4: APPLICATION OF GENERALIZED KATAPAYADI TO STEGANOGRAPHY

4.1 Background

In steganography there have been attempts to automate the process of generating innocuous text to embed sensitive information, but so far not a single fool-proof system is available which generates steganograms that can get past human beings [Bergmair04]. Moreover, it has been hypothesized that a completely automatic system is not possible in the near future [Bergmair05]. The focus of generating steganograms is not only restricted to the construction of grammatically correct sentences, but involves additional requirements of selecting substitutions which are factual, intelligible, consistent with the linguistic ability of normal human beings, and importantly, retain the innocuousness of the text. Because a completely automatic procedure is not feasible, and a manual process is inadequate, it is proposed here that we consider an interactive man-machine environment (IMME) in which a human expert constructs the steganogram and the machine provides complementary assistance in this task.

The research reported here is on linguistic steganography, the heart of which is synset replacement. Unlike conventional models, the katapayadi-based model of linguistic steganography—we call it k-steganography—generates a synset with each of the synonyms representing one and the same state of a message. This feature facilitates the construction of several steganograms to embed a single message. K-steganography follows a well-defined process of encryption (namely, homophonic substitutions using the k-association map). Hence it has a flavor of cryptography. The meta-rules involved in the process are user-defined; they cannot be machine generated. Also the form of the encryption is innocuous, and necessarily involves a human expert in the deciphering stage. In this sense (innocuousness) the encryption is a steganogram.

We demonstrate two schemes for embedding a message in any source language into any other target language. The first scheme is based on basic katapayadi logic for number-to-text embedding. If the message is non-numeric, then preprocessing—numeric enciphering—is needed. (The index of a word in a source-language dictionary is suggested as its possible numeric enciphering) The second scheme is based on generalized katapayadi logic, i.e., k-logic. This involves a suitable generalization of the target language model which treats the building blocks of words
as "characters", and partitions them into "vowels" and "consonants", such that the number of "consonants" of the target language model are two or more times the size of the source language alphabet. (With fewer consonants, it would not be possible to generate homophones of each plaintext symbol.)

It is experimentally shown that user-constructed mapkey-based k-association maps for homophonic substitutions are as secure as randomly generated ones, which are inherently hard to crack. This is a major empirical result of the present research. The k-association maps inherit the usual katapayadi properties such as mnemonic value, simplicity and programmability. An analysis of the proposed k-steganographic scheme shows that it outperforms established techniques in terms of data rate and code breaking strength. It is demonstrated that, to a good extent, stealth as well as error detection and correction are achievable due to human involvement in IMME.

Finally, possible extensions and modifications are suggested for using k-steganography as an enhancement of the schemes reported in the literature, and an approach towards building a fully automated stegosystem is sketched.

4.2 Motivation for K-steganography

A synset—i.e., a word and its synonyms—is an equivalence class. (A singleton word can be treated as a special case: a synset of cardinality 1.) The number of synsets in a language determines the number of "atomic ideas" that can be expressed in any given language. Further reflection hints at a more complex situation: properly speaking, synonyms do not express identical ideas. Rather, they articulate similar but subtly different entities.

Assume that the number of words in a language is fixed. Then we have two broad cases: There is a large number of synsets of small cardinality, or there is a small number of synsets of large cardinality. In the first case a large number of "atomic ideas" can be expressed, and in the second case there are many different ways of expressing the same idea (or similar but subtly different ideas). These properties of any natural language are utilized in linguistic steganography.

As discussed in the previous Chapter, synset-based text replacement is an important part of the design of steganographic systems. A large number of atomic ideas imply many cover domains, and a large number of synonyms mean many cover states for
the construction of a steganogram. The capacity to encode messages is a direct result of the number of cover states. Increased capacity to hold subliminal data makes it possible to introduce subtleties, which increases expressiveness. The larger the number of synonym choices for replacement, the better the quality of steganograms.

Synset size is a parameter that the user should be able to control. Synonymy, hyponymy, etc., are well-established linguistic features which have been traditionally used to generate synsets. Typical synset sizes are small, so we want to increase this number in order to aid expressiveness. This is of course not feasible in conventional dictionaries where the meanings of words are fixed, because if we tried to increase synset size, the differences between the meanings of synonyms would no longer remain subtle, i.e., the "synonyms" would no longer remain interchangeable. However, if we could generalize definitions of the synset-generating linguistic features (examples are given in Section 3.6) we could increase synset sizes.

Synonyms are interchangeable by definition, so we define that the distance between synonyms is 0. The distance function is binary—it evaluates to 1 if two words are not interchangeable. This function should be flexible, so as to restrict or expand a synset according to the required size. Thus there is a need to define a distance function that generalizes the definition of interchangeability. K-logic helps us in achieving this aim, i.e., defining a distance function to generate controlled sets of synonyms, because in the domain of k-logic, "meaning", "synonym" and "synset" are no longer linguistic terms.

4.3 The K-steganographic approach
Generating a k-steganogram involves following steps: Take a sequence of synsets that can suitably embed a message. Make a sentence (or several sentences, depending on the length of the message and the available words) by taking one word out of each synset in the sequence. Label the synset sequence as "valid" if the resulting sentence is semantically well-formed. This forms one steganogram satisfying the user specifications. Note that unlike conventional lexical steganography, here all elements of a synset represent a single state of the message, since the synonyms are images of the same plaintext string. Therefore the mechanism makes it feasible to embed a single message in several distinct steganograms by choosing different synset elements.
The construction of a k-steganogram involves applying k-logic, i.e., the one-to-many katapayadi encoding, to generate synsets and then construct valid sentences which embed the message. One way to construct such innocuous text is to generate all possible valid synset sequences and select suitable sentences by applying heuristic search. The other approach involves the application of heuristics while generating synsets partially, and iterating till a satisfactory solution is reached. The former is computationally intensive and so requires high-performance system resources. The latter involves intelligence and hence needs human intervention to handle the heuristic part. We envision an interactive man-machine environment, with the machine to generate synsets and the human to select words that form the sentences of the innocuous text.

On the other hand, taking advantage of the fact that katapayadi decoding is a many-to-one mapping, it is possible to create an algorithm for decoding a k-steganogram which extracts the subliminal message without any human intervention. Here the mapkey serves as the stego-key.

K-steganography is different from the conventional version in at least one significant way. Normally, the index of the synonym in the synset conveys the subliminal message. That is, synonyms have the same meaning in the domain of the cover, whereas at the subliminal level they are different—it is the particular choice that carries the message. In k-logic, it is the other way around—a synset is made out of fine homophones, i.e., the distance between the synonyms is zero at the subliminal level, while in the cover they have a different meaning. (This is because the distance function is no longer purely linguistic. For example, Rama and Krishna are synonyms as they both map to 52 in the katapayadi scheme.)

While constructing a k-steganogram in a man-machine interactive environment, the human has to look at the machine-generated synonyms as elements for the composition of the cipher text, and choose a sequence of words (each word chosen from a synset) to form the valid innocuous text. During the steganogram construction process the user may state the specifications of other required elements of the cover (e.g., superfluous words) for completing the message embedding. The system generates synsets for these elements also. To generate synsets as well as the other elements we need a dictionary of the ciphertext language.
The homophonic substitution cipher is an important technique in cryptology for encrypting plaintext messages. We have argued in Section 2.4 that katapayadi is a kind of homophonic substitution cipher. Making innocuous forms by adding the necessary superfluous text is a feature of steganography. As mentioned in Section 4.1, from the viewpoint of construction and decoding (i.e., for authentic users) katapayadi encryption is a kind of cryptogram, but from the viewpoint of breaking (i.e., for hackers) this is a steganogram. The human intervention involved in the proposed scheme of message embedding cannot be modeled cryptographically. Hence we prefer the term “k-steganography”.

