Strengthening INORMALS Using Context-based Natural Language Generation

Soni Yora & Ari Moesriami Barmawi*

Graduate School of Informatics, School of Computing, Telkom University, Kawasan Pendidikan Telkom, Sukapura, Kec. Dayeuhkolot, Bandung 40257, Indonesia
*E-mail: mbarmawi@melsa.net.id

Abstract. The noiseless steganography method that has been proposed by Wibowo can embed up to six characters in the provided cover text, but more than 59% of Indonesian words have a length of more than six characters, so there is room to improve Wibowo’s method. This paper proposes an improvement of Wibowo’s method by modifying the shifting codes and using context-based language generation. Based on 300 test messages, 99% of messages with more than six characters could be embedded by the proposed method, while using Wibowo’s method this was only 34%. Wibowo’s method can embed more than six characters only if the number of shifting codes is less than three, while the proposed method can embed more than six characters even if there are more than three shifting codes. Furthermore, the security for representing the number of code digits is increased by introducing a private key with the probability of guessing less than 1, while in Wibowo’s method this is 1. The naturalness of the cover sentences generated by the proposed method was maintained, which was about 99% when using the proposed method, while it was 98.61% when using Wibowo’s method.

Keywords: Baudot-Murray code; INORMALS; linguistic-based steganography; natural language generation; sentence paraphrasing.

1 Introduction

Along with the increasing exchange of digital data in daily communication, mainly in the form of texts, it is necessary to guarantee message confidentiality and avoid message forgery. One method to guarantee message security (specifically for message authenticity) is text steganography.

Desoky has proposed Mature Linguistic Steganography (Matlist) to hide secret messages in documents that have certain domains based on random series data [2]. Later, Desoky proposed NORMALS (Normal Linguistic Steganography) to overcome vulnerabilities, linguistic flaws, and limitation issues of Matlist. NORMALS employs natural language generation (NLG) to generate noiseless (flawless) and legitimate cover texts by manipulating the non-random series input parameters of the NLG system to camouflage the data in the generated text [1].
Wibowo and Barmawi have proposed INORMALS to improve NORMALS using a modified Baudot-Murray code to encode secret messages and represent the generated code digits as answer sentences to questions in a questionnaire (e-money).

The output of INORMALS is a cover text consisting of sentences and timestamps. INORMALS can embed up to six characters in the cover text [3]. However, more than 59% of Indonesian words are longer than six characters, and thus it is difficult to conceal secret messages in Indonesian, even for a single word only. Hence, possibilities for an updated method that can accommodate longer secret messages should be investigated. Furthermore, the number of code digits used in the INORMALS method is written in plain text as part of the timestamp [3]. The number of code digits should be concealed to reduce the risk of the secret message being broken by attackers and improve the system’s robustness.

This paper proposes solutions to overcome these shortcomings of INORMALS. The embedding capacity is increased by reducing the code digit representation to sentences. In the proposed method, the security level for representing the number of code digits is improved using a private key, while maintaining the naturalness of the sentences using sentence paraphrasing based on contextual synonym substitution [5] and sentence pattern transformation [6,7]. This method can be used on any language, but in this research the Indonesian language was used, because Wibowo’s study used the Indonesian language. Thus, the performance of his method and our proposed method could be compared. In the case of other languages, the language’s corpus along with its semantic and grammar rules must be used.

This research also used external input in the form of a questionnaire consisting of eighteen questions with five answers on a Likert scale as used by Wibowo. Since the Indonesian language was used in this research, the cover text was in Indonesian. Based on the experiment result, the proposed method generates more natural sentences and increases the embedding capacity.

2 INORMALS

INORMALS is a modification of the Normal Linguistic Steganography method that employs natural language generation (NLG) [9] techniques to generate noiseless (flawless) and legitimate cover texts by using a questionnaire to camouflage the data in the generated text. INORMALS uses an e-money questionnaire with answers in five-point Likert-scale form [3]. The maximum number of questions is eighteen. The INORMALS architecture is shown in Figure 1.
The INORMALS Encoder converts the message’s text into a modified Baudot-Murray code [3]. The modified Baudot-Murray code is an encoding system that applies a quinary number system [12] consisting of 125 codes, with the left and right columns mapping the bigrams to characters. For selecting the left or the right side, shifting codes are used, represented by the number 444. The bigram mapping to Baudot-Murray code uses filtering of a corpus containing 1,000 articles. An example of a Baudot-Murray code table is shown in Table 1.

The following is an example of generating a cover text with INORMALS. The secret message is ‘PECAT’, with the length of the message is 4. The INORMALS Encoder converts the secret message into Baudot-Murray code based on the bigram column, which in this example resulted in the number of code digits is 3 (in code ‘024 414 444 431’). INORMALS represents one code digit by one sentence derived from a questionnaire. When there are remaining questions, it is always generated in full to avoid casting suspicion on the cover text. Another type of sentence in the INORMALS cover text represents the number of code digits. This number is represented by a timestamp.
INORMALS uses static template sentences to represent one code digit, which are selected according to the number of code digits. One digit has three sentence options that can be selected randomly [3]. Table 2 shows examples of the template sentences.

Table 1  Bigram-Baudot-Murray code map.

