of bits each of the transformed words hides (following the pattern), until we find:

   $b \leq$ sum of bits

   (The last word which bits were added into the sum hides the b^{th} bit)
2. Based on the pattern used to hide the ciphertext, determine the type of the word that hides the b^{th} bit, find it in the appropriate dictionary and record its position *P* in the dictionary
3. Based on the type of the word, determine *n* (the number of bits of ciphertext that the word hides)
4. Convert *P* into a binary number of *n* bits

5. If (b mod n) = 0, then the desired bit b is the n^{th} (last) bit of the recovered n bits, otherwise, the desired bit b is the (b mod n)th bit of the recovered n bits

Data Expansion

The amount of data expansion associated with our syntactical substitution method depends on the size of each of the dictionaries used. The larger each of the used dictionaries, the more bits of random data can be replaced by a dictionary word and hence the smaller amount of data expansion. We need to keep this in mind when coming up with new patterns for replacement of random bits. There are, in the English language, far fewer interjections, pronouns, conjunctions, and prepositions than nouns, adjectives, verbs, or adverbs. Therefore, in order to minimize the amount of data expansion, our patterns for random bit replacement should only rarely include interjections, pronouns, conjunctions, or prepositions. See Table 2 for the comparison of dictionaries and the expansion associated with their use in our syntactical substitution method.

Dictionary type	**Dictionary size**	**Average word length (in characters)**	***n***	**Expansion**
Noun	29,271	8.1	14	420
Adjective	10,751	8.7	13	497
Pronoun	72	5.4	6	753
Verb	5,290	7.1	12	440
Adverb	893	7.7	9	673
Preposition	73	4.6	6	646
Conjunction	40	6.6	5	1,116
Interjection	73	4.8	6	673

Table 2. Dictionaries available for syntactical substitution (*n* is the number of bits of ciphertext that can be replaced by a dictionary word. Expansion includes space or period, which is appended to the word during syntactical substitution.)

As a result, our syntactical substitution that hides ciphertext using the pattern in Figure 3 expands data by about 460% of its original size.

Entropy of Transformed Data

To determine whether our syntactical substitution method would defeat automated tools that filter out ciphertext, we measured the entropy (using Shamir's approximation) of the transformed data. Our measurements show that

on average, a window of 64 bytes of data transformed by this technique contains about 25 unique byte values. Recall that a window of 64 bytes of structured data contains about 26 unique byte values. Therefore, the entropy of the transformed data is very close to the entropy of structured data and so it is unlikely that the transformed data would be filtered out by automated random data detection tools based on an entropy calculation.

HMM Test of Syntactical Substitution

To verify the effectiveness of our syntactical substitution method, we used the HMM test for "Englishness" to determine how close the transformed ciphertext is to English. Our tests show that using this method, the probability of the transformed text being English is, on average, calculated to be 0.97, which matches the results we found for legitimate English text. That means that our syntactical substitution technique for a random data transformation produces data indistinguishable from English using either Shamir's measure of entropy or our HMM test for English.

Summary of Results

The tables below summarize our findings related to our methods for ciphertext hiding.

Data	**Entropy**
Random data	58
English text	26
Base-64 encoded random data	42
Base-64 encoded English text	36
Data generated by our word substitution method	24
Data generated by our syntactical substitution method	25

Table 3. Entropy of data (using Shamir's approach of measuring entropy)

Data	**Probability of the transformed text being English (on average)**
English text	0.97
Data generated by our word substitution method	0.68
Data generated by our syntactical substitution method	0.97

Table 4. "Englishness" of the transformed data (using our HMM test for English)

Conclusion

We explained that it is often desirable to hide the existence of an encrypted communication. We then discussed Shamir's entropy approximation, which provides an efficient test to automatically detect ciphertext. This is due to the high entropy of ciphertext as compared to plaintext data. We then discussed a simple word substitution method of converting ciphertext into data with less entropy. This technique would avoid automated screening based on an entropy calculation.

We then presented an approach, based on a Hidden Markov Model (HMM), which was able to defeat the word substitution method. By including English syntactical information into our transformation tool, we were able to defeat this HMM detection tool. That is, our syntactical substitution method converts ciphertext into transformed text that is sufficiently "English-like" to overcome a simple entropy calculation as well as a more sophisticated HMM analysis.

Of course, this is only the beginning of an "arms race". The next step would be to build an analysis tool that can automatically detect that the output of our syntactical transformation tool is not sufficiently "English-like". Then we could attempt to design a more effective transformation tool so that its output would not be detected by this new detector, and so on. However, at each iteration the cost

of detection is likely to be significantly higher that at the previous level. If we can drive the cost up sufficiently high, then we will have made large-scale automated detection impractical.

Works Cited

[1] Base-64 encoding. (n. d.). Retrieved September 24, 2004 from: http://www.betrusted.com/downloads/products/key/tools/v50/crypto/c-docs/html/cryptocdevquide-18.html

[2] Kadous, W. (1998). Hidden Markov Models. Retrieved December 9, 2004 from: http://www.cse.unsw.edu.au/~waleed/phd/tr9806/node12.html

[3] Hidden Markov Models. (n. d.). Retrieved December 9, 2004 from: http://www.mathworks.com/access/helpdesk/help/toolbox/stats/hidden_2.html

[4] Hidden Markov Models. (n. d.). Retrieved December 11, 2004 from: http://www.cs.brown.edu/research/ai/dynamics/tutorial/Documents/HiddenMarkovModels.html

[5] Shamir A, van Someren N. (1999). Playing Hide and Seek with Stored Keys. *Proceedings of the Third International Conference on Financial Cryptography, 1648,* 118-124.

[6] Stamp, M. (2003). DEFCON 11 Trip Report. Retrieved October 5, 2004 from: http://home.earthlink.net/~mstamp1/tripreports/defcon11.html

[7] Stamp, M. (2004). A Revealing Induction to Hidden Markov Models. Retrieved November 5, 2004 from: http://www.cs.sjsu.edu/faculty/stamp/RUA/HMM.pdf

[8] Taitelbaum, B. (2003). Hidden Markov Models. Retrieved December 9, 2004 from: http://occs.cs.oberlin.edu/~btaitelb/projects/honors/honorsnode5.html

[9] The Brown Corpus of Standard American English. Retrieved November 8, 2004 from: http://cs.sjsu.edu/faculty/stamp/brown/

[10] Wikipedia: The Free Encyclopedia. Steganography. Retrieved March 3, 2005 from: http://en.wikipedia.org/wiki/Steganography