4.4 Modeling of k-stegosystems

Two approaches to k-steganography are proposed, and design examples to illustrate these approaches are presented. The first uses a variation of the basic katapayadi scheme, while the other uses k-logic, i.e., the generalized katapayadi scheme. For easy reference, call them BK (basic katapayadi) and GK (generalized katapayadi), respectively. A homophonic mapping using the principles of the katapayadi scheme is used in both BK and GK. In addition, the sender and receiver share a common mapkey to the k-association map. The modalities of the two approaches and the results of experimentation are discussed below. First the common attributes of the two are described, and then the differences are highlighted.

4.4.1. Association map for a k-stegosystem

Recall from Section 2.3 that in katapayadi schemes it is necessary to have at least a 1:2 ratio, preferably 1:3 or 1:4, of plaintext symbols to consonants of the cipher text for the construction of the homophonic map. The plaintext and ciphertext languages are determined a priori. What needs to be done is to form distinctive ways of looking at these languages—“models” in other words—so that the ratio constraint is satisfied. This can be achieved in two different ways. In one method, the number of ciphertext consonants is given. If the number of plaintext symbols is larger than about 1/3 of the number of consonants, then some other model of plaintext has to be visualized with a sufficiently smaller symbol set, and the given plaintext has to be denoted accordingly. In the other method, the number of plaintext symbols is given, so we have to visualize a ciphertext model with an expanded consonant set.
Using a k-association-map, homophones of the source symbol sequence—digit sequence in the case of BK—comprising the message and the synsets corresponding to those homophones are generated. As the synsets are in the cipher text language, we have in effect created a steganogram by embedding the plaintext message in the cipher text language.

4.4.2 Construction of a key-based k-association map

The k-association map forms the core of k-steganography. There can be many (user specified) ways to construct a k-association map. For the purposes of steganography, ideally it could be a randomly generated map, but a key-based map is more welcome for its mnemonic value. Below we give a scheme and an example of constructing a key based k-association map. The strength of this scheme is analyzed in Section 4.9.

The table that defines the k-association map contains ciphertext consonants. If these are arranged randomly, there is no clue to help breaking the steganogram. This apparently random association map needs to be key-based, in order that the sender and the receiver can share it.

The ciphertext alphabet is assumed to occur in a lexicographic sequence. To increase codebreaking strength, there is a mapkey that is employed to randomize the lexicographic sequence and a vowel set that the user can change from session to session. The scheme is that the randomization is done by, first, inserting blank spaces in the consonant sequence according to a user-defined heuristic which uses the mapkey; second, placing this expanded sequence in the map table in row-major order; and finally, arbitrarily permuting the columns in the table, again according to a heuristic based on the mapkey.

Example 4.1: One possible blank insertion heuristic is: Blanks are to be added into the conventional consonant sequence as follows: (1) identify a small integer with each vowel (call this the “blank multiplier”); (2) note the consonant prior to every vowel in the key; (3) locate that consonant in the consonant sequence; (4) immediately following this consonant add blanks in the consonant sequence (the number of blanks determined by the “blank multiplier”). The column permutation heuristic used below is: assume that the blank-expanded consonant sequence has been placed in the map table in row-major order; associate the indices \( I = 0, 1, 2, 3, \ldots \), with successive
consonants of the map key; and swap column 1 = 0, 1, 2, 3, ... with the column containing the I-th map key consonant.

Assume that the ciphertext uses the Roman alphabet, the mapkey is “multixfremwash”, and the vowel set is ‘i’, ‘x’, ‘w’, ‘e’, ‘h’ with blank multipliers 1, 2, 3, 4, 5, respectively. With these parameters, the first and last rows in the Table 4.1 give the column indices before and after the permutation (i.e., the columns for digits 0, 1, 2, 3, ... become the columns for digits 8, 7, 1, 0, ...)

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
<td>f</td>
<td>g</td>
<td>i</td>
<td>-</td>
<td>-</td>
<td>j</td>
<td></td>
</tr>
<tr>
<td>k</td>
<td>m</td>
<td>-</td>
<td>-</td>
<td>n</td>
<td>o</td>
<td>p</td>
<td>q</td>
<td>r</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>s</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
<td></td>
</tr>
<tr>
<td>t</td>
<td>u</td>
<td>-</td>
<td>v</td>
<td>y</td>
<td>z</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>7</td>
<td>1</td>
<td>0</td>
<td>9</td>
<td>5</td>
<td>4</td>
<td>2</td>
<td>3</td>
<td>6</td>
<td></td>
</tr>
</tbody>
</table>

Table 4.1: k-association map with mapkey “multixfremwash”

### 4.4.3 Generating synsets from the k-association map

The meta-rule for generating the one-to-many mapping from homophones to synsets is user-defined. More examples of the construction of rules are given below. (The user is cautioned that to the extent that the user choice is arbitrary—depending upon the language model visualized and / or expected size of the synsets etc.—other meta-rules can be widely different from the ones quoted here.)

**Example 4.2:** “Only the first three consonants of a word have decoding value”

The purpose of this rule is to control the synset size. It allows numeric enciphering range 0 to 999, i.e., ideally partitions the dictionary into one thousand parts. The more the number of consonants considered for decoding, the more are the partitions, and vice versa. A larger number of partitions implies reduced synset size. This is obvious, because as the length of the prefix to be matched increases, the number of matches decreases. Table 4.2 below summarizes the result of our experiment with the 30,000-word Oxford Pocket Dictionary.
Table 4.2: Synset size as a function of the number of consonants decoded

<table>
<thead>
<tr>
<th># of consonants decoded →</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of partitions</td>
<td>100</td>
<td>842</td>
<td>3786</td>
</tr>
<tr>
<td>Largest synset size</td>
<td>1287</td>
<td>449</td>
<td>129</td>
</tr>
<tr>
<td>Average size of a synset</td>
<td>303</td>
<td>36</td>
<td>8</td>
</tr>
</tbody>
</table>

Example 4.3: In the katapayadi scheme, normally consonants are "meaning-ful" and vowels are superfluous, but not in this example. With this particular meta-rule the user defines the second and subsequent vowel clusters in a word to be meaningful "consonants", whereas consonants and the leading vowel cluster are superfluous "vowels". Sample homophone pairs, with meaningful symbol sequences in parentheses, are: (1) outside/sunshine (i, e), (2) nowadays/runaway (a, ay), (3) revalue/opaque (a, ue). This meta-rule is a requirement of the language-model visualized for the GK example (Example 4.6) below.

In Section 4.8 below we discuss another meta-rule, namely, assigning new meanings to words, in order to increase synset size.

4.5 Details of the basic and generalized k-steganographic schemes

Typical setups for the BK and GK schemes are described below, and steganogram construction in the two systems is illustrated using examples from our experiments.

4.5.1 Details of the BK scheme

The typical katapayadi map is from digits to consonants. In the case of steganography these are consonants of the ciphertext language. Because the plaintext message language is in general different from digits, we need a "pre-processing" layer for which we propose a one-time pad in the plaintext language. It is proposed that a dictionary or a book can serve as the one-time pad. The sender and receiver share the pad as well as the key to the k-association map.

A sequence is made of the words defined in the dictionary or words occurring in the book and an index is assigned to each word in the sequence. The plaintext message is to be enciphered by indices of the words in the one-time pad. The essential difference between a dictionary and a book is that each word occurs exactly once the former and one or more times in the latter. More indices mean more synonyms in the ciphertext language.
Example 4.4: Preprocessing: The one-time pad is constructed by taking the words in the Pocket Oxford Dictionary (of about 30,000 words), randomizing the word sequence with a pseudo-random number generator (known to both the sender and the receiver and nobody else), and assigning indices to the resulting list of words.