<table>
<thead>
<tr>
<th>No.</th>
<th>Code</th>
<th>Left</th>
<th>Right</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000</td>
<td>AN</td>
<td>IM</td>
<td>132</td>
</tr>
<tr>
<td>001</td>
<td>N</td>
<td>EP</td>
<td>133</td>
</tr>
<tr>
<td>002</td>
<td>NG</td>
<td>O</td>
<td>134</td>
</tr>
<tr>
<td>003</td>
<td>D</td>
<td>AU</td>
<td>140</td>
</tr>
<tr>
<td>004</td>
<td>EN</td>
<td>E</td>
<td>141</td>
</tr>
<tr>
<td>010</td>
<td>A</td>
<td>ID</td>
<td>142</td>
</tr>
<tr>
<td>011</td>
<td>DA</td>
<td>R</td>
<td>143</td>
</tr>
<tr>
<td>012</td>
<td>I</td>
<td>LU</td>
<td>144</td>
</tr>
<tr>
<td>013</td>
<td>ER</td>
<td>KI</td>
<td>200</td>
</tr>
<tr>
<td>014</td>
<td>KA</td>
<td>MO</td>
<td>201</td>
</tr>
<tr>
<td>020</td>
<td>P</td>
<td>AB</td>
<td>202</td>
</tr>
<tr>
<td>021</td>
<td>M</td>
<td>IH</td>
<td>203</td>
</tr>
<tr>
<td>022</td>
<td>AR</td>
<td>P</td>
<td>204</td>
</tr>
<tr>
<td>023</td>
<td>YA</td>
<td>KS</td>
<td>210</td>
</tr>
<tr>
<td>024</td>
<td>PE</td>
<td>RK</td>
<td>211</td>
</tr>
<tr>
<td>030</td>
<td>LA</td>
<td>TO</td>
<td>212</td>
</tr>
<tr>
<td>031</td>
<td>ME</td>
<td>PU</td>
<td>213</td>
</tr>
<tr>
<td>032</td>
<td>AT</td>
<td>EH</td>
<td>214</td>
</tr>
<tr>
<td>033</td>
<td>AK</td>
<td>NO</td>
<td>220</td>
</tr>
<tr>
<td>034</td>
<td>G</td>
<td>RS</td>
<td>221</td>
</tr>
<tr>
<td>040</td>
<td>S</td>
<td>DU</td>
<td>222</td>
</tr>
<tr>
<td>041</td>
<td>T</td>
<td>GK</td>
<td>223</td>
</tr>
<tr>
<td>042</td>
<td>RA</td>
<td>NK</td>
<td>224</td>
</tr>
<tr>
<td>043</td>
<td>TA</td>
<td>IR</td>
<td>230</td>
</tr>
<tr>
<td>044</td>
<td>SI</td>
<td>UL</td>
<td>231</td>
</tr>
<tr>
<td>100</td>
<td>DI</td>
<td>PR</td>
<td>232</td>
</tr>
<tr>
<td>101</td>
<td>IN</td>
<td>F</td>
<td>233</td>
</tr>
<tr>
<td>102</td>
<td>GA</td>
<td>RB</td>
<td>234</td>
</tr>
<tr>
<td>103</td>
<td>AL</td>
<td>NJ</td>
<td>240</td>
</tr>
<tr>
<td>104</td>
<td>BA</td>
<td>OM</td>
<td>241</td>
</tr>
<tr>
<td>110</td>
<td>K</td>
<td>UP</td>
<td>242</td>
</tr>
<tr>
<td>111</td>
<td>UN</td>
<td>Y</td>
<td>243</td>
</tr>
<tr>
<td>112</td>
<td>B</td>
<td>HI</td>
<td>244</td>
</tr>
<tr>
<td>113</td>
<td>SE</td>
<td>EG</td>
<td>250</td>
</tr>
<tr>
<td>114</td>
<td>AS</td>
<td>HU</td>
<td>301</td>
</tr>
<tr>
<td>120</td>
<td>NA</td>
<td>UH</td>
<td>302</td>
</tr>
<tr>
<td>121</td>
<td>K</td>
<td>GE</td>
<td>303</td>
</tr>
<tr>
<td>122</td>
<td>TU</td>
<td>DO</td>
<td>304</td>
</tr>
<tr>
<td>123</td>
<td>T</td>
<td>UD</td>
<td>310</td>
</tr>
<tr>
<td>124</td>
<td>EL</td>
<td>PO</td>
<td>311</td>
</tr>
<tr>
<td>130</td>
<td>EM</td>
<td>IP</td>
<td>312</td>
</tr>
<tr>
<td>131</td>
<td>PA</td>
<td>EC</td>
<td>313</td>
</tr>
</tbody>
</table>

Note: '_' is space.
The number of code digits is represented by a timestamp in seconds and milliseconds. This process aims to let the receiver know what the number of code digits is. The seconds and milliseconds represent the number of code digits using Eq. (1):

\[
\text{second} = \text{hour} + \text{minute} - \text{numberOfCodeDigits} \\
\text{millisecond} = \text{random}(0 - 499) \text{ if } \text{second} \leq 0 \\
\text{millisecond} = \text{random}(500 - 999) \text{ if } \text{second} > 0
\]  

(1)

For example, the number of code digits is 12, and the current time is 17:17:27. The second value of the result from Eq. (1) is 32 and the millisecond is 524. The complete sentence is ‘Laporan ini dihasilkan secara otomatis oleh sistem pada tanggal 2020-08-17 jam 17:17:32.524’. An example of a cover text from INORMALS is shown in Figure 2.

