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].

Received September 29th, 2021, Revised May 27th, 2022, Accepted for publication June 8th, 2022.
Copyright © 2022 Published by IRCS-ITB, ISSN: 2337-5787, DOI: 10.5614/itbj.ict.res.appl.2022.16.2.1

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.

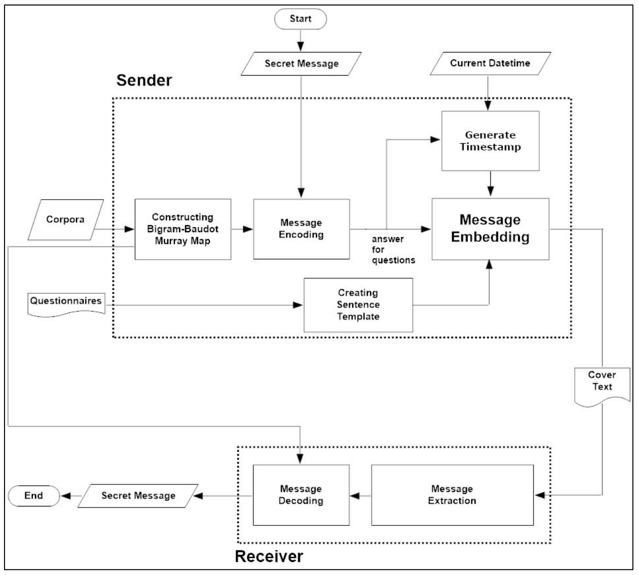

Figure 1 Overview of INORMALS.

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.

Code	Left	Right	No.	Code	Left	Right	No.	Code	Left	Right
000	AN	IM	42	132	TE	_G	84	314	TR	AJ
001	N_	EP	43	133	SA	OD	85	320	_E	HW
002	NG	_O	44	134	AH	IG	86	321	ES	UJ
003	_D	AU	45	140	_A	EY	87	322	RT	RC
004	EN	E_	46	141	H_	_F	88	323	RU	_W
010	A_	ID	47	142	MA	IB	89	324	GG	UB
011	DA	_R	48	143	BE	IF	90	330	GU	VE
012	I_	LU	49	144	ON	RD	91	331	RE	RP
013	ER	KI	50	200	IK	PI	92	332	KU	SP
014	KA	MO	51	201	RI	NU	93	333	RO	FO
020	_P	AB	52	202	HA	LO	94	334	AY	CO
021	_M	IH	53	203	TI	D_	95	340	BI	SK
022	AR	P_	54	204	_Y	SY	96	341	JA	SH
023	YA	KS	55	210	NI	FI	97	342	SU	A
024	PE	RK	56	211	U_	FA	98	343	NS	B
030	LA	TO	57	212	KE	RM	99	344	_L	C
031	ME	PU	58	213	IT	RH	100	400	BU	D
032	AT	EH	59	214	IA	O_	101	401	OR	E
033	AK	NO	60	220	AI	VA	102	402	KT	F
034	G_	RS	61	221	UA	OS	103	403	UT	G
040	_S	DU	62	222	NT	NC	104	404	UM	H
041	_T	GK	63	223	AM	JI	105	410	AA	I
042	RA	NK	64	224	UK	JE	106	411	ET	J
043	TA	IR	65	230	AD	CE	107	412	_H	K
044	SI	UL	66	231	_U	KN	108	413	AG	L
100	DI	PR	67	232	IS	F_	109	414	CA	M
101	IN	_C	68	233	LI	RL	110	420	_J	N
102	GA	RB	69	234	NY	GO	111	421	GI	O
103	AL	NJ	70	240	LE	NN	112	422	_N	P
104	BA	OM	71	241	AP	_V	113	423	US	Q
110	_K	UP	72	242	_I	NF	114	424	OL	R
111	UN	Y_	73	243	NE	UI	115	430	MI	S
112	_B	HI	74	244	R_	RN	116	431	KO	T
113	SE	EG	75	300	DE	IO	117	432	ST	U
114	AS	HU	76	301	M_	OT	118	433	MP	V
120	NA	UH	77	302	EB	UG	119	434	UR	W
121	K_	GE	78	303	S_	OG	120	440	JU	X
122	TU	DO	79	304	MB	KR	121	441	WA	Y
123	T_	UD	80	310	IL	RJ	122	442	MU	Z
124	EL	PO	81	311	EK	TK	123	443	ED	_
130	EM	IP	82	312	L_	AW	124	444	SHIFTING	
131	PA	EC	83	313	ND	YE				

Note: '_' is space.

Table 2 Sentence templates.