The message is an English-language text (e.g., the single word “slow”). The words in the message are denoted by their indices in the pad (e.g., 04883 for “slow”). Each digit in the index sequence is mapped to homophones according to a k-association map. The map used below has “Lara is an enthusiastic woman” as its mapkey, the characters ‘a’, ‘e’, ‘i’, ‘o’ and ‘u’ as its vowels, and the blank multipliers are 1 for ‘a’ and ‘u’, 2 for ‘e’ and ‘o’ and 3 for ‘i’:

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>k</td>
<td>j</td>
<td>h</td>
<td>p</td>
<td>d</td>
<td>f</td>
<td>g</td>
<td>b</td>
<td>l</td>
<td></td>
</tr>
<tr>
<td>m</td>
<td>r</td>
<td>q</td>
<td>-</td>
<td>v</td>
<td>t</td>
<td>n</td>
<td>-</td>
<td>s</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>y</td>
<td>-</td>
<td>w</td>
<td>-</td>
<td>z</td>
<td>-</td>
<td>-</td>
<td>x</td>
<td>-</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 4.3: k-association map of the mapkey “Lara is an enthusiastic woman”

The homophones for 0, 4, 8 and 3 are {‘c’, ‘m’, ‘y’}, {‘p’, ‘v’}, {‘b’, ‘s’, ‘x’} and {h}.

In this instance, the user-defined meta-rules for embedding the message are:
1. Articles and alternate (2nd, 4th, 6th, ...) words in the steganogram are superfluous.
2. Only the first three consonants of the remaining words are to be decoded.

Synsets are found for the homophones, e.g., Maya for 00 (by assumption the leading 0 is ignored), passionate for 488, etc. Superfluous words are added, according to the meta-rules, to complete a sentence and thus make innocuous text. In Example 4.5 below, several steganograms have been made for each of 4 messages.

Example 4.5:

Slow: Dictionary index 04883. The boldface consonants in the sentences below map to the index 4883 (alternate words are struck off because they are superfluous).

Maya is passionate about Yahoo! (0 0 4 8 8 0 3)

4. Steganography research
Ma' am lost the map at the seashore (0 0 0 4 8 8 3)
The memo is impossible, said Sahu (0 0 0 4 8 8 3)

_Innocent: 18253_

Reuse the wood in couch repair
Erase it with a whitener

_Successful: 31820_

Horse riding is Jimi's hobby
Christmas is joy
Ah! Rosewood's mimic

_It's Bacon: 3412 8846_

Have a cake from Joe, see how spongy it is
Oh! comparatively, your website is expensive
Hi, my move to your type of job was spontaneous

**4.5.2 Details of the GK scheme**

The plaintext language is given, so its symbols are known. In steganography the ciphertext language usually is some natural language. Assume without loss of generality that the plaintext and ciphertext languages are English.

The requirement for generating the homophonic association map is that the number of ciphertext consonants are 3 or 4 times the number of plaintext symbols. This is clearly impossible, given a literal interpretation of terms like "symbol" and "consonant". So we need a more generalized definition of the term "ciphertext consonant".

**Example 4.6:**

The plaintext symbols are 'a' to 'z'. Vowel clusters, i.e., contiguous sequences of vowels, including the semi-vowel 'y', occurring in dictionary words are treated as ciphertext "consonants", e.g., in the word "contiguous", the "consonants" are "o", "i" and "uou". The consonants occurring in dictionary words are treated as ciphertext "vowels" (which, according to the katapayadi scheme, do not carry any information). In a dictionary of some 40,000 words we obtain 101 "consonants" as compared to the 26 symbols of the plaintext.

The consonants are listed in Table 4.4 below.
Table 4.4: 101 Consonants—medial vowel clusters in an English language Dictionary

Quoted from the Section 4.3 above:

“During the steganogram construction process the user may state the specifications of the other required elements of the cover for completing the text embedding”

Accordingly, in this example the user-specified meta-rule is: Only the second vowel cluster in each word of the embedding is decodable. (If there is only one vowel cluster in the word, the word automatically becomes superfluous.) The number of medial clusters in the dictionary is 101, about four times the alphabet size. Therefore we counted only medial vowel clusters in our “consonant” set. (This specification is a slight modification of the user-defined meta-rule mentioned in Example 4.3.)

The message is one word long: “slow”.

The steganogram construction process is as follows:

1. Construct the set of medial vowel clusters (what we have called the “ciphertext consonants”). This is a one-time activity.

2. Generate the homophonic map from the plaintext symbols ‘a’-‘z’ to the “consonants”. (The map depends on the user. In this example the map is given in Table 4.4. For other steganograms, it can be retained or changed.)

3. Obtain the homophones of ‘s’, ‘l’, ‘o’, and ‘w’ as follows:
   - ‘s’: “ioi”, “iou”, “eo”, “oa”
   - ‘l’: “iu”, “e”, “iou”, “eea”
   - ‘o’: “ua”, “oya”, “uyi”, “ayi”
   - ‘w’: “oye”, “oe”, “ayou”, “ieu”

4. Steganography research
4. Apply the meta-rules to generate the synsets. Here, the synsets consist of words with the homophones above as the second vowel cluster. Some of the synonyms—there are very many of them—are given below.

's': moreover, someone, romeo, reload, blackboard, surgeon

'l': wife, wooden, weekend, people, serious, devious

'o': mudguard, casual, casually, manually, visualize

'w': destroyer, employed, monsieur, adieu

5. The superfluous words, which do not carry the message, will have at most one vowel cluster. There are many, many such words. (The ones employed in the present steganogram are: a, was, in, mood, he, took, it, light, drug, and, went, off).

6. The user (i.e., the "man" part of the man-machine interface) has to use his discrimination to generate the innocuous text.

Because of multiple choices available in the synsets, this construction procedure can yield many innocuous texts embedding the same message (namely, "slow").

One particular steganogram to encode "slow" is:

"A surgeon was in a weekend mood. He took it casually, employed a light drug and went off."

There are a total of 18 words in the steganogram. Only four of the words are meaningful, i.e., they carry message information. Each yields one letter of the message. The meaningful words are italicized, and within these words the encoding of each plaintext letter is in bold font.

4.6 Quality metrics

As discussed in the previous Chapter, the performance of a steganographic system is measured by its data rate, stealth, entropy and robustness. The outcomes of our experiments are discussed below.

4.6.1 Data rate

If there are 10,000 words in a colloquial English dictionary, the numeric enciphering of a word is 4 digits long (0000 to 9999). These are the indices of words in the one-time pad. By the procedure for BK adopted in Section 4.5.1 above, one word of an innocuous text embeds 0 to 3 digits (a superfluous word embeds 0 digits, and by the user definition given in Example 4.4, a maximum of 3 consonants are to be decoded in a meaningful word.) Our experiments indicate that because of the superfluous
words needed for the innocuous text, the construction of a steganogram takes about one word per digit. The steganogram would need four words to embed a colloquial word (due to its 4-digit numeric enciphering). This is corroborated experimentally: the length of steganograms was observed to be 4 to 5 times the length of the messages. If the dictionary size was increased by a factor of 10, the numeric enciphering would increase by 1 digit, which would in turn increase the steganogram size by one word. In other words, the data rate, i.e., ratio of the length of the steganogram to the length of the message, is about the same as the (decimal) order-of-magnitude of the dictionary size. Our hypothesis is that steganogram length will increase approximately logarithmically with dictionary size.

Due to the meta-rule used in Example 4.6, the GK steganogram has about as many words as there are letters in the message, not counting the superfluous words required to generate the innocuous text. Our experiments show that the number of words required to generate the steganogram is 4 to 5 times the number of letters in the message. We realistically assume that 4.5 letters is the average word length in a colloquial English text. This yields a figure of 18 to 20 letters in the steganogram for every letter in the message. In the GK example above, the length of the steganogram is 72 letters for a 4-letter word.