---

**Table 2** Sentence templates.

<table>
<thead>
<tr>
<th>Question</th>
<th>Answer</th>
<th>Sentence Pattern</th>
<th>Sentences</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0</td>
<td></td>
<td>Anda / sama sekali tidak merasa terbantu / dalam mengelola aktivitas keuangan Anda / sejak menggunakan e-money</td>
<td>Anda / sama sekali tidak merasa terbantu / dalam mengelola aktivitas keuangan Anda / sejak menggunakan e-money</td>
</tr>
<tr>
<td>0 0 1</td>
<td></td>
<td>Sejak menggunakan e-money / Anda / sama sekali tidak merasa terbantu / dalam mengelola aktivitas keuangan Anda /</td>
<td>Sejak menggunakan e-money / Anda / sama sekali tidak merasa terbantu / dalam mengelola aktivitas keuangan Anda /</td>
</tr>
<tr>
<td>0 0 2</td>
<td></td>
<td>Sejak menggunakan e-money / Anda / sama sekali tidak merasa terbantu / dalam mengelola aktivitas keuangan Anda / sehari-hari /</td>
<td>Sejak menggunakan e-money / Anda / sama sekali tidak merasa terbantu / dalam mengelola aktivitas keuangan Anda / sehari-hari /</td>
</tr>
</tbody>
</table>

---

**Figure 2** Example of cover text in INORMALS [3].
3 Proposed Method

In the proposed method, the number of code digits is concealed by a private key that is generated using a pseudo-random generator [4], while the sentences are generated by paraphrasing sentences using contextual synonym substitution [5] and sentence pattern transformation [6,7]. An overview of the proposed method is shown in Figure 3.

To provide the data to be processed by the proposed method, preprocessing was done, which consisted of bigram to Baudot-Murray code mapping, creating a Likert scale for the answer choices; creating sentences as responses to the questions in the questionnaire based on the answers by the user; creating sentences to represent the shifting code; and creating synonym sets and grammar rules.
3.1 Pre-processing

Modified Baudot-Murray code is an encoding system that applies a quinary number system [12] consisting of 125 codes, with left and right columns that map bigrams to characters [3]. To select the left or the right column, shifting codes are used, represented by the number ‘444’. In the proposed method, constructing the bigram to Baudot-Murray code mapping was conducted using the INORMALS method, which analyzes n-grams from a corpus containing 1,000 articles [11].

The pseudo-code for constructing the Baudot-Murray code table is shown in Algorithm (1).

Algorithm 1

FUNCTION Construct-Ngram-Baudot-Murray-Table(corpus)
    Input: Indonesian corpus
    Output: modified n-gram Baudot-Murray mapping table
    //get 124 * 2 the-most-Ngram-collection
    the-most-Ngram-collection = Get-The-Most-NGram(corpus);
    //sort by descending (the most to the least)
    Sort-the-most-Ngram-collection-by-descending(the-most-Ngram-collection);
    //put Shifting Code (444) at the end of the-most-Ngram-collection
    the-most-Ngram-collection.ADD('444');
    RETURN the-most-Ngram-collection;
END FUNCTION

FUNCTION Get-The-Most-NGram(corpus)
    Input: Indonesian corpus
    Output: n-gram in accordance with its frequency percentage value
    //compare frequency total-percentage value among n-gram chars type
    //start from 2-gram until 5-gram
    FOR n-gram-type = 2 to 5
        chars-with-percentage = Get-Ngram-Char-With-Frequency-Percentage(n-gram type);
        IF TOTAL(chars-with-percentage.percentage) > the-most-ngram THEN
            the-most-ngram = chars-with-percentage;
        END IF
    END FOR
    RETURN the-most-ngram;
END FUNCTION

FUNCTION Get-Ngram-Char-With-Frequency-Percentage(corpus, n-gram-type)
    Input: Indonesian corpus
    Output: n-gram chars with its frequency percentage value
    //collect n-gram-chars for every n-gram-type
    FOR every char of n-gram-type in the corpus
        n-gram-collection.ADD(chars of n-gram-type);
    END FOR
    //Counting the number of n-gram-type based on group of chars then
    //dividing by the number of all n-gram types in the corpus
    FOR every char in n-gram-collection
        //calculate the percentage of n-gram chars
        percentage = GROUPCOUNT(n-gram-collection) / LEN(n-gram-collection);
        n-gram-char-group.ADD(chars, percentage);
    END FOR
    RETURN n-gram-char-group;
END FUNCTION
Answer choices are words or phrases represented by numbers, such that the answer chosen by a user can be interpreted as a code. Furthermore, one answer can be represented by more than one word or phrase, so that more sentence variations can be generated to represent one answer. For example, if the answer chosen by a user is ‘strongly disagree’ (sangat tidak setuju), then it can be represented by ‘amat tidak setuju’, ‘sama sekali tidak setuju’, or ‘benar-benar tidak setuju’. These answer choices are keywords that represent a code. The pseudo-code for constructing the five-point Likert scale of answer choices is shown in Algorithm (2).

