BŞEÜ Fen Bilimleri Dergisi
7(2), 739-754, 2020

BSEU Journal of Science
DOI: 10.35193/bseufbd.634578

e-ISSN: 2458-7575 (http://dergipark.gov.tr/bseufbd)

Araştırma Makalesi - Research Article

Bilgiyi Matematiksel İfadeye Gizlemek İçin Yeni Bir Yaklaşım

Muhammed Milani[1*]

Geliş / Received: 18/10/2019 ***Revize / Revised: 07/06/2020*** ***Kabul / Accepted: 22/06/2020***

ÖZ

Bilgi gizleme son zamanlarda çok ilgi gördü. Steganografi için çeşitli yöntemler geliştirilmiş ve aynı zamanda gizli verileri tespit etmek için uygun steganalizler tasarlanmıştır. Bununla birlikte, steganaliz tarafından dikkate alınmamasi için yeni bir kapak türünde sunulan bir yaklaşım daha az şüpheli olabilir. Bu makale bir mesajı matematiksel ifadeye dönüştürebilen bir yöntem önermektedir. Oluşturulan matematiksel ifade, oldukça güvenli bir metinle birlikte mesajı iletmek için bir kapak olarak kullanılabilir. Önerilen yöntem, uygun bir stokastik dilbilgisi kullanarak, belirli bir metne dayalı matematiksel bir ifade oluşturma yeteneğine sahiptir. Dilbilgisi kurallarının olasılığı, taşıyıcı metne göre ve matematiksel ifadelerin türüne göre belirlenebilir.

Anahtar Kelimeler- Bilgi Gizleme, Steganografi, Steganaliz, Matematiksel İfade, Stokastik Dilbilgisi

[1*]Sorumlu yazar iletişim: mmilani@bandirma.edu.tr (https://orcid.org/0000-0003-2450-0280)
Bilgisayar Mühendisliği, Bandırma Onyedi Eylül Üniversitesi, Balıkesir/Bandırma - Türkiye

BŞEÜ Fen Bilimleri Dergisi
7(2), 739-754, 2020

BSEU Journal of Science
DOI: 10.35193/bseufbd.634578

e-ISSN: 2458-7575 (http://dergipark.gov.tr/bseufbd)

A Novel Approach for Hiding Information into Mathematical Expression

ABSTRACT

Information hiding has received a lot of attention recently. Various methods have been developed for steganography, and at the same time, proper steganalysis has been designed to detect hidden data. However, an approach that is introduced in a new type of cover can be less doubtful due to the lack of attention to steganalysis analyses. This paper proposes a method that can convert a message into a mathematical expression. The generated mathematical expression can be used as a cover to transmit the message along with a highly secure text. The proposed method, using a suitable stochastic grammar, has the ability to create a mathematical expression based on a given text. Probability of grammar rules can be determined in accordance with the carrier text and based on the type of mathematical expressions.

Keywords- Information Hiding, Steganography, Steganalysis, Mathematical Expression, Stochastic Grammar

BŞEÜ Fen Bilimleri Dergisi
7(2), 739-754, 2020

BSEU Journal of Science
DOI: 10.35193/bseufbd.634578

e-ISSN: 2458-7575 (http://dergipark.gov.tr/bseufbd)

I. INTRODUCTION

Cryptography is a widely used technique for secret communication [1] and is applied by various encryption and decryption methods. However, cryptography alone is not enough for secure information transmission because the encrypted text may attract attention. To address this problem, steganography is one of the ways proposed to send information securely using stego-texts that cause the slightest doubt. Typically, in these systems, texts can be concealed as messages in a variety of covers. An inserted message should only cause subtle changes in the covers. Generally, the covers can be classified in four categories: text, image, video, and audio [2]. Of course, there may be other covers that are not included in these four categories, such as executable files [3]. Among these categories, image-based steganography has been used more widely and data is included in parts of the image that is less recognizable [4].