4.6.2 Stealth

The examples above demonstrate that a reasonable quality in the construction of innocuous steganogram is achievable by an average user of the language. Since the scheme utilizes human skills in higher order language-constructs like phrases, sentences and paragraphs, there is no danger of over- or undergeneration of the language in the steganographic text. High quality steganograms (which would not easily invite the attention of a professional hacker) can be expected with the participation of skilled language-users in a man-machine interactive environment. The innocuous text is not restricted to any specialized or narrow domain, so a study of the history of the system does not yield any code-breaking hints. Because the distance function for generating synsets—and in general the manner of embedding the message in the steganogram, as demonstrated in the examples above—is user-defined, it is hard for unauthorized persons to predict the distribution of clues.
4.6.3 Entropy
Words are tweaked in order to embed a message in a cover. Recall, as discussed in the Section 3.8.2, that entropy is a measure of the change in the naturally-occurring frequency of words in the text as a result of tweaking—the higher the change in frequency, the lower the entropy. From the research reported earlier [Atallah01b, Chapman02, Wayner Mimic97, Winstein Lsteg99] we note that 4 to 5 words are selected for tweaking in a cover of 100 to 200 words. This degree of alteration in the text does not significantly disturb the default linguistic properties of the text and provides sufficient entropy.

K-steganography does not work on pre-existing text, but generates the text of the steganogram in the process of embedding the message. Therefore the entropy metric discussed above cannot be directly applied to it. In our calculations of the data rate it is observed that in general a BK-steganogram of length 5 to 7 words would contain a one-word message. Assuming an average length of a dictionary word to be 6.5 characters (compared to 4.5 characters for colloquial words), a GK-steganogram would take 25 to 30 words to embed a one-word message. However, this figure can vary depending on user-defined meta-rules: it is under user control to generate text that is consistent with default linguistic properties of the ciphertext language and to maintain entropy as required.

4.6.4 Robustness
Recall from Section 4.3 that k-steganogram construction involves cryptography in the application of k-logic for map-based development of synsets, and steganography in the selection of synonyms by humans for the construction of innocuous text. Hence robustness analysis of k-steganography also involves both cryptanalysis and steganalysis. In the former, we study the code-breaking strength of the k-association map against brute-force attacks, i.e., the difficulty of finding out the correct k-association map used in a given steganogram. In the latter, we consider the effect of intended malicious attacks on the innocuous text. Since the k-steganography approach is similar to that of the well-known Baconian cipher, a comparison of the robustness of the two schemes has been attempted below.

(Digression: The Baconian cipher is described for the sake of completeness: It is a historically known stegosystem that provides a meta-level formula or grammar to
generate innocuous messages for the given subliminal messages [Wikipedia BC, Ucf BC03]. They do not have the constraint of a one-to-one mapping between the cover and the subliminal domains. The user dynamically constructs the natural language cover for the message to be communicated. Here each character to be used in the subliminal message is mapped to a binary string consisting of, say, “0” and “1”. The alphabet of the cover message is then partitioned into two subsets. In the encoded subliminal message, any occurrence of a “0” is denoted by a character from the first subset, and of a “1” by a character from the second subset. For example, if the two subsets are a ... k and l ... z, then the subliminal message “Its Bacon” is encoded by the string “010000 10010 10001 00001 00000 00010 01101 01011” and can be communicated using the cover “I like the new bag. We had the bag kids get Nana less.” [Ucf BC03])

Finding the association map is the essence of codebreaking as far as cryptological part of k-steganography concerned. Brute force attacks involve trying all possible combinations in the absence of the key. Decision attacks, on the other hand, exploit some clue or knowledge about the cryptoscheme. If the key-based map is random, then there is no choice other than a brute force attack. We show that the scheme provided below generates a key-based k-association map whose performance is measured to be close to that of a random map. Therefore we can draw the conclusion that breaking a k-steganogram necessarily involves a brute force attack. The effort involved in a brute force attack is computed in Section 4.11 below in terms of a combinatorial analysis of the k-map.

The possibility of error detection and correction in k-steganography is related to robustness analysis. This is discussed next.

4.7 Error analysis

Errors are possible because of two reasons: either a bit or a letter gets altered due to noise in the communication medium, or an intruder deliberately modifies a steganogram.

A change in a steganogram that leads to an occurrence of an unexpected word or string is a definite indication of an error. The innocuousness of the steganogram gets disturbed and hence the error is detected. Examples include a string which is not part of the user dictionary or a word which gets decoded to something outside the message.
space. A suspicious authentic user can either check the possibility of corrections using close matches in the dictionary or message space or re-request the steganogram.

**Example 4.7:** (unexpected word in a steganogram indicating an error)

(i) Maya is **fassionate** about Yahoo! → (0 688 3) = index of “potable”

(ii) Ma’am lost the **gap** at the seashore → (0 748 883) = indices of “bedroom”, “slow”

In the first example, the dictionary support would suggest two possible spelling corrections for the error string “fassionate”, namely, “fascinate” and “passionate”. They get decoded to indices of the words “lieutenant” and “slow” respectively. The receiver has to match these words to the expected elements in the message space and select the suitable one. (Here the cover word is “passionate”, and the message is “slow”.) In the second example “gap” is the erroneous word. The dictionary support cannot be of help in this instance. The receiver has to get a clue only after decoding takes place. A suitable correction can be suggested by replacing the “gap” in the steganogram with the homophones “map” or “cap”, which both lead to the message “slow”.

If an alteration—whether accidental or deliberate—does not introduce an observable error and thus retains the innocuousness of the steganogram, chances are high that an intended user will be misled. In the examples below the message embedded in the BK-steganograms is “slow”. Correctly constructed steganograms for this message have been given in Example 4.5 above. The steganograms below do not lose the innocuousness of form in spite of the introduction of an error. (The errors are underlined, the syllables to be decoded are in boldface, the numeric deciphering of the message is in parentheses and the message uncovered by decoding of the steganogram is in brackets.)

**Example 4.8:**

<table>
<thead>
<tr>
<th>Steganogram</th>
<th>Deciphering</th>
</tr>
</thead>
<tbody>
<tr>
<td>(i) Saya is passionate about Yahoo! → (80 488 3)</td>
<td>(lycra, slow)</td>
</tr>
<tr>
<td>(ii) Ma’am lost the tape at the seashore → (0 548 83)</td>
<td>(snow, slow)</td>
</tr>
<tr>
<td>(iii) The memo is impossible, said Sasi → (0 488 8)</td>
<td>(walkover)</td>
</tr>
<tr>
<td>(iv) Maya is particular about Yahoo! → (80 415 3)</td>
<td>(lycra, refulgent)</td>
</tr>
<tr>
<td>(v) Ma’am lost the map at the road → (0 4 8 15)</td>
<td>(duress)</td>
</tr>
<tr>
<td>(vi) Ma’am lost the road at the seashore → (0 15 883)</td>
<td>(swan)</td>
</tr>
</tbody>
</table>
(vii) The memo is not possible, said Sahu → (0 65 85) ≡ [sheaf]
(viii) Maya loves Yahoo! → (0 3) ≡ [skip2]

The error in examples (i) to (iii) of Example 4.8 is due to a small alteration (of one or two letters). A suspicion is raised because these embeddings, which are supposed to yield the same message, are generating indices to three different words in the plaintext dictionary. A clue for correction is obtained by observing the portions of the numeric enciphering that are identical (488 in this example). Comparing (i) and (ii) one can guess that the error is in the part of steganogram representing the first word. Comparing with (iii) one can get a clue that the message could be a single word and so the word “slow” can be inferred.