Algorithm 2

FUNCTION Construct-Likert-Scale-Of-Answer-Choice(questionnaire)
Input: e-money questionnaire
Output: Likert scale of answer choice table
//manual synonym analysis of Likert scale of answer choices
FOR every question in the questionnaire
   FOR options-nth of Likert scale in a question
      answer-choices[options-nth].ADD(Get-Synonym-Of-Likert-Scale-ByLManually);
   END FOR
   Likert-scale-of-answer-choice-collection.ADD(answer-choices);
END FOR
RETURN Likert-scale-of-answer-choice-collection;
END FUNCTION

To generate sentences from the answer choices, where the context of the answers has to be maintained, it is necessary to analyze the placement of the answers in a sentence. The determination of the placement of an answer in a sentence is done manually by considering the context of the sentence. The placement of the answer is marked as ‘~’. For example, if the sentence is ‘Layanan e-money ~ membantu saya dalam mengelola aktivitas keuangan saya sehari-hari’, then ‘~’ is placed between the words ‘e-money’ and ‘membantu’ to represent the context of the answer sentence. The pseudo-code for creating space for the placement of a response to a question in a sentence is shown in Algorithm (3).

Algorithm 3

FUNCTION Create-Space-For-Response-Of-Questionnaire(questionnaire)
Input: e-money questionnaire
Output: list of sentences with space for the response to the questionnaire
//manual space of response analyzing for each question in questionnaire
FOR every question in questionnaire
   sentence-with-space = Manual-Space-Of-Response-Analyzing(question);
   list-of-sentence-with-space.ADD(sentence-with-space);
END FOR
RETURN list-of-sentence-with-space;
END FUNCTION
Based on the bigram to Baudot-Murray mapping table, shifting codes are represented by the code ‘444’. In the embedding process of the proposed method, a shifting code can be represented by one sentence, while in Wibowo’s method it is represented by three sentences. Because the shifting code is going to be used after a code that consists of three digits, the sentence representing the shifting code lies in the fourth or (multiple of four)-th sentence. Therefore, the maximum number of shifting codes used in the cover sentence is six, since the maximum number of codes that can be embedded in one cover sentence is six digits. Each sentence that represents a shifting code has a keyword. Keywords are used to identify whether a sentence in the cover text represents a shifting code or not. Sentences that represent shifting codes are manually created while maintaining the context of the sentences related to the sentences before and/or after it. The pseudo-code for creating sentences for representing the shifting codes is shown in Algorithm (4).

**Algorithm 4**

```
FUNCTION Create-Sentence-For-Representing-Shifting-Code(questionnaire)
    Input: e-money questionnaire
    Output: list of sentences to represent the shifting code of each 3rd sentence
    //manual list of sentences to represent shifting code analysis
    //for each 3rd sentence
    FOR every question in the questionnaire
        IF INDEX(question) modulo 3 == 0 OR INDEX(question) == 1 THEN
            sentence-shifting-code = Create-Manually-By-Context-Relation(question);
            list-of-sentence-shifting-code.ADD(sentence-shifting-code);
        END IF
    END FOR
    RETURN list-of-sentence-shifting-code;
END FUNCTION
```

For sentence paraphrasing, creating synonym sets was conducted using contextual synonym substitution tools [5], while sentence pattern transformation used three pattern transformations based on Indonesian grammar [6,7]. The sentence pattern transformation rules is shown in Table 3.

<table>
<thead>
<tr>
<th>Rule</th>
<th>Sentence Pattern</th>
<th>Transformation</th>
<th>Type of Sentence</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Subject – Predicate</td>
<td>Predicate – Subject</td>
<td>Inverted or vice versa</td>
</tr>
<tr>
<td>2</td>
<td>Subject – Predicate – Object</td>
<td>Subject – Predicate - Adjunct</td>
<td>Active to passive or vice versa</td>
</tr>
<tr>
<td></td>
<td>• Subject – Predicate – Object – Adjunct</td>
<td>• Adjunct – Subject – Predicate – Object</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Subject – Predicate – Adjunct</td>
<td>• Adjunct – Subject – Predicate</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Subject – Predicate – Adjunct</td>
<td>Adjunct position changes</td>
<td></td>
</tr>
</tbody>
</table>
Transforming a sentence while maintaining generality and formality has to refer to sentence structures that are often used in formal sentences. This process is done by analyzing the corpus. Fourteen sentences often appear in the Indonesian language, as shown in Table 4, and the pseudo-code for creating sentence structures often used in the corpus is shown in Algorithm 5.

Table 4  Structure of sentences often used in the corpus.

<table>
<thead>
<tr>
<th>No.</th>
<th>Syntactical</th>
<th>No.</th>
<th>Syntactical</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A-C-S-P</td>
<td>8</td>
<td>S-P</td>
</tr>
<tr>
<td>2</td>
<td>A-S-P</td>
<td>9</td>
<td>S-P-A</td>
</tr>
<tr>
<td>3</td>
<td>A-S-P-O</td>
<td>10</td>
<td>S-P-O</td>
</tr>
<tr>
<td>4</td>
<td>S-A-P</td>
<td>11</td>
<td>S-P-O-A</td>
</tr>
<tr>
<td>5</td>
<td>S-A-C-P</td>
<td>12</td>
<td>S-P-O-C</td>
</tr>
<tr>
<td>6</td>
<td>S-CONJ-P-O-P-A</td>
<td>13</td>
<td>S-P-O-C-A</td>
</tr>
<tr>
<td>7</td>
<td>S-CONJ-P-O-P-O</td>
<td>14</td>
<td>S-P-C</td>
</tr>
</tbody>
</table>

Algorithm 5

FUNCTION Create-Structure-Of-Sentence-Often-Used(corpus)

