Hiding the Hidden
by Mark Chapman and George Davida

Vinodh Jayaram

1 Abstract

The paper presents a system for generating innocent text from a secret message. The system transforms the innocent text back into the secret message at the receiver's end. The paper gives an overview of the transformation processes and tries to demonstrate the quality of the generated innocent text.

2 Motivation

The object of steganography is to deny the very existence of a secret message. Encryption provides protection of the message to be conveyed, but gives away the fact that there is a secret message. An intercepted encrypted message guarantees decrypting attacks! A message concealed in an innocent sounding message however, is more secure, because there is a finite probability that the presence of a secret message is itself not detected. In this case, there is no deciphering attack at all.

Many governments restrict or out-right ban the use of crypto-systems either within the country or in trans-border communications. The use of cryptography is viewed as a threat to their intelligence or counter-intelligence activities. Even democratic governments like the United States have export controls on crypto-systems, and have even tried to prosecute cryptographers.

The paper presents an approach to thwart this censorship by making the presence of ciphertext non-obvious.

3 Basic Operation

The NICETEXT software project transforms cipher-text to text that looks like natural language. Given a piece of generated natural language, it recovers the embedded cipher-text.

The software simulates different "writing styles" either by example or by the use of Context-Free-Grammars. The transformation process selects
"writing styles" independent of the input cipher-text. The reverse-process ignores the style, and relies on a simple substitution mechanism to recover the cipher-text.

4 Hiding the presence of cipher-text

Ciphertext is usually unintelligible - an apparently random stream of data. Analysis of properties of data streams on public networks will detect cipher-text. Hence, all random data becomes suspect.

If some cryptography is allowed, perhaps for authentication purposes, then it is possible to hide information in that cipher-text. When a censor examines the cipher-text, he should be convinced that it is a normal cryptogram used for authentication. But it actually would contain secret information.

If no cryptography is allowed at all, then there are alternative techniques to disguise cipher-text. For example, by changing the format of the file to suggest that it is compressed data. But a censor could expose the cipher-text by attempting to uncompress the file.

The authors' system, NICETEXT, converts cipher-text to "harmless looking" natural language. But how "harmless" it looks depends on the sophistication of the reader. An automated system would probably get fooled, but it is quite possible to discover the input to the NICETEXT system. If it is random, the transmission is suspect.

5 The System: NICETEXT and SCRAMBLE

Given cipher-text C, we need to transform it to text T, which appears innocuous to a censor.

NICETEXT: C -> T is a family of functions that map binary strings to sentences in a natural language.

A particular NICETEXT function is specified by a code dictionary D, and a style source S.

NICETEXT_{D,S}(C) -> T

The system maps binary strings to text using code dictionary D and the style source S. The output will be a set of sentences. How much sense the sentences make depends on the size of the dictionary and the sophistication of the style source.
The inverse of NICETEXT, SCRAMBLE, maps the generated set of sentences back to the input to NICETEXT.

\[ \text{SCRAMBLE}_D(T) \rightarrow C \]

SCRAMBLE only uses the code dictionary, D, and ignores the style information in T.

6 How it is done

Code dictionaries for the NICETEXT system consist of words categorized by type. A style source selects sequences of types independent of the ciphertext. The dictionary will have a set of ciphertext-to-plaintext entries for each type category. NICETEXT transforms ciphertext by choosing the corresponding plaintext in the type category specified by the style source.

The style source defines case-sensitivity, punctuation, and white-space independent of the input cipher-text, and the reverse process simply parses the individual words, and replace them with the corresponding cipher-text in the code dictionary.

A basic example of a NICETEXT function is illustrated - the code dictionary has 2 words and no options for style. The dictionary would be

<table>
<thead>
<tr>
<th>Code</th>
<th>Word</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>ned</td>
</tr>
<tr>
<td>1</td>
<td>tom</td>
</tr>
</tbody>
</table>

Let the style source be undefined. If the ciphertext was 011, then

\[ \text{NICETEXT}_{D,S}(011) \rightarrow \text{nedtomtom} \]

SCRAMBLE, the inverse of NICETEXT, first recognizes 'ned' from its input, and maps ned \( \rightarrow 0 \). Continuing, \( \text{SCRAMBLE}_D(\text{nedtomtom}) \rightarrow 011 \).

A style source could tell NICETEXT to add space between words. This would make the generated text look natural.

The length of the innocuous text \( T \) is always longer than the ciphertext. In the previous example, three bits of cipher-text was transformed into 9 bytes of innocuous text. With spaces between words, that would be 11 bytes. The number of letters per word in the dictionary influences the expansion rate.

With a style source, the codes alone in the dictionary will not be unique, but all (type, code) tuples will be unique.

Let \( D_2 \) be the same code dictionary as above. Let \( S_1 \) be a style source that defines the type as male or female independent of C. Let it be

\[ S_1 = \text{male} \quad \text{female} \quad \text{male} \]
<table>
<thead>
<tr>
<th>Type</th>
<th>Code</th>
<th>Word</th>
</tr>
</thead>
<tbody>
<tr>
<td>Male</td>
<td>0</td>
<td>ned</td>
</tr>
<tr>
<td>Male</td>
<td>1</td>
<td>tom</td>
</tr>
<tr>
<td>Female</td>
<td>0</td>
<td>jody</td>
</tr>
<tr>
<td>Female</td>
<td>1</td>
<td>tracy</td>
</tr>
</tbody>
</table>

NICETEXT$_{D_s,S_1}(011)$ first reads the type from the style source $S_1$, which is male. It then looks in $C$ for the first bit, 0. It then maps the tuple (male, 0) to ned.

Proceeding further, it ends up with **nedtracytom**

A style source increases the credibility of the generated text. Depending on what the generated text is supposed to be, an appropriate style source can be chosen.

Other style aspects which increase the readability and hence, credibility of innocuous text are case sensitivity, punctuation and white space. If the SCRAMBLE function knows to ignore all such frills, the NICETEXT has freedom to generate many more strings, such as

"Jody? Tom? TOM!!!"

The construction of large and sophisticated dictionary tables is key to the success of the NICETEXT system, along with the selection of a compatible, sophisticated style source. A style source $S$ is compatible with a dictionary if all the types in $S$ are found in $D$, and all punctuation in $S$ is not a word in $D$.

7 Software components

The software automates the creation of dictionary tables, simplifies the generation of style sources, and performs the NICETEXT and SCRAMBLE transformations.

A valid code dictionary would be a text file containing (type, word) pairs. A style source could be built as a sentence model. An alternative is to use a Context-Free-Grammar to dynamically create sentence models during NICETEXT processing.

8 Conclusion

The paper lucidly introduces an approach to disguise cipher-text in an innocent-sounding message. The motivation for such a system is explained in detail, and also possible fallacies and shortcomings. The description of
the NICETEXT and SCRAMBLE software is effectively conveyed. The examples used are simple, and on the whole, make the paper very readable even to a lay-person.