In examples (iv) and (v) a word that carries message information is selected for replacement. Examples (vi) to (viii) are cases where a meaning-preserving form is generated as a modified steganogram. The examples demonstrate that any small change in a BK-steganogram leads to a radically different value after the decoding. Assuming that the sender constructs the message dynamically, the receiver can detect the error if the received message does not semantically match any of the expected messages. Error correction is possible, depending on the receiver’s judgment.

The examples above demonstrate that it is possible to avoid mistakes by employing classical techniques of error recovery, e.g., sending the data redundantly by composing several steganograms with the same message. One outcome of the redundancy is that the probability of introducing the same error in all redundant forms is very small. The cost of generating redundant steganograms is an overhead, but the payoff is the suspicion raised due to inconsistency in the received data. It is obvious that the greater the redundancy employed, the higher is the cost, but the chances of error detection are also higher.

Examples 4.7 and 4.8 above can be studied for error analysis of GK-steganograms also. For GK there is no restriction a priori on the plaintext language. Therefore if we assume without loss of generality that the plaintext is in the form of decimal numbers and the ciphertext is in English, then all the points discussed above for BK-steganogram are equally applicable to GK-steganograms.
Some simpler techniques can help in countering malicious attacks that retain innocuousness of the steganogram. For instance, we can design a message space where the different messages are sufficiently apart (i.e., there is sufficient differentiation in the character sequences that form the various messages). In this situation automatic error detection and recovery is possible to some extent. This would involve identifying close substitutes for the suspect data by a lookup in the message space.

For example, define the "distance" between two messages (their numeric enciphering in the case of BK-steganography) to be the number of characters (digits) that are different. If we maintain the minimum distance between any two messages at 3, then one-letter errors can be automatically corrected. In this sense, messages which are one character apart are homophonic. In our example, where the index "04883" indicates the message "slow", messages with indices *4883, 04*83, 048*3 and 0488* are homophones of "slow". ("*" is a wildcard for digits.) These encipherings are not allowed to represent different messages in the message space. The only different messages in the message space could be those for which the enciphering has at most two common digits, e.g., 04554 or 01890 as compared to 04883. A two-letter error in this case may not suggest a unique correction but a small set of possibilities from which a human expert can make a reasoned guess.

The choice of a bigger domain for the message space could be another effective method of error detection. Acceptable messages would be small in number and scattered in the domain space, so the chance would be high that an error results in an unexpected outcome. Suppose there are N synonyms that can replace a word in the steganogram while retaining its innocuousness and n out of these N choices get decoded to elements of the message space, then the probability that a deliberate attempt to introduce an error will mislead the intended user by generating a wrong message is (n-1) / N.

4.8 Overloading the superfluous structure

Meta-rule for the overloading of the superfluous structure can be seen as an advanced feature of k-steganography. This has two-fold benefits: increase in synset size and security.
An effective increase in synset size is possible by making meta-rules for operator definition. For instance, as in the *Vamiya suktam* example of Section 2.2, "sya" has traditionally meant "multiply". In a similar manner, we can overload the superfluous structure of a steganogram with new semantics. The new definition would be an added meta-rule. Examples of such meta-rules are listed in Table 4.5 below.

<table>
<thead>
<tr>
<th>No.</th>
<th>Overloaded Structures</th>
<th>New semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>and, to sum up</td>
<td>Add</td>
</tr>
<tr>
<td>2</td>
<td>lost, forgot, &quot;-&quot;</td>
<td>Subtract the next operand from the following one</td>
</tr>
<tr>
<td>3</td>
<td>about, of</td>
<td>Multiply</td>
</tr>
<tr>
<td>4</td>
<td>reduce, &quot;/&quot;</td>
<td>Divide</td>
</tr>
<tr>
<td>5</td>
<td>went up, climb, ride</td>
<td>Exponentiation. The left operand (what “went up”) is the exponent and the right operand is the mantissa</td>
</tr>
<tr>
<td>6</td>
<td>comma, &quot;,&quot;</td>
<td>Sequence</td>
</tr>
<tr>
<td>7</td>
<td>said, &quot;:.&quot;</td>
<td>Concatenate</td>
</tr>
<tr>
<td>8</td>
<td>reuse</td>
<td>Double</td>
</tr>
<tr>
<td>9</td>
<td>&quot;!&quot;</td>
<td>Negate or complement (unary minus)</td>
</tr>
</tbody>
</table>

**Table 4.5: Meta-rules to overload superfluous structures with new semantics**

Note that using the new semantics given in Table 4.5 above, one can generate many representations (i.e., homophones) of a numeric figure simply by applying arithmetic. This also makes it convenient to represent very small and very large numbers.

The three steganograms denoting number 4883 in the example in Example 4.5 of Section 4.5.1 may not continue to denote the same number. If the new semantics is applied they may get deciphered to different values. Conversely, in addition to their originally available homophones they have got new representations because of the new semantics.

**Example 4.9:**
1. Maya is passionate about Yahoo!
   With superfluous structure: Maya is passionate about Yahoo! yields \(0 \ 0 \ 4 \ 8 \ 8 \ 0 \ 3\)
   If “about” and “!” stand for multiply and negate: the sentence yields \(488 \times (-3) = 1464\).

2. Ma’am lost the map at the seashore.
With superfluous structure: Ma' am lost the map at the seashore yields (0 0 0 4 8 8 3)
If “lost” stands for subtract: we get 883 - 4 = 879.

3. The memo is impossible, said Sahu
With superfluous structure: The memo is impossible, said Sahu yields (0 0 0 4 8 8 3)
If “said” stands for concatenate, the resulting figure is 4883, the same as earlier.

4. A huge figure, $399^{(500)}$, can be embedded in “Jack and Jill went up the hill”.
Assume that we are using the map in Table 4.3 (mapkey “Lara is an enthusiastic woman”). If “and” and “went up” denote add and exponentiation, then Jack and Jill went up the hill = $399^{(201+299)} = 399^{(500)}$

The addition of meta-rules makes the cipher more complicated for the hacker, and hence harder to break. However, because deciphering is the machine’s job, the meta-rules do not require extra work from authentic receivers of the data.

4.9 Security aspects of mapkey based k-association maps
The material presented in this Section lays the foundation for the cryptanalysis of k-steganography. We first show that the permutation of rows and columns of a k-association map determined by a mapkey leads to a new map which is comparable with a randomly generated map. Next we show that a change in the mapkey is an effective technique that retains the required security against attacks which use statistical analysis of historical data. Our hypothesis is that k-steganography is secure provided the mapkey is not compromised and it remains secure if the mapkey is dynamically changed at sufficiently short intervals. Unlike the case with conventional cryptosystems, this security compliance does not unduly burden the user because of the human-friendly attributes of k-logic.

4.9.1 Distance between k-association maps
If say 75% of the consonants in one map are in the exact same position in a second map, then by looking at the first map we know a lot about the second; we claim that this feature is indicative of the closeness between the two maps. The question is posed: Given two key-based association maps, what is the distance between them? In other words, how similar are they? To prohibit unauthorized access to the map, we would like maps to be far apart from one another—i.e., knowing one should give very few clues about another.
The purpose of the experiment below is to develop a heuristic to find the distance between two maps and hence to generate maps at a desired distance from one another.

In this experiment, a fundamental choice is whether the map is key-based or random. In the context of this choice, we checked the influence of vowel usage frequency on synset generation. According to language theory, vowels occur more commonly than consonants, and according to our model, vowels are user-selected. We can have a standard vowel set, or non-standard user-selected vowel sets.