Input: corpus
Output: structure of sentence often used table
//do parsing for every sentence in corpus using PATR Tool [10]
FOR every sentence in corpus
    pattern-of-sentence = PATR-Parsing(sentence);
    list-of-patterns.ADD(pattern-of-sentence);
END FOR
//Count each group of the same pattern in list-of-patterns
list-of-patterns-used = Count-Grup(list-of-patterns);
//we define threshold of percentage greater than 1% of pattern-sentences
//that will be used as common and formal sentence in Indonesian
FOR every pattern-of-sentence in list-of-patterns-used
    IF pattern-of-sentence.percentage >= 1% THEN
        list-of-frequently-used-sentence-patterns.ADD(pattern-of-sentence);
    END IF
END FOR
RETURN Sort-By-Descending(list-of-frequently-used-sentence-patterns);
END FUNCTION

3.2  Message Concealment

The first process to conceal the secret message is encoding the message into codes based on the Baudot-Murray code table. For example, the secret message ‘PECAT’ was encoded as 024 414 444 431, where the number of code digits is twelve. The proposed method represents one non-shifting code by one sentence derived from the questionnaire, while one shifting code is represented by one sentence that is not derived from the questionnaire. The maximum number of sentences representing non-shifting code digits is equal to the maximum number of questions in the questionnaire. In this case, the maximum number of questions was eighteen. As for the shifting codes, three shifting code digits are represented
by one sentence. The maximum number of shifting codes used in a cover text is six because the maximum number of codes that can be embedded in one cover text is six digits. Therefore, six sentences representing shifting codes can accommodate eighteen shifting code digits. Thus, the maximum number of code digits that can be accommodated by the proposed method is 36.

Furthermore, in the embedding process, in the code that represents ‘PECAT’, the first digit is 0, represented by the answer choice ‘benar-benar tidak’, and the sentence that is generated as the first index is ‘Layanan e-money ~ membantu saya dalam mengelola aktivitas keuangan saya sehari-hari’. Thus, the sentence becomes ‘Layanan e-money benar-benar tidak membantu saya dalam mengelola aktivitas keuangan saya sehari-hari’. Furthermore, ‘saya’ is substituted by ‘anda’ because, from the reader’s point of view, the first-person pronoun must be substituted by a second-person pronoun. So the sentence becomes ‘Layanan e-money benar-benar tidak membantu anda dalam mengelola aktivitas keuangan anda sehari-hari’. Furthermore, ‘saya’ is substituted by ‘anda’ because, from the reader’s point of view, the first-person pronoun must be substituted by a second-person pronoun. So the sentence becomes ‘Layanan e-money benar-benar tidak membantu anda dalam mengelola aktivitas keuangan anda sehari-hari’. The substitute word for ‘Fasilitas’ is ‘Layanan’, for ‘menunjang’ it is ‘membantu’, and for ‘menjalankan’ it is ‘mengelola’.

After conducting synonym substitution, the pattern sentence is transformed into three possible sentence pattern transformations using the rules in Table 4. Finally, the sentence is parsed using constraint-based formalism tools [10] to identify the transformation possibilities. For example, the sentence is transformed into a passive sentence, so it becomes ‘Anda benar-benar tidak ditunjang oleh layanan e-money dalam menjalankan aktivitas keuangan anda sehari-hari’.

The last process to generate a sentence is checking the structure of the sentence and whether the sentence structure is often used in Indonesian formal sentences. If this is the case, then the sentence is included in the cover text/stego. Otherwise, the sentence transformation is canceled. Finally, sentence generation based on the questions and their answers is applied to all remaining questions that do not represent code digits of the secret message. Thus, the sentences of the cover text are complete, thus avoiding attacker suspicion.

The final sentence generation is used to generate a sentence that represents a hidden number of code digits in the form of a timestamp, which consists of the current date, hour, minute, second, and millisecond. The number of code digits is hidden in the second and millisecond part. Since the second and millisecond parts are concatenated, the variable representing the concatenation of the second and millisecond part is called ‘secondmillisecond’. Because the maximum second
value is 59 and the maximum millisecond value is 999, the maximum value of secondmillisecond is 59999. Eq. (2) is used to calculate the secondmillisecond values:

\[
\text{SecondMilliseconds} = \text{Random} + \text{numberOfCodeDigits}
\]  

The random value is obtained using the Linear Congruential Random Number Generator [4]. The random value synchronization pattern between the sender and the receiver is based on a trigger when the sender creates the cover text and when the receiver receives the cover text. The random value created by the sender and the receiver is always the same because it is made by the pseudo-random generator method based on a seed value that has been agreed upon before the send and the receiver communicate with each other for the first time. Since both parties synchronize the random value, it is necessary to check whether the receiver has received the message transmitted by the receiver or not [13,14]. The pseudo-code for embedding process is shown in Algorithm (6).

Algorithm 6

FUNCTION Message-Embedding(message, sentence-with-space-table, answer-choice-table, sentence-for-representing-shifting-code-table, Baudot-Murray-codes-table, synonym-list, frequently-used-sentence-pattern-table)

Input: message is secret message to be hidden, sentence-with-space-table for selecting sentence with space for inserting answer's Likert scale, answer-choice-table for selecting answer choice of Likert scale from code list digit, sentence-for-representing-shifting-code-table to represent shifting code, Baudot-Murray-codes-table for encoding, synonym-list, frequently-used-sentence-pattern-table for checking common and formal sentence patterns during the transformation process