Question	Answer	Sentence Pattern	Sentences
0	0	0	*Anda / sama sekali tidak merasa terbantu / dalam mengelola aktivitas keuangan Anda / sejak menggunakan e-money*
0	0	1	*Sejak menggunakan e-money / Anda / sama sekali tidak merasa terbantu / dalam mengelola aktivitas keuangan anda /*
0	0	2	*Sejak menggunakan e-money / Anda / sama sekali tidak merasa terbantu / dalam mengelola aktivitas keuangan Anda / sehari-hari /*

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):

$$\begin{aligned} &\text{second} = \text{hour} + \text{minute} - \text{numberOfCodeDigits} \\ &\text{millisecond} = \text{random}(0 - 499) \text{ if second} \leq 0 \\ &\text{millisecond} = \text{random}(500 - 999) \text{ if second} > 0 \end{aligned} \tag{1}$$

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.

Anda sama sekali tidak merasa terbantu dalam mengelola aktivitas keuangan Anda sejak menggunakan e-money.
Anda terkadang bisa bertransaksi lebih cepat dengan e-money, walaupun Anda sangat jarang bertransaksi lebih tepat dengannya.
Anda hampir tidak pernah membawa banyak uang tunai sejak menggunakan e-money.
Anda sering sekali mendapatkan manfaat khusus dari merchant sejak menggunakan e-money.
Keluarga Anda sangat menyarankan Anda menggunakan e-money, walaupun teman-teman Anda ragu-ragu menyarankan Anda menggunakannya.
Orang-orang di lingkungan Anda menentang Anda menggunakan e-money.
Anda mengikuti komunitas dimana banyak anggotanya yang rutin menggunakan e-money.
Anda menilai besaran biaya layanan e-money tidak wajar.
Anda sangat berkeberatan dengan besaran biaya awal dari layanan e-money, walaupun Anda setuju dengan besaran biaya transaksi dari layanannya.
Anda berpendapat layanan e-money memberikan manfaat yang sangat buruk dilihat dari besaran biaya yang diperlukan.
Anda berpendapat layanan e-money memberikan nilai yang biasa saja dilihat dari besaran biaya yang diperlukan.
Anda terkadang menggunakan layanan e-money untuk berbelanja.
Anda sering menggunakan layanan e-money untuk membeli pulsa telekomunikasi atau listrik prabayar.
Anda sering sekali menggunakan layanan e-money untuk membayar tagihan.
Anda terkadang menggunakan layanan e-money untuk transfer uang.
Anda sering menggunakan layanan e-money untuk tarik uang tunai.
Laporan ini dihasilkan secara otomatis oleh sistem pada tanggal 2020-08-17 jam 17:17:32:524.

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.

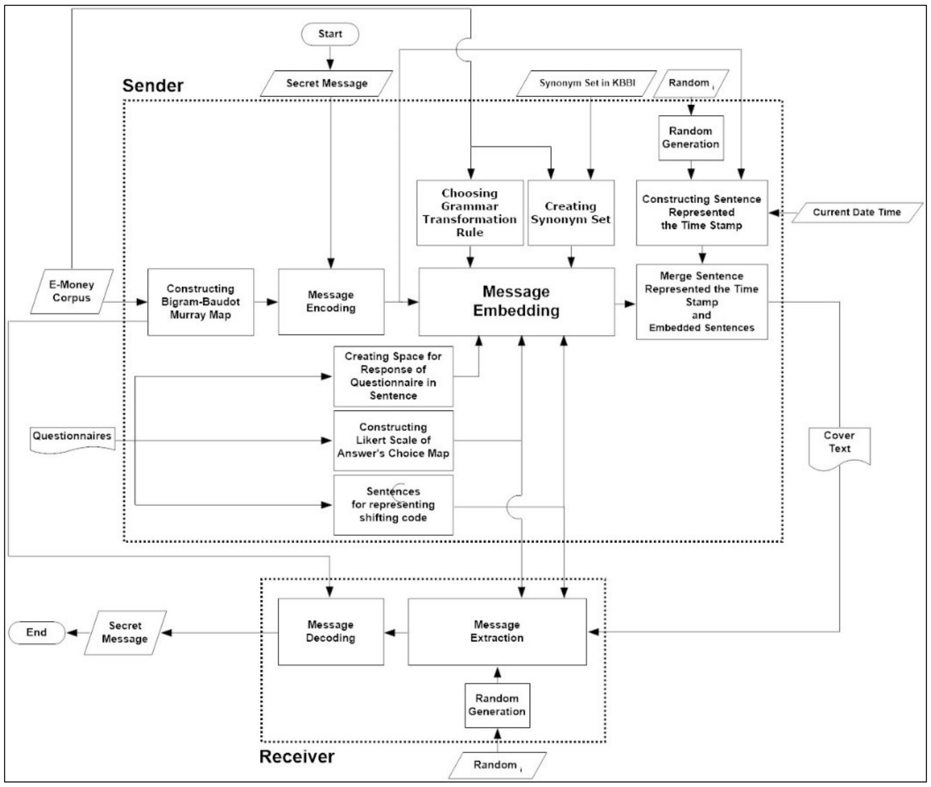
Start
Sender
Secret Message
Synonym Set in KBBI
Random
Random Generation
Choosing Grammar Transformation Rule
Creating Synonym Set
Constructing Sentence Represented the Time Stamp
Current Date Time
E-Money Corpus
Constructing Bigram-Baudot Murray Map
Message Encoding
Message Embedding
Merge Sentence Represented the Time Stamp and Embedded Sentences
Creating Space for Response of Questionnaire in Sentence
Questionnaires
Constructing Likert Scale of Answer's Choice Map
Sentences for representing shifting code
Cover Text
End
Secret Message
Message Decoding
Message Extraction
Random Generation
Receiver
Random