We fixed the ciphertext language to be English. We selected standard vowels and four types of non-standard vowels, namely, letters occurring with high frequency, average frequency and low frequency and letters selected randomly. In each case the size of the vowel set was fixed at 5, to facilitate comparison with the standard vowel set. Over 100 different maps were tested, out of which, for the purpose of analysis and reporting of results, a total of 20 samples are selected: five key-based and five random samples for the standard and non-standard vowel sets, respectively, for a total of (5+5)*2 = 20. (Call the four sets of 5 samples by the names SVK, NSVK, SVR and NSVR.) For identification, these maps are assigned a number from 1 to 20. The 21st is the basis map for the standard vowel set, built by putting the given consonant sequence into the map table in row-major order, without the subsequent addition of blanks or permutation of columns. (Call the basis map by the name B).

The heuristic used for quantifying the distance between maps is defined as follows:

A 26 x 21 matrix L is created, with rows indexed by letters of the English alphabet and columns indexed by data set number. The (i, j)-th element of L contains the column number of the i-th letter in the j-th map, where i = 1, 2, ... 26 and j = 1, 2, ..., 21. There is some dissimilarity between any two maps j1 and j2, because some letters will have shifted position between j1 and j2. We want to quantify this dissimilarity for each of the letters 'a', 'b', 'c', ..., or in other words, for the i-th letter in this lexicographic order, for i = 1, 2, ... 26. Define the difference for the i-th letter in the lexicographic order to be ( L(i, j1) - L(i, j2) ) mod 10, i.e., the distance between the columns in maps j1 and j2 where the i-th letter occurs, starting with the column in j1 and counting in left-to-right wraparound order.
Behind defining the distance measure in this manner, there is an implicit assumption regarding the behaviour of a systematic brute force attack, namely: the attacker probes the map in contiguous order, one column after another. If the i-th letter is present in map j1 but not in map j2, then all 10 columns of j2 need to be probed. Conversely, if it is absent in j1 then there is no need to probe j2 at all. The distance is computed accordingly.

The distance between maps j1 and j2 is the summation of differences over all letters between these two maps. That is,

\[
\text{distance } (j_1, j_2) = \sum_{i=1}^{26} (L(i, j_1) - L(i, j_2)) \mod 10.
\]

Statistics for the distance heuristic are summarized below: The figures in Table 4.6 indicate average pair wise distances between the five categories. (The matrix below is symmetric, so only the upper triangular part is given.)

<table>
<thead>
<tr>
<th>Map Category</th>
<th>B</th>
<th>SVK</th>
<th>SVR</th>
<th>NSVK</th>
<th>NSVR</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>0</td>
<td>91</td>
<td>84</td>
<td>114</td>
<td>105</td>
</tr>
<tr>
<td>SVK</td>
<td>95</td>
<td>98</td>
<td>111</td>
<td>102</td>
<td></td>
</tr>
<tr>
<td>SVR</td>
<td>89</td>
<td></td>
<td>111</td>
<td>108</td>
<td></td>
</tr>
<tr>
<td>NSVK</td>
<td>112</td>
<td></td>
<td></td>
<td>108</td>
<td></td>
</tr>
<tr>
<td>NSVR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>120</td>
</tr>
</tbody>
</table>

Table 4.6: Average pair-wise distances between k-association maps of different categories

The major conclusions drawn from the averaged data are:

1. The distance of maps of all categories from maps with non-standard vowel sets (columns in Table 4.6 labeled NSVK and NSVR) is larger than their distance from maps with the standard vowel set (columns labeled SVK and SVR).
2. On the distance measure, key-based maps (columns SVK and NSVK) perform about as well as random maps (columns SVR and NSVR, respectively).
3. The distance between two key-based maps is often greater than the distance between two random maps. Moreover, the distance between any given map and a key-based map is a matter of choice. It does not take much effort—our experience is that intuition suffices—for the participating “man” in the IMME to

4. Steganography research
choose a mapkey that will control the distance. That is, key-based maps provide as much security as random ones due to choices at map-formation time.

4.9.2 Similarity between k-association maps

The conclusion of the previous experiment was that maps built with mapkeys are close to randomly generated maps and that a map "sufficiently distant" from another can be generated by mere inspection. The aim of the next experiment is to determine the effect of replacing a map by another. Informally speaking, we attempt to check the probability that a new map retains some of the characteristics of an earlier map. The particular question is: how many keys would not change their numeric encoding when the map is changed? A large number of keys retaining the same encoding in the new map means that the impact of using a new map is not significant. A small number of keys having the same encoding across different maps means that a hacker cannot get decoding clues and that sufficient security can be obtained if the map is changed periodically.

A dictionary (the Pocket Oxford Dictionary, approx. 30,000 words) was enciphered using each of the 21 representative maps discussed above. The keys were built out of the first two consonants of the words, and therefore were mapped into 100 values. The outcome is compiled in the following graph, and the information contained in it is summarized in Table 4.7 below.

![Graph showing the impact of changing the map on the keys](image)

**Figure 4.1:** Impact of changing the map on the keys

From Table 4.7 below we can conclude that less than 1% of keys have the same old encoding in 61% of the cases (left half of the Table 4.7). In 38% of the cases 1 to 6% of the keys have retained the old encoding. Rarely, in 1% of the cases, 7 to 11% of the keys have got mapped to the same encoding even if a new map is employed.

4. Steganography research
Table 4.7: Impact of changing the map on the keys: summary of results

<table>
<thead>
<tr>
<th>% of maps</th>
<th>% of keys that retain the old mapping</th>
<th>% of maps</th>
<th>% of keys that retain the old mapping</th>
</tr>
</thead>
<tbody>
<tr>
<td>17.6</td>
<td>0.1</td>
<td>15.8</td>
<td>1</td>
</tr>
<tr>
<td>4.8</td>
<td>0.2</td>
<td>11.8</td>
<td>2</td>
</tr>
<tr>
<td>4.4</td>
<td>0.3</td>
<td>5.3</td>
<td>3</td>
</tr>
<tr>
<td>9.2</td>
<td>0.4</td>
<td>1.8</td>
<td>4</td>
</tr>
<tr>
<td>5.7</td>
<td>0.5</td>
<td>1.8</td>
<td>5</td>
</tr>
<tr>
<td>5.3</td>
<td>0.6</td>
<td>1.3</td>
<td>6</td>
</tr>
<tr>
<td>2.6</td>
<td>0.7</td>
<td>0.4</td>
<td>7</td>
</tr>
<tr>
<td>4.8</td>
<td>0.8</td>
<td>0.4</td>
<td>10</td>
</tr>
<tr>
<td>6.6</td>
<td>0.9</td>
<td>0.4</td>
<td>11</td>
</tr>
</tbody>
</table>

If the encoding space is increased from size 100 to size 1,000 by making keys with 3 consonants, then obviously there will be even less of a probability that the keys will get the same encoding when the map is changed. The new map will effectively carry fewer clues and hence security is retained. A consequence of expanding the key space is that the synset size is smaller and so involves more skill and effort in the construction of the steganogram. In the other direction a bigger synset size has the advantage of a larger choice of word selection, which facilitates easier construction of a larger number of innocuous steganograms for a single message and hence enhances security. In either case, the scheme ensures the required level of security.

4.10 Analysis of K-steganography

An attempt is made to explain here and in Section 4.11 the worth and effectiveness of the BK and GK steganographic schemes in comparison to the Baconian cipher.

BK has the advantage of using a one-time pad, an undoubtedly effective technique for ensuring security. The construction of innocuous text in a steganogram requires effort in proportion to the message length. Conventionally, bit-encrypted messages are embedded in the cover to produce the steganogram, and this provides added security to the data. The conversion of a text message to numeric form in BK—“book cipher” in cryptology—functions like bit-encryption of the message in conventional steganography. The kind of compression that takes place due to bit-encryption also happens with the numeric denotation in k-steganography. The advantage of the book
cipher in BK is that the message can be determined dynamically. (Unlike bit-encryption, here it is not necessary to hardcode the message space a priori. It is sufficient to decide upon the basic elements formulating the messages, namely, words, phrases, etc. The sender can enjoy the freedom of constructing the message just-in-time according to his creativity and natural linguistic instincts and skills. The variety of forms used in the construction contributes to security.)