Video is also known as a cover that requires a large number of operations to insert the message [5]. But, because of the ability to insert a large volume of messages, they have attracted attentions [6]. Inserting a message in audio is also possible. Usually, the human ear can comprehend sounds that range from 20HZ to 20KHZ [7]. So, in some audio-type covers, this feature is used to insert data in an unrecognizable manner [8]. Texts have also been used as a cover in some steganography methods [3]. Although the number of text-based methods is less than other methods, the large volume of text data-type in these methods has made them attractive.

In contrast to steganography, steganalysis [9] is the science of unauthorized extraction of the inserted message. It can be said that steganalysis and steganography are similar to a hide-and-seek game [10]. With the development of a variety of methods for steganography, several methods have been developed for Steganalysis [11]. Obviously, the new methods are more reliable in secure information transmission as the steganalysis has not focused on new covers yet.

In this paper, a method for the steganography of data in mathematical expressions is proposed as a new cover generation method. In terms of the characteristics and nature of mathematical expressions, they can be a good place to hide information. Efforts have been made to produce texts that seem real [12]. With the same view, it is possible to propose ways to generate mathematical expressions that are not ambiguous.

Mathematics has an important root in human evolutionary history. Computers are very good at solving problems if correct commands are used. Symbolic computation or algebraic computation refers to the development and evaluation of mathematical expressions. Computer applications that can conduct symbolic calculations are called Computer Algebra Systems (CAS) or symbol manipulation systems. The increasing use of CAS systems has enlarged the rule of computer systems in the various areas. For example, computer algebra has an important role in designing and experimenting formulas that are required in numerical programs.

To hide sensitive data, we can also use the techniques that are proposed in CAS and provide a suitable platform for steganography systems. In this paper, using the concept of CAS, an efficient method of Cover Generation is proposed. The proposed method generates mathematical expressions using the rules that are chosen by the message. The generated expression is the cover that carries the message. Because math phrases have not yet been used to insert the message, it can cause the least sensitivity to steganalysis systems.

In Section 2, Cover Generation Steganography systems are expressed, and then the mathematical expressions and the operations that are applicable to them are described. These operations involve producing, parsing, and displaying math phrases. Also, in this section we will propose a new method for insertion and retrieval of a message. Finally, we evaluate the proposed method and conclude in Section 3.

II. MATERIALS and METHODS

A. Cover Generation Steganography

Steganography involves hiding information in cover objects; be it video, audio, image or even text. Unlike other steganography methods in which we decide which cover object (audio, video, text or image) to hide secret

message in, the cover generation technique does the opposite. In the cover generation technique, the steganography system generates the cover object.

Instead of embedding information into a cover, it is possible to generate the cover based on the message by using rules of a proper grammar. Thus, the generation cover can be sent as destinations for message carriers. If the natural cover is produced, this method can be transmitting data with higher confidence. Because general covers are mostly checking by steganalysis.

A well-known method in linguistic steganography is mimic functions [12]. Mimic functions are used to generate cover text using statistical methods which can somehow make sure that the generated message cannot be detected by computers outfitted with statistical profiles as humans cannot read the huge traffic of information passing through networks. The mimic function is built from a function called Huffman compression, and used to make the machine crazy of fool so that the mimic texting will be looking completely meaningless or like nonsense text, just because of the error of a grammar created to fool the machine. In automated generation with the help of style source and large and well-organized dictionary of words, secret message bits can be transformed into stego-text which cannot be detected easily by the adversaries [13].

On the other hand, there are a variety of algorithms which can implement cover generated audio steganography like the one proposed in [14]. In this method, the sender establishes the vital basic rules (i.e. amplitude, scale to be used and sampling rate) of the communication. Furthermore, as proposed in [15], a cover image can be generated from text using an immature technique. The technique can be implemented in both gray scale and RGB images. In this paper, the production of mathematical expressions as a new medium has been taken into consideration, and a method has been proposed using proper grammar rules to embeda message into a mathematical expression.