Figure 3 Overview of the proposed method.

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
  //of each question in the questionnaire
  FOR every question in the questionnaire
    //1-5 Likert scale option
    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 $3^{rd}$ 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 3 Table of sentence transformation rules.

Rule	Sentence Pattern	Transformation	Type of Sentence
1	Subject – Predicate	Predicate – Subject	Inverted or vice versa
2	Subject – Predicate – Object	Subject – Predicate - Adjunct	Active to passive or vice versa
3	• Subject – Predicate – Object – Adjunct • Subject – Predicate – Adjunct	• Adjunct – Subject – Predicate – Object • Adjunct – Subject – Predicate	Adjunct position changes

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.

Syntactical	No.	Syntactical
A-C-S-P	8	S-P
A-S-P	9	S-P-A
A-S-P-O	10	S-P-O
S-A-P	11	S-P-O-A
S-A-C-P	12	S-P-O-C
S-CONJ-P-O-P-A	13	S-P-O-C-A
S-CONJ-P-O-P-O	14	S-P-C

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
  //sort the most to least
  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*'. Next, the words of the sentence will be substituted using synonym sets excluding the keyword. For example, the sentence becomes '*Fasilitas e-money benar-benar tidak menunjang anda dalam menjalankan 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:

$$SecondMillisecond = Random + numberOfCodeDigits \tag{2}$$

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 tranformation 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
      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, ‘, ‘, subject, predicate, object);
  END IF
  RETURN Check-Common-Structure(sentence, new-sentence, sentence-elements,
         table-of-frequently-used-sentences);
END FUNCTION

FUNCTION Check-Common-Structure(original-sentence, new-sentence,
         sentence-elements, table-of-frequently-used-sentences)
  Input: original-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.

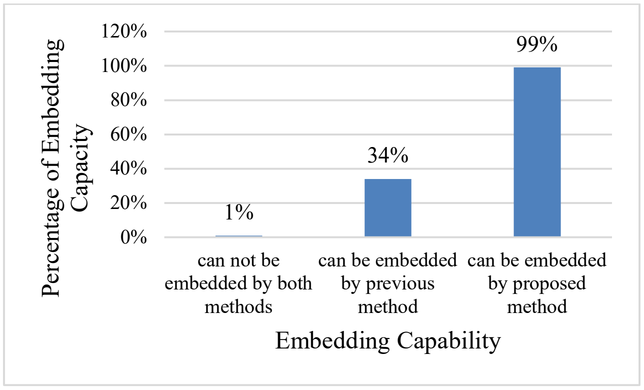

Figure 4 Comparison of embedding capacity.

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.

$$\mathrm{P(numberOfCodeDigits)} = \frac{1}{\mathrm{Maximum(Random)}} \tag{3}$$

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

Wibowo's Method	Proposed Method
1	$\frac{1}{\mathrm{Maximum(Random)}}$

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.

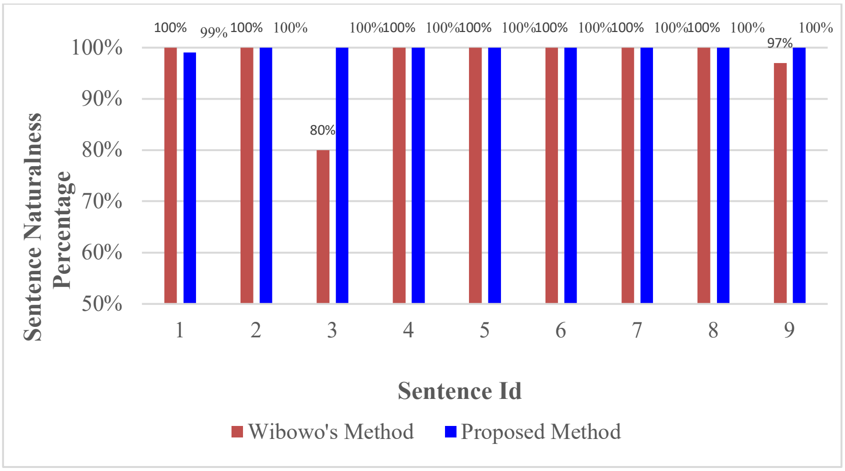

Figure 5 Naturalness comparison between sentences generated by Wibowo's and the proposed method (sentence 1 to 9).