BK takes advantage of a dictionary (or book) to encipher the message with word indices. This is at the cost of the length of the message, which grows logarithmically with dictionary size. Logarithmic growth appears to be insignificant but its effect upon the steganogram is not so. An additional digit requires one syllable, which (as discussed above) would typically lead to an additional word plus associated superfluous elements in the steganogram. In the other direction, reducing the size of the dictionary will reverse this cost. In practice it will be sufficient to use a colloquial dictionary of much smaller size, after the elimination of a large number of rarely used words which normally would not be used in the cover anyway. Typically a small number of words are used frequently. If a book is used as a one-time pad instead of a dictionary, then the more common a word is, the more indices it will have. This will increase the synset size and therefore facilitate the construction of the steganogram.

We propose to interpret k-steganography as a generalization of the Baconian cipher systems discussed in Section 4.6.4. Whereas the plaintext alphabet is encoded in the Baconian cipher as a binary string, here the encoding is a decimal numeric string in the basic katapayadi system. In the generalized katapayadi system the mapping is not necessarily to numbers, but to an arbitrary alphabet which is isomorphic to a numeric string with a user-defined radix. It is mentioned in passing that since the radix is under user-control, predictability decreases and hence code-breaking strength increases. A comparative statement of the codebreaking strengths of the Baconian cipher versus k-steganography is given in the cryptanalysis section (Table 4.8 of Section 4.11) below.

There are two types of steganograms—without and with superfluous text. Not counting superfluous text, for a one-character message the Baconian steganogram takes \( \log_2 (\text{no. of plaintext symbols}) \), while in k-steganography the length is \( \log_{\text{radix}} \) (no. of plaintext symbols) times a small constant. The latter is smaller, since the radix...
will be considerably larger than 2 (the radix is 10 in BK and could be equal to the
number of plaintext symbols in GK). The "small constant" is 1 when, as recommended,
the number of cipher text consonants is about 3 times the number of plaintext symbols,
and increases according as the number of consonants in a user-specified language is
smaller. Recall that in such situations, we need to visualize a different model to define
consonants; a "consonant" could be a combination of characters, e.g., vowel clusters
(vowel strings of length 1 to 4 and average length 2.6 as observed in our cluster
database) have been treated as consonants in our GK experiment.

In a typical encryption example, where both plaintext and ciphertext languages are
English, the Baconian cipher would encode a one-letter message using a bit string of
length 5, whereas if the consonant model discussed above in Section 4.5.2 is followed
then a GK-steganogram would map the letter to a vowel cluster of average length 2.6
characters. (Note: encoding and decoding with the GK scheme is possible without
superfluous structure, though not with the meta-rule used above.)

Without superfluous characters, every character is decodable. This case is more like a
cryptogram, and there is little scope to create an innocuous text to embed the message.
The introduction of superfluous characters in k-steganography provides the scope for
generating higher-order linguistic structures like words, phrases and sentences for
embedding. These structures are more likely to appear innocuous, and hence increase
stealth. The distribution of clues in the expanded text increases its codebreaking strength.

When superfluous characters are taken into account, steganogram length increases. In
the Baconian cipher, one bit of the message is encoded in one word. One character of
the message, denoted as 5 bits, needs as many as 5 words. Hence one word-message,
say of length 4 characters will require 20 of words in the steganogram. Recall that in
our experiment BK-steganograms took 3 to 7 words for a one-word message and a
GK-steganogram took 18 words. The steganogram length for the Baconian cipher is
observed to be a little larger than the figure for GK, and about an order of magnitude
more than that for BK. However, the Baconian cipher cannot take full advantage of
increased text size because the protocol defines a single control character in each
word (which alone is to be decoded). There is no such limitation in k-steganography.
The result is an increase in the dispersal of clues, leading to a comparatively more

4. Steganography research
robust steganogram. Finally, the mnemonic map of the katapayadi method is a user-friendly feature, whereas the Baconian cipher does not have this advantage.

4.11 Cryptanalysis of K-steganographic systems
According to Kirchoff's law, in cryptology nothing except the key can be assumed to be secure from intruders. In particular the encryption algorithm should be published before it is put into practice. Similarly in steganography, the algorithm to generate a steganogram should be disclosed and only the key is to remain secret. With k-steganography, the k-logic to construct steganograms would be the publicly available algorithm and the mapkey-based k-association map would be the secret key.

A bruteforce attack would consist of finding and trying out all possible k-association maps for a given plaintext and ciphertext language. Therefore, to determine the code-breaking strength of k-steganography we need to know the number of k-association maps. This figure is calculated below:

1. Recall that in a k-association map without additional inserted blanks, n is the number of plaintext symbols, m is the number of ciphertext consonants and k is the number of rows in the map table. Thus $k = \lceil m/n \rceil$, i.e., $m/n$ rounded up to the nearest integer. To obtain the advantage of homophones we expect $m$ to be more than $n$ by a factor of 2 to 3 or more. Assume that the number of ciphertext consonants is exactly $k \times n$. The number of possible k-association maps is calculated below:

2. We generate an association map one column at a time, placing $k$ ciphertext consonants in each column. If the consonants are not to be repeated, then the set of unused consonants is depleted, $k$ consonants per column. The number of different ways of composing the $i$-th column is $m \cdot (i-1)^k \binom{m}{k}$.

3. The map is generated by composing the $i$-th column, for $i = 1, 2, \ldots, n$. So the number of different ways of composing the map is $\prod_{i=1}^{n} m \cdot (i-1)^k \binom{m}{k}$.

4. Generalized katapayadi (k-logic) allows user selection of vowels. Assuming that vowels are 25% of the ciphertext alphabet, and the number of consonants $m$ is $nk$, then: the alphabet size is $(4/3)nk$, the number of vowels is $(1/3)nk$, and the number of ways to select a vowel set out of the alphabet is $(4/3)nk \binom{nk}{(1/3)nk}$. 

4. Steganography research
5. For a given sequence of plaintext symbols, the total number of possible ways to construct an association map is due to the multiplication of the two expressions above:

$$\prod_{i=1}^{n} \binom{n-i}{k} C_k \cdot \binom{n}{k} C_k \cdot \binom{(4/3)^n}{k} C_k \cdot \binom{(1/3)^n}{k} C_k$$

6. To obtain the total number of mappings between the plaintext symbols and the ciphertext consonants, multiply the above product by n! (for the number of permutations on the set of plaintext symbols).

Simplifying this product we get that the number of distinct association maps is

$$(m! / k^n) \cdot n! \cdot \binom{(4/3)^m}{k} C_k \cdot \binom{(1/3)^m}{k} C_k$$

Table 4.8 below gives the value of this figure when ciphertext languages are (i) Sanskrit-like and (ii) English-like, i.e., alphabet sizes 42 and 26 and vowel set sizes 10 and 6, respectively. The results below are produced for two cases: (i) plaintext with 10 symbols and (ii) plaintext with 8 symbols. Next we give the value of this figure for a third case: the plaintext language is English-like, i.e., with 26 plaintext symbols, and the ciphertext language is also English-like in which 101 vowel clusters are treated as “consonants” in the k-association map. Finally, the combinations involved in the basic katapayadi scheme and the Baconian cipher are computed, for the comparison of their respective code breaking strengths. In the Table 4.8, “# computations” means number of maps to be generated. The code-breaking time, i.e., the time to generate all possible maps, is measured in years, assuming a system that computes $10^8$ maps per second.