B. Mathematical Expressions

Mathematical expressions are held in different forms according to the special structure. One of the most common methods is the tree structure. The relationship between the operators and operands of an expression and its structure can be shown graphically by using expression trees. Each operator is represented by a node. Operators with the lowest precedence appear at the top of the tree, whereas the operators with higher precedence are put at the bottom of the tree. Each level shows the connection between an operator and its operand or operands.

Processes such as production, traversing and conversion into readable forms can be performed on expression trees. However, a proper grammar is required to work with this structure. Context Free Grammars (CFG) is used to parse and produce mathematical expressions. Input string passes through a parser which verifies the syntax of that input. At the same time an Abstract Syntax Tree (AST) which is a suitable structure to work with mathematical expressions is created. This tree serves as the main core part of the proposed methodology to apply various improvement algorithms on input data. The next section will address the development of an appropriate grammar for the proposed approach.

1) Grammar Definition: In this paper, we will focus on mathematical rules and introduce some grammars to work with them. Backus-Naur Form (BNF) grammar is useful to define context-free grammars in programming languages as it has simple notations; recursive structures, and widely available. We use this type of grammars because BNF is supported by many compiler generation tools such as YACC [16], LEX [17], and JavaCC[18].

Each rule in this grammar has a non-terminal on the left side and a collection of terminals and non-terminals on the right. During the language generation, depending on the top-down or bottom-up parsing method, a rule will be expanded or collapsed.

BŞEÜ Fen Bilimleri Dergisi
7(2), 739-754, 2020

BSEU Journal of Science
DOI: 10.35193/bseufbd.634578

e-ISSN: 2458-7575 (http://dergipark.gov.tr/bseufbd)

List 1. An E-BNF grammar for mathematical expressions

```
G = {Σ, T, V, P, S}
V = {expr, op, func, var, number, digit}⊆ Σ
T = {x, Sin, Cos, Tan, Log, Exp, Sqrt, +, -, *, /}⊆Σ
Σ = T ∪ V
S = { expr }
Production :
<expr>::= <expr><op><expr>
        |(<expr>)
        |<func>( <expr> )
        |<var>
        |<Number>
<op>::='+' | '-' | '*' | '/' | '^'
<func>::= 'Sin' | 'Cos' | 'Tan' | 'Log' |'Exp' | 'Sqrt'
<var>::='x'
<number>::= '-' ? <digit> + ('. ' <digit> +)?
<digit>::=[' 0'-'9']
```

Mathematical expressions as mentioned in this paper can contain operations such as addition or subtraction, functions such as sin or cos, special symbols such as integral, etc. Given a grammar developed for a particular mathematical expression, all operators, functions, and symbols, variables and numbers will be members of the terminal set. The non-terminal set will be determined based on the production rules of the grammar. The designed grammar must generate arithmetic expressions with an operator and its operands, or a mathematical function with arguments. The operand or argument itself might be a number, a variable, or another mathematical expression.

The production rules of a grammar might be recursive since there are operands of the expression type. For various types of mathematical expressions, different grammars can be used. In this section, a derivative system for mathematical expressions with some modifications to a grammar mentioned in [19] is developed. This grammar is called Extended-BNF grammar and shown in List 1.

The extended-BNF in List has five operators and six functions. A simple mathematical expression usually contains these operators. However, it can be modified with the insertion of some other operators, various functions, and particular symbols.

2) Grammar Conversion: The grammar defined in the previous section is general and can generate any math expressions. However, this grammar has a problem of generating more than one expression tree for a given expression. As a simple example, two different expression trees for "7 + 2 + 5" are given in Figure 1.

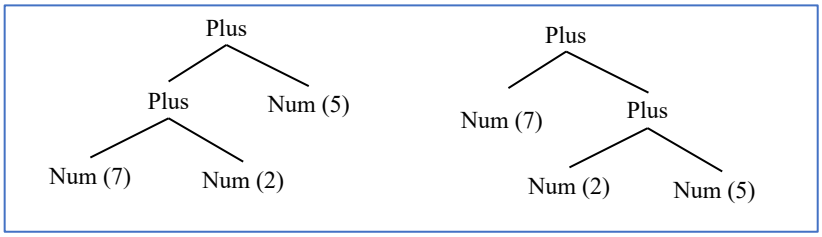

Figure 1. Two deferent ASTs for the "7+2+5" expression

This situation may cause major problems for the proposed method in this paper even if there are no problems with some systems. In steganography, this can lead to the extracting of a different message from the embedded message. Therefore, there is a need to develop the grammar appropriately.