Output: cover text

//get code-list
code-list = Message-Encoding(message, Baudot-Murray-codes-table);
sentences-of-digits = Get-Sentences(code-list, sentence-with-space-table, answer-choice-table, sentence-for-representing-shifting-code-table);
synonymous-sentences = DoSynonymSubstitution(sentences-of-digits, synonym-list, answer-choice-table, sentence-for-representing-shifting-code-table);
transformed-sentences = Transform-Sentences(synonymous-sentences, frequently-used-sentence-pattern-table);
sentence-for-number-of-digits = Create-Sentence-For-Representing-The-Number-Of-Code-Digits(code-list);
RETURN Concatenate(transformed-sentences, sentence-for-number-of-digits);
END FUNCTION

FUNCTION Get-Sentences(code-list, sentence-with-space-table, answer-choice-table, sentence-for-representing-shifting-code-table)

Input: code list as result message encoding, sentence-with-space-table for selecting sentence with space for inserting the answer's Likert scale, answer-choice-table for selecting answer choice on Likert scale from code list digit, sentence-for-representing-shifting-code-table to represent shifting code
Output: list of new sentences with answer choices in accordance with quinary
Likert scale or sentence from sentence-for-representing-shifting-code
//scanning for every quinary digit in code-list
//index-shifting for identifying index shifting position, set to first index = 1
index-shifting = 1
FOR every digit in code-list
//Check if 3 digits are shifting code. If yes, then get sentence from
//sentence-for-representing-shifting-code-table. If no, then get sentence from
//sentence-with-space-table
IF 3 digits are shifting code THEN
new-sentence = Get-Sentence-For-Representing-Shifting-Code(index-shifting);
index-shifting = index-shifting + 1;
ELSE
new-sentence = Get-From-Sentence-With-Space-For-Response-Table(digit);
Likert-scale-answer = Get-From-Answer-Choice-Table(digit);
new-sentence = Insert-Likert-Scale-Answer(Likert-scale-answer,new-sentence);
END IF
new-list-sentences.ADD(new-sentence);
END FOR
RETURN new-list-sentences;
END FUNCTION

FUNCTION DoSynonymSubstitution(list-of-new-sentences, synonym-list, answer-choice-table, sentence-for-representing-shifting-code-table)
Input: list-of-new-sentences after inserting the Likert scale of answer choice
or sentence for representing shifting code, synonym list
Output: list of sentences after conducted synonym substitution
//Parsing every word except keyword of Likert scale or
//keyword of sentence for representing shifting codes.
FOR every sentence in list-of-new-sentences
FOR every word in a sentence
//check if the phrase is a keyword
//try to get the phrase
phrase = Get-Phrase(word, list-of-new-sentences);
IF phrase is not in the answer-choice-table AND phrase is not
in the sentence for-representing-shifting-code-table THEN
word-class = Get-Word-Class(word); //using PATR Tool [10]
new-word = Get-Synonym(word, synonym-list);
Replace-Word-By-Its-Synonym(new-word, *sentence);
END IF
END FOR
sentences-after-synonym-substitution.ADD(sentence);
END FOR
RETURN sentences-after-synonym-substitution;
END FUNCTION

FUNCTION Message-Encoding(message, Baudot-Murray-table)
Input: secret message to be hidden and Baudot-Murray table
Output: list of Baudot-Murray code
//collecting bigrams of the message
current-code-position = LEFT //current-code-position to identify shifting code
FOR every bigram in message
//finding the bigram in Baudot-Murray-table
code = Get-Code-From-Baudot-Murray-Table(bigram, Baudot-Murray-table);
IF code is not found THEN //finding the unigram's code
unigram = bigram[0];
code = Get-Code-From-Baudot-Murray-Table (unigram, Baudot-Murray-table);
END IF
IF code.POSITION <> current-code-position THEN
  current-code-position = code.POSITION;
  code = Concatenate( '444', code.quinary); //add shifting code
END IF
list-codes.ADD(code.digit);
END FOR
RETURN list-codes;
END FUNCTION

FUNCTION Transform-Sentences(sentences-after-synonym-substitution, table-of-frequently-used-sentences)
Input: sentences-after-synonym-substitution, table-of-frequently-used-sentences for checking general and formal sentence pattern while the transformation process
Output: list of sentences after the transformation process

//Scanning for every sentence in sentences-after-synonym-substitution
FOR every sentence in sentences-after-synonym-substitution
  //check if the element of sentence only subject and predicate or vice versa
  //have a chance to be transformed into an inverted sentence
  IF LEN(sentence-elements) are 2 THEN
    new-sentence = Inverted-Transformation(sentence, sentence-elements, table-of-frequently-used-sentences);
  ELSE
    //check if an active sentence can be transformed into a passive sentence or vice versa
    sentence-elements = Get-Sentence-Elements(sentence); //using PATR Tool [10]
    verb = Get-Predicate(sentence-elements);
    prefix = Get-Prefix(verb);
    IF prefix is ‘me’ OR ‘di’ THEN //can be transformed
      new-sentence = Active-Passive-Transformation(sentence, sentence-elements, table-of-frequently-used-sentences);
    END IF
    //check if the adjunct position in the sentence can be changed
    adjunct = Get-Adjunct-Element(sentence-elements);
    IF adjunct is at the first OR at the end of sentence element THEN
      new-sentence = Adjunct-Position-Transformation(sentence, sentence-elements, table-of-frequently-used-sentences);
    END IF
  END IF
  list-of-sentences-transformed.ADD(new-sentence);
END FOR
RETURN list-of-sentences-transformed;
END FUNCTION