<table>
<thead>
<tr>
<th>Basic katapayadi scheme</th>
<th>K-logic scheme</th>
<th>Baconian cipher</th>
</tr>
</thead>
<tbody>
<tr>
<td># computations</td>
<td>Breaking time</td>
<td># computations</td>
</tr>
<tr>
<td># plaintext alphabet = 8; # ciphertext alphabet = 42; # ciphertext consonants = 32</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$32C_{32}$</td>
<td>$3.2 \times 10^{16}$</td>
<td>$(32! / (4!^8)^{32!} / 42! C_{10})$</td>
</tr>
<tr>
<td># plaintext alphabet = 8; # ciphertext alphabet = 26; # ciphertext consonants = 20</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$26C_{20}$</td>
<td>$3.4 \times 10^{12}$</td>
<td>$(20! / (3!^2)^{20!} / 26! C_6)$</td>
</tr>
<tr>
<td># plaintext alphabet = 10; # ciphertext alphabet = 26; # ciphertext consonants = 32</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$40C_{32}$</td>
<td>$2.4 \times 10^{14}$</td>
<td>$(40! / (4!^5)^{40!} / 42! C_{10})$</td>
</tr>
<tr>
<td># plaintext alphabet = 10; # ciphertext alphabet = 26; # ciphertext consonants = 20</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$20C_{20}$</td>
<td>$3.2 \times 10^{16}$</td>
<td>$(20! / 26! C_{6}) / 26!$</td>
</tr>
<tr>
<td># plaintext alphabet = 26; # ciphertext alphabet = 26; # (visualized) ciphertext consonants = 104</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 4.8: Comparison of code breaking strength: BK and GK schemes vs Baconian cipher

4. Steganography research
Remarks:
1. The basic katapayadi scheme follows the lexicographical sequence of the alphabet (the different maps are due to different choices of spaces in the table i.e. association map). The k-logic scheme makes different maps based upon different arrangements of the alphabet in the table, which depend on the user-specified mnemonic mapkey.

2. Computations in the case of the Baconian cipher are firstly due to the choices available for the selection of binary strings to represent the plaintext alphabet, and secondly due to the number of ways the given ciphertext alphabet is partitioned into two subsets. To obtain a "reasonable" lower bound on this figure with which we can compute codebreaking times, we assume that the ciphertext alphabet is equi-partitioned. (Note that $^n\text{C}_i$ is maximized when $i = \lceil n/2 \rceil$. If we had not assumed equi-partitioning the lower bound would be less tight.)

The results show that the katapayadi scheme does not withstand brute force attack by contemporary computers— the katapayadi cipher can be broken within a small fraction of second for a plaintext of alphabet size 8 or 10—but its generalization, i.e., k-logic, demonstrates remarkably great strength (billions of years in the simplest cases, and unbelievably larger amounts for the other cases). Further, the k-logic scheme outperforms the Baconian cipher.

The computing system could be a high performance cluster or a grid with a large number of participating machines, which could compute many more than $10^8$ maps per second. In this situation the codebreaking times would be scaled down appropriately. Even then, the breaking time would remain unreachably large for k-steganograms.

The cryptanalysis of k-steganography above is to indicate the resource costs for codebreaking. In practice, human experts would apply heuristics—just as they do for conventional steganography—based on their domain knowledge, in order to dramatically reduce map computations.

4.12 Comparison of k-steganography with existing systems
1. K-steganography does not use pre-existing text as a cover, but generates new text in the process of creating a steganogram. The advantage of this approach is an increased data rate. We have demonstrated above in Section 4.5 that a human can compose a
natural language k-steganogram of length 10 to 20 words to hide a one-word message. This is comparable to the data rate of Hydan and the hiding capacity provided by JPEG picture files, discussed in the previous Chapter. In comparison, well-known natural language steganographic systems, (e.g., Nicetext, Spammimic and systems by Weinstein and Atallah discussed in the previous Chapter) use a cover of length 100 to 200 words to hide a one-word message [Bergmair05]. Thus we have achieved a higher data rate by a factor of 10 as compared to earlier natural language steganographic systems.

2. The use of synsets while creating a steganogram is a fundamental feature of lexical steganography. However, in k-steganography the selection of different contexts while juxtaposing message-carrying words is like the approach of contextual steganography.

3. K-steganography would work like Spammimic if human intervention in the construction of the steganogram is replaced by finite automata. Also, it would work like Nicetext if instead of generating a steganogram from scratch we let the system use a template for the syntactic structure of the cover language and choose synonyms using an annotated dictionary.

4. The distinguishing feature of k-steganography is that it inherits katapayadi LSC features of user-friendliness, programmability, adaptability and safety (discussed in Section 2.5).

4.13 Towards a completely automated stegosystem

It is feasible to provide a tutor interface for the schemes developed in this Chapter. The tutor software would support demonstration and practice sessions, so that the average user of the language could develop his steganographic skills.

This model of k-steganography may be extended to a completely automated system. The training data for the system can be the man-machine interactions that take place while human experts juxtapose suitable word sequences to yield information-bearing sentences. This data should generate clues for semantic associations of the words, which in turn should lead to valid expressions. The next step is that the system should generate conceptual dependencies of these contexts and maintain a database of this information. This research could lead to the creation of a system that would identify a suitable context where the synset sequences denote the message while the human
being merely provides the meta-rules. Generating such a system which learns from the context, generates conceptual dependencies and develops new dependencies on its own is a research topic that lies beyond the scope of this work. This issue is discussed again in Chapter 9.

4.14 Conclusions

A novel scheme for linguistic steganography has been proposed, in which generalized katapayadi has been employed to create synonyms which will be used by the "man" part of IMME to embed the message in a text cover.

We argue that k-steganography is more flexible than earlier methods because it offers greater freedom for the dynamic construction of a steganogram and wide scope for the messages to be embedded. K-steganography provides a meta-level formula or grammar to generate innocuous steganograms for a given subliminal message. It does not have the constraint of a one-to-one mapping between the cover and the subliminal domains. We have shown that the homophonic nature of the k-association map provides the user with enhanced capabilities for the construction of natural language steganograms. This facility is complemented by user-defined heuristics to define or control the synonym generation process. Further, the built-in subjectivity due to human participation—both in the construction of synonyms and in juxtaposing the synonyms to form an innocuous natural language embedding for the message—contributes towards increasing the code-breaking strength of the steganogram.

The code-breaking strength of k-steganography is in large measure due to the k-association map. It has been shown that in the absence of a mapkey, k-association maps are almost as hard to guess as randomly generated ones.

K-steganography is adaptive in the sense that it has the same advantage as that of book ciphers, a time-tested secure scheme. Inherent in the robustness of k-steganography is the presence of system-assisted human judgment (i.e., intelligence facilitated by means of IMME). Hence it supports error detection and correction of the kind that comes naturally to humans. The scope of the error detection and correction is further enhanced with conventional techniques like sending data redundantly.

4. Steganography research
Compared to contemporary solutions, the data rate of k-steganography is higher by a factor of as much as 10. Combinatorial analysis shows that it can take billions of years for a na"ive brute-force attack (i.e., the calculation of all k-association maps in some pre-determined order) to break a k-steganogram. The magnitude of this time interval signifies that substantial resources—advanced technology systems, more sophisticated attacks or human expertise—need to be invested to decipher it. We have shown that k-steganography can be interpreted as a generalization of the Baconian cipher, and further that that k-steganography outperforms the Baconian cipher.

Finally it is hypothesized that k-steganography, suitably combined with contemporary systems like Nicetext or Spamminic, may evolve into a higher performance automatic stegosystem compared to those reported in the literature. Further, it is suggested that an intelligent stegosystem could emerge out of the extrapolation of the training potential of k-steganographic systems, by replacing the human user in IMME with a computing system along with a suitable machine-learning algorithm.

Given its distinguishing advantages, it is probably worth implementing the k-steganography model as a possible commercial product, and its adaptive characteristics call for further exploration of its research potential.