BŞEÜ Fen Bilimleri Dergisi
7(2), 739-754, 2020

BSEU Journal of Science
DOI: 10.35193/bseufbd.634578

e-ISSN: 2458-7575 (http://dergipark.gov.tr/bseufbd)

The basic problem of the grammar presented in List 1is the existence of left recursive rules. This type of rules creates ambiguity in generated mathematical expressions. Therefore, the grammar that has been used must be developed in a way that the rules do not have this property. In the production of the terms discussed in this article, there is no need to observe the priority of the operator. Because the main purpose is the production of the expression, the evaluation is not intended. In addition, the rules written for a non-terminal must be numbered in order to use at the time of execution of the algorithm based on these numbers. Accordingly, the desired grammar is developed in **List 2**.

List 2. Modified grammar for mathematical expressions

```
G = {Σ, T, V, P, S}
V = {E, C, Num, Digit, Var}⊆ Σ
T = {x, Sin, Cos, Tan, Cot, Log, Exp, Sqrt, +, -, *, /}⊆Σ
Σ = T ∪ V , S = { E }
Production :
    < E >::= < C > '+' < E >
    < E >::= < C > '-' < E >
    < E >::= < C > '*' < E >
    < E >::= < C > '/' < E >
    < E >::= 'Sin (' < E > ')'<T>
    < E >::= 'Cos (' < E > ')'<T>
    < E >::= 'Tan (' < E > ')'<T>
    < E >::= 'Cot (' < E > ')'<T>
    < E >::= 'Log (' < E > ')'<T>
    < E >::= 'Sqrt (' < E > ')'<T>
    < E >::= 'Abs (' < E > ')'<T>
    <T>::= '+'< E > | '-'< E > | '*'< E > | '/'< E > | < E >
    < C > ::= < Var > | < Num >< Var > | < Num >
    < Var >::='x'
    < Num >::= '-' ? < Digit > + ('. ' < Digit > +)?
< Digit >::=[' 0'-'9']
```

3) Generating Mathematical Expressions: Based on the proposed method, it is necessary to generate mathematical expressions using the grammar. Generating phrases should be such that the numbered rules are chosen based on the message data. It is also possible to include a part of the data in numbers that are assumed as a coefficient in the mathematical expressions. The methodology for generating mathematical expressions in accordance with the proposed methodology is presented below.

Generating AST via Grammar

The binary tree is a suitable data structure that supports the inclusion of algebraic operators and single parameter functions. It provides a simple way to represent operator precedence. Besides, it is easy to convert the tree structure to other data formats for the requirements of document formatters. Another advantage is about the development of formal grammars because each node of the tree suggests a recursive invocation among parent and child ones constructed by a grammar rule.

Figure 2 shows a block diagram of mathematical components required to create a mathematical expression using trees.

BŞEÜ Fen Bilimleri Dergisi
7(2), 739-754, 2020

BSEU Journal of Science
DOI: 10.35193/bseufbd.634578

e-ISSN: 2458-7575 (http://dergipark.gov.tr/bseufbd)

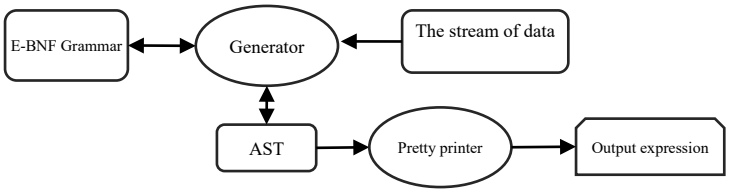

Figure 2. The methodological components for tree-based generation of expressions

In Figure 2, The "Generator" first places the beginning rule of a grammar into an AST and then recursively expands the non-terminals in that rule replacing them with the other rules according to input data.