FUNCTION Inverted-Transformation(sentence, sentence-elements, table-of-frequently-used-sentences)
Input: sentence after synonym substitution process, sentence-elements are elements as a result parsing process to define sentence transformation, table-of-frequently-used-sentences as transformation rule
Output: sentence inverted or vice versa

//swapping words based on their element
first-phrase = Get-First-Phrase(sentence-elements);
second-phrase = Get-Second-Phrase(sentence-elements);
IF first element of sentence-elements is predicate THEN
  new-sentence = Concatenate(second-phrase, first-phrase);
ELSE
  new-sentence = Concatenate(first-phrase, second-phrase);
END IF
RETURN Check-Common-Structure(sentence, new-sentence, sentence-elements,
table-of-frequently-used-sentences);
END FUNCTION

FUNCTION Active-Passive-Transformation(sentence, sentence-elements,
table-of-frequently-used-sentences)
Input: sentence after synonym substitution process,
sentence-elements are elements as parsing result
process to define sentence transformation,
table-of-frequently-used-sentences as rule transformation
Output: passive sentence or vice versa
//get verb of predicate of sentence and check if it supports
//active to passive transformation
verb = Get-Verb-Of-Predicate(sentence);
original-word = Get-Original-Word(verb);
suffix = Get-Suffix(*verb, original-word);
IF prefix of *verb is 'me' THEN
  new-verb = Concatenate('di', original-word, suffix);
ELSE
  new-verb = Concatenate('me', original-word, suffix);
END IF
new-subject = Get-Object-Element(sentence-elements);
new-object = Get-Subject-Element(sentence-elements);
new-sentence = Concatenate(new-subject, new-verb, new-object);
RETURN Check-Common-Structure(sentence, new-sentence, sentence-elements,
table-of-frequently-used-sentences);
END FUNCTION

FUNCTION Adjunct-Position-Transformation(sentence, sentence-elements,
table-of-frequently-used-sentences)
Input: sentence after synonym substitution process,
sentence-elements are elements as parsing result
to define sentence transformation,
table-of-frequently-used-sentences as transformation rule
Output: sentence after adjunct position changes
subject = Get-Subject-Element(sentence-elements);
predicate = Get-Predicate-Element(sentence-elements);
object = Get-Object-Element(sentence-elements);
//swapping adjunct element position
adjunct = Get-Adjunct-Element(sentence-elements);
IF adjunct is first element of sentence THEN
  new-sentence = Concatenate(subject, predicate, object, adjunct);
ELSE
  //add coma if adjunct is at the first of sentence
  new-sentence = Concatenate(adjunct, ',', object, subject, predicate);
END IF
FUNCTION Check-Common-Structure(sentence, new-sentence, sentence-elements, table-of-frequently-used-sentences)
Input: sentence is the sentence before transformation,
new-sentence is the sentence after transformation,
sentence-elements are the result of parsing,
table-of-frequently-used-sentences to check valid common
and formal sentence pattern
Output: new sentence after checking if its pattern is common and formal
//parsing to check the generality and formality of the sentence
//if no, revert to the original sentence
IF sentence-elements are in table-of-frequently-used-sentences THEN
RETURN new-sentence;
ELSE
RETURN original-sentence;
END IF
END FUNCTION

FUNCTION Create-Sentence-For-Representing-The-Number-Of-Code-Digits(code-list)
Input: code-list as result of message encoding
Output: sentence for representing the number of code digits
//Get new random value using Linear Congruential Method [4]
new-random-value = Random-Value();
second-millisecond = FORMAT(new-random-value + LEN(code-list), 'HH.MMM');
pre-sentence = 'Laporan ini dihasilkan secara otomatis oleh sistem pada
tanggal ';
RETURN Concatenate(pre-sentence, GetCurrentDate(), second-millisecond);
END FUNCTION

FUNCTION Random-Value()
previous-random = Get-Previous-Random-From-Database();
//the value 16672,4 and 50013 is the best composition for generating optimum
//random value between 0-59999 (max secondmillisecond), which uses Linear
//Congruential Method
new-random-value = (16672 * previous-random + 4) modulo 50013;
Save-New-Random-Value-Into-Database(new-random-value);
RETURN new-random-value;
END FUNCTION

On the receiver side, the extraction process is started by finding the number of
code digits representing the secret message from the timestamp using Eq. (2).
Furthermore, the keywords in the sentences are identified, where the number of
keywords is identical to the number of code digits. The code digits form codes to
decode the secret message using the Baudot-Murray code table. The pseudo-
codes for extraction and decoding are shown in Algorithm (7) and Algorithm (8),
respectively.

Algorithm 7
FUNCTION Message-Extraction(cover-text, answer-choice-table, sentence-for-representing-shifting-code-table)
Input: cover-text received by the receiver,
answer-choice-table as a reference to check keywords,
sentence-for-representing-shifting-code-table as a reference to check keywords
Output: code-list to decode the message
//scanning for every sentence in the cover-text
FOR every sentence in the cover-text
//try to identify a keyword in the sentence
//search keyword by phrase of sentence
WHILE keyword is not found
phrase = Scan-Phrase(sentence);
keyword = Find-Keyword(phrase, answer-choice-table);
END WHILE
IF keyword is not found THEN
digit = ‘444’; //the keyword must be shifting code
ELSE
digit = Get-Digit(phrase, the answer-choice-table);
END IF
codes.ADD(digit);
END FOR
RETURN codes;
END FUNCTION