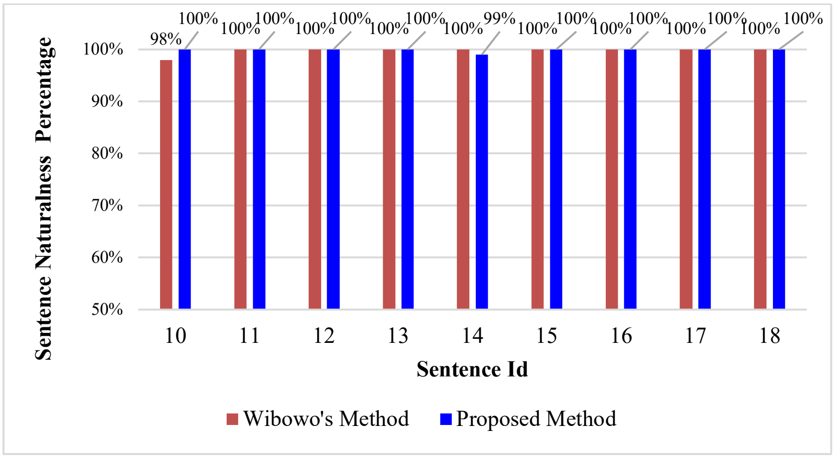

Figure 6 Naturalness comparison between sentences generated by Wibowo's and the proposed method (sentence 10 to 18).

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):

$$\mathrm{NSS(sm)} = \mathrm{NSSB(sm)} + \mathrm{NSSU(sm)} + \mathrm{NSSS(sm)} \tag{4}$$

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

[1] Desoky, A., *NORMALS: Normal Linguistic Steganography Methodology*, Journal of Information Hiding and Multimedia Signal Processing, **1**(3), pp. 145-171, July 2010.

[2] Desoky, A., *Mature Linguistic Steganography Methodology (Matlist)*, Journal of Security and Communication Networks, **4**(1), pp. 697-718, 2010.

[3] Wibowo, A. & Barmawi, A.M., *INORMALS Improving Using the Modified Baudot-Murray Code*, ICCNS '16: Proceedings of the 6th International Conference on Communication and Network Security, pp. 113-118, 2016.

[4] Hull, T.E. & Dobell, A.R., *Random Number Generators*, Journal of Society for Industrial and Applied Mathematics, **4**(3), pp. 230-254, July 1962.

[5] Muhammad, A. & Barmawi, A.M., *Paraphrasing Method Based on Contextual Synonym Substitution*, Journal of ICT Research and Applications, **13**(3), pp. 257-282, 2019.

[6] Alwi, H., Dardjowidjojo, S., Lapoliwa, H. & Moeliono, A.M., *Indonesian Dictionary*, Ed. 4, Balai Pustaka, 2012. (Text in Indonesian)

[7] Sneddon, J.N., Adelaar, A., Djenar, D.N. & Ewing, M.C., *Indonesian Reference Grammar*, Ed. 2, Allen & Unwin, 2008.

[8] Chang, C.Y. & Clark, S., *Practical Linguistic Steganography using Contextual Synonym Substitution and a Novel Vertex Coding Method,* Journal of Computational Linguistics, **40**(2), pp. 403-448, 2013.

[9] Reiter, E. & Dale, R., *Building Natural Language Generation Systems*, The Press Syndicate of the University of Cambridge, 2000.

[10] Muhammad, A. & Kamariah, K. *Banjarese Sentence Parser Using PC-PATR Parser*, Jurnal Linguistik Komputasional, **3**(1), pp. 20-23, 2020. (Text in Indonesian)

[11] Jurafsky, D. & Martin, J.H., *Speech and Language Processing: An Introduction to Natural Language Processing*, Computational Linguistics, and Speech Recognition, Ed. 2, Prentice Hall, 2008.

[12] Hammarström, H., *Rarities in Numeral Systems*, Rethinking Universals, pp. 11-60, 2010.

[13] Hunt, C., *TCP/IP Network Administration*, ed. 3, O'Reilly, 2002.

[14] Mozilla Documentation, *HTTP Response Status Codes*, https://developer.mozilla.org/enUS/docs/Web/HTTP/Status#client error responses, (7 February 2021).