Note that the rules presented in List do not consider the precedence of operators because it is not matter for the proposed method. List 3 displays the main algorithm of the methodology performed by the Generator component of Figure 2.

List 3. The expression-generating algorithm

AST ← Start symbol E
While exists any data in *input stream*
  N ← Get a data from stream
expr ← Select N^{th} rule
AST ← Replace *E* with *expr* in *AST*
In-order traverse the AST and print mathematical expression

The mathematical expressions generated using the method presented in List 3 are distributed uniformly and based on input data. Mathematical expressions that are commonly used in resources have different distributions. For example, there are usually more operators than the Cot function in the expressions. Stochastic Grammars [20] can be used to control the status of the invoked rules. Initializing an invocation probability for each rule in the algorithm, controlled through the expression steps, we can create mathematical expressions that are more realistic. In List 4, the grammar of List 2is represented by hypothetical possibilities.

List 4. A stochastic grammar with probabilities placed next to each production rule

1. < E >::= < C > '+' < E > (.17)
2. < E >::= < C > '-' < E > (.17)
3. < E >::= < C > '*' < E > (.12)
4. < E >::= < C > '/' < E > (.12)
5. < E >::= 'Sin (' < E > ')'< T > (.06)
6. < E >::= 'Cos (' < E > ')' < T > (.06)
7. < E >::= 'Tan (' < E > ')' < T > (.05)
8. < E >::= 'Cot (' < E > ')' < T > (.04)
9. < E >::= 'Log (' < E > ')' < T > (.05)
10. < E >::= 'Sqrt (' < E > ')' < T > (.10)
11. < E >::= 'Abs (' < E > ')' < T > (.06)
1. < T > ::= '+' < E > (.2)
2. < T > ::= '-' < E > (.2)
3. < T > ::= '*' < E > (.2)
4. < T > ::= '/' < E > (.2)
5. < T > ::= < E >(.2)

< C > ::= < Var > | < Num >< Var >
< Var >::='x'
< Num >::= '-' ? < Digit > + ('. ' < Digit > +)?
< Digit >::=[' 0'-'9']

BŞEÜ Fen Bilimleri Dergisi
7(2), 739-754, 2020

BSEU Journal of Science
DOI: 10.35193/bseufbd.634578

e-ISSN: 2458-7575 (http://dergipark.gov.tr/bseufbd)

Wayner has proposed a method for using probabilistic grammars using Huffman coding [12]. We can also act in the same way in our proposed method. For this purpose, we can consider the probability of each rule as the frequency of that rule and obtain a unique code by using the Huffman tree for each rule. List 5 shows the grammar of List 4 using the Huffman algorithm.

List 5. A stochastic grammar with Huffman code placed next to each production rule

```
1.  < E >::= < C > '+' < E >    (000)
2.  < E >::= < C > '-' < E >    (001)
3.  < E >::= < C > '*' < E >    (010)
4.  < E >::= < C > '/' < E >    (011)
5.  < E >::= 'Sin (' < E > ')'  (1000)
6.  < E >::= 'Cos (' < E > ')'  (1001)
7.  < E >::= 'Tan (' < E > ')'  (1010)
8.  < E >::= 'Cot (' < E > ')'  (1100)
9.  < E >::= 'Log (' < E > ')'  (1101)
10. < E >::= 'Sqrt (' < E > ')' (111)
11. < E >::= 'Abs (' < E > ')'  (1011)
1.  < E' > ::= '+' < E >        (00)
2.  < E' > ::= '-' < E >        (010)
3.  < E' > ::= '*' < E >        (011)
4.  < E' > ::= '/' < E >        (10)
5.  < E' > ::= < E >            (11)
    < C > ::= < Var > | < Num >< Var > | < Num >
    < Var > ::= 'x'
    < Num > ::= '-' ? < Digit > + ('. ' < Digit > +)?
    < Digit > ::= ['0' – '9']
```

Expressions generated from the grammar of Table 5 will have a more realistic distribution. Of course, the grammatical probabilities must be determined depending on the covers which the mathematical expressions will be placed in. For this purpose, it is easy to calculate the probabilities by examining the actual mathematical expressions and determining the number of repetitions of each of the rules.