Algorithm 8
FUNCTION Message-Decoding(code-list, Baudot-Murray-code-table)
Input: code-list from message-extraction process, Baudot-Murray-code-table as reference to decode code-list into the message
Output: the message sent by the sender
//scanning for every code in code-list
//code is 3 digits of the quinary number
current-position-of-code = LEFT;
FOR every code in code-list
chars = Get-Chars-From-Baudot-Murray-Table(code);
IF code is shifting code THEN
current-position-of-code = NOT current-position-of-code; //swapping
END IF
//get chars from Baudot-Murray code table
chars = Get-Chars(code, current-position-of-code, the answer-choice-table);
message = Concatenate(message, chars);
END FOR
RETURN message;
END FUNCTION

4 Experimental Evaluation
The embedding capacity was evaluated by conducting 300 message embeddings, where the length of each message was greater than six. Efficiency was gained whenever a message contained a shifting code, because the proposed method only uses one message for a shifting code, while Wibowo’s method uses three messages. To observe the embedding capacity, the number of messages should be greater than six, which both methods can accommodate.
Based on Figure 4, it can be seen that only 1% of messages could not be embedded by both methods; 34% of messages could be embedded by Wibowo’s method; 99% of messages could be embedded by the proposed method. Thus, it can be concluded that the proposed method can embed 99% of messages with more than six characters.

The security level for representing the number of code digits in the proposed method depends on the number of code digits hidden in a timestamp, which is represented in Eq. (2). Thus, the security level can be calculated by calculating the probability of guessing the secondmillisecond value, which is hidden in the cover text. Since the secondmillisecond value is random, where the random value is kept secret by both parties, it makes obtaining the number of code digits very difficult. Finally, we can conclude that the security level depends on the probability of obtaining the number of code digits represented in Eq. (3), while in Wibowo’s method, the guessing probability of the number of code digits is 1, because the key can be directly obtained from the timestamp. A comparison of the security levels of both methods is shown in Table 5.

\[
P(\text{numberOfCodeDigits}) = \frac{1}{\text{Maximum(Random)}}
\]  

Table 5  Probability comparison for guessing the number of code digits between Wibowo’s and the proposed method.

<table>
<thead>
<tr>
<th>Wibowo’s Method</th>
<th>Proposed Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(\frac{1}{\text{Maximum(Random)}})</td>
</tr>
</tbody>
</table>

In Wibowo’s method, sentences are manually generated, which may cause human grammatical and semantic errors. A sentence with grammatical and semantic errors will cause unnaturalness. Based on linguistic steganography analysis, an
unnatural sentence could raise the suspicion of the attacker. Therefore, it is essential to evaluate the naturalness of the sentences to increase the imperceptibility of the cover [8].

In this study, an evaluation of the naturalness of the sentences in the cover text was conducted based on human judgment, using thirty respondents. The respondents were divided into two groups, namely experts and non-experts. The experts were linguistic lecturers, and the non-experts were undergraduate students, nonlinguistic lecturers, and ordinary employees. To evaluate the sentences, questionnaires about the naturalness of the sentences, from a grammar as well as a semantic point of view, had to be answered by the respondents. The evaluation was conducted on sentences generated by Wibowo’s method and the proposed method.

Figures 5 and 6 show that Wibowo’s method had three sentences with a naturalness value less than 100%, namely, sentences 3, 9 and 10, whereas the proposed method had two sentences with a naturalness value less than 100%, namely, sentences 1 and 14. Thus, the average sentence naturalness percentage of Wibowo’s method was 98.61%, while the naturalness percentage of the proposed method was 99.89%. In this case, the naturalness of the proposed method was higher than that of Wibowo’s method.

![Figure 5](image_url)  
**Figure 5** Naturalness comparison between sentences generated by Wibowo’s and the proposed method (sentence 1 to 9).
Since the security of both methods depends on the naturalness of the sentences generated, the security of the proposed method is higher than that of Wibowo. This is because the cover text generated by Wibowo’s method is more suspicious to attackers than the sentences generated by the proposed method. Thus, it is easier for attackers to get the number of digits hidden in the timestamp with Wibowo’s method than with the proposed method.

The worst case occurs when the full message length cannot be embedded by the proposed method and a number of questions from the questionnaire and sentences representing shifting codes has to be added. For example, the message ‘DOMINO ASLI O’ results in codes consisting of the following six bigrams and one unigram: 444 122 444 430 444 033 444 140 444 430 444 233 444 002. In this case, representing one bigram or one unigram requires three sentences, while one sentence is required to represent the shifting codes. Therefore, calculating the number of sentences required to represent the secret message and the number of sentences required to represent the shifting codes can be defined as follows in Eq. (4):

$$NSS(sm) = NSSB(sm) + NSSU(sm) + NSSS(sm)$$  \hspace{1cm} (4)

where NSS is the number of sentences required to represent a secret message, consisting of the number of sentences required to represent bigrams (NSSB), unigrams (NSSU), and shifting codes (NSSS).
5 Conclusion

Based on the experimental result, it can be concluded that the embedding capacity of cover texts using the proposed method is higher than that of Wibowo’s method. This is because the proposed method decreases the number of sentences that have to be used to represent shifting codes. Another contribution of this research is that the security level of the number of code digits was improved by introducing a private key, which is very difficult for attackers to guess. To maintain the naturalness of the generated sentences, sentence paraphrasing is used.

In this study, the sentences representing shifting codes were created manually. In the future, it is necessary to build these automatically by keeping the context of the sentence before and after the sentences that represent the shifting code. This would allow all sentences to be built dynamically.

References