Representation of Mathematical Expressions

Similar to programming languages, there are various mathematical languages to describe mathematical expressions. Three most well-known mathematical languages are discussed below: MPL, MathML, and LaTex. If the result of the syntax tree is a mathematical expression, e.g. derivation, then we will need to display it properly. AST has a tree form and therefore, it has to be converted into one of human-readable, LaTex, or MathML form. Traversing AST in infix technique will allow us to get the proper terms to construct the desired output. Figure 3 shows the AST of "(3cos (x + 1)) / 2".

BŞEÜ Fen Bilimleri Dergisi
7(2), 739-754, 2020

BSEU Journal of Science
DOI: 10.35193/bseufbd.634578

e-ISSN: 2458-7575 (http://dergipark.gov.tr/bseufbd)

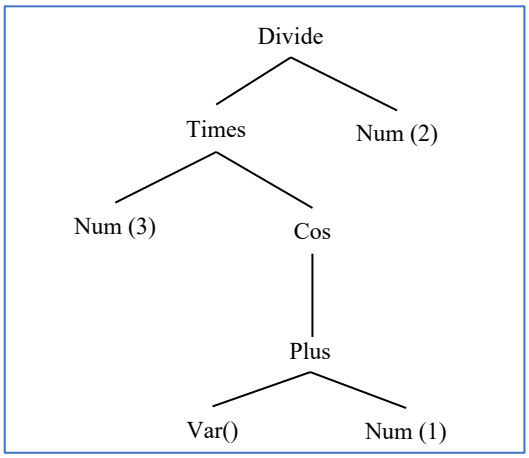

Figure 3. An example of a mathematical expression in AST

The result of infix traverse or simply syntax tree of Figure 3 is:

Divide (*Times*(*Num*(3), *Cos*(*Plus*(*Var*() , *Num*(1)))) , *Num*(2))

By applying output commands to each *class* in traversing code, required output format can be generated.

4) Parsing and interpreting of Mathematical Expression: For automatically generating parsers, there are many different generator tools that can create source codes in various languages. Typically examples of these tools are YACC [16]and bison [21], for imperative languages, ml-yacc[22], and happy [23] for functional languages, JavaCup [24], and JavaCC [18] for object-oriented languages. They require a special type of grammar as input, and generate either LL(k) or LR (k) parsers. Therefore, after choosing a parser generator, a proper grammar must be described according to the specifications of that generator.

JavaCC is a java-based parser generator that generates a top-down parser. Top-down parsers or recursive decent parsers allow the use of more general grammars. The only limitation of these parsers is that left recursion is not allowed because this may lead to infinite recursion. Top-down parsers have a structure identical to the grammar specification and are thus easier to debug. Embedding user code to generated abstract syntax tree in a top-down parser is simple due to the easiest of passing arguments and values across the nodes of the parse tree.

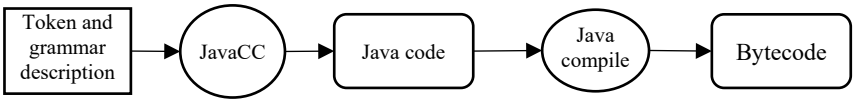

Figure 4. Parser generation with JavaCC

Converting Math Expression into AST

Converting a string into another model or data structure requires a converting process called translation. In an input string representing a mathematical expression, each operator has a rule that indicates the number of operands and the precedence of that operator to others. Such a translator which operates on mathematical expressions should keep these rules in a converted model, too.

The string form of mathematical expressions requires to be converted into mathematical tokens by a lexical analyzer. An example is shown below to demonstrate the input string and its token stream.

Table 1. Input string and its token stream

Input string	Token Stream
$3x^2+2(x-12)$	Num(3) Var Power Num(2) Plus Num(2) LPR Var Minus Num(12) RPR

In Table 1 the terms used to describe the token stream are for presentation purposes.

The token stream should be handled by mathematical rules which check if for example parentheses are in pair, and each operator has enough operand. These rules will be applied by a mathematical grammar.

The token stream will be analyzed to see whether it satisfies all rules defined by a mathematical grammar. According to the need, the syntax analyzer can report true or false for correct structure of the input stream, or generate a parse tree as an input to the next steps.

The parse tree is generated by a unit called parser. The leaves in the parse tree are terminals, and other nodes are non-terminals. However, it is appropriate to use another kind of tree called syntax tree. The main difference is that leaves are operands and nodes are operators. Besides, it offers a simple and efficient tree to work with mathematical expressions. The syntax tree is also called Abstract Syntax Tree or shortly AST. Figure 5 shows the steps to generate AST from an input string as a mathematical expression.

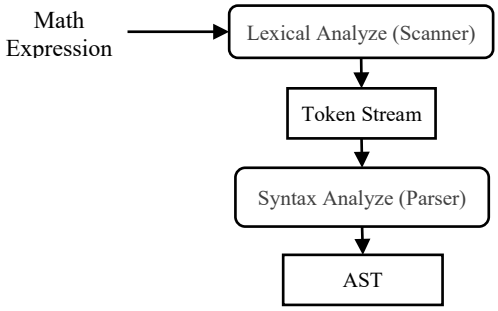

Figure 5. The steps to generate AST

Type inconsistency does not occur in mathematical expressions. Therefore, we will not use a semantic analyzer. Compiler-compiler tools can be used for convenience. Figure 6 demonstrates the steps required to produce the AST for our mathematical grammar using compiler-compiler tools.

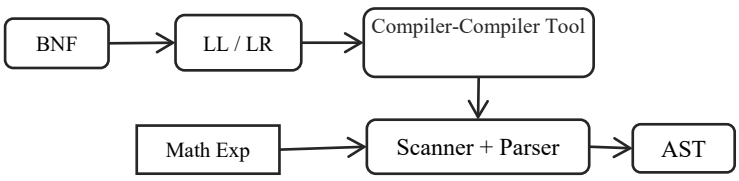

Figure 6. The steps required to produce the AST for the mathematical grammar

C. Embedding Message

In this section, the proposed method will be described. One of the parts of the steganography is to insert a message in a cover. Since the proposed method is a kind of cover generation, the message should be converted

into a cover of the type of mathematical expression. For this purpose, we will use the production method as presented in the previous section. Figure 7 shows the steps of the proposed methodology.

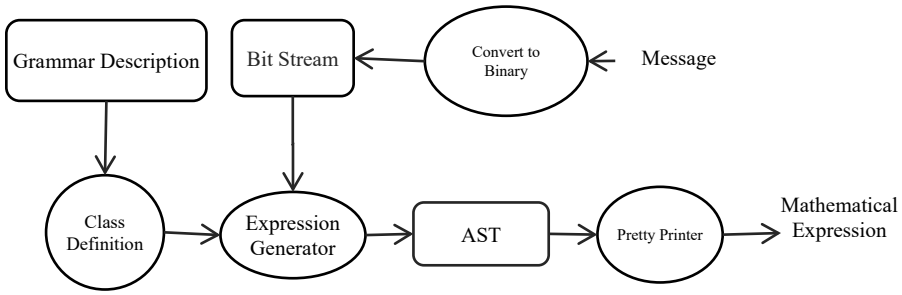

Figure 7. Components of the proposed methodology

As mentioned in the section on generating mathematical expressions, to generate mathematical expressions, you need an input through which the rules are selected for production. This entry is provided via the bit stream that is created from the message. Also, to increase the capacity of an insert message in a mathematical expression, instead of using the *Num ->Digit* rule, we can use a 3 or 4 bit number directly. Bits taken from the input are evaluated in sequence and selected by the Huffman codes assigned to each desired grammar rule. In cases where the rule has non-terminal *Num*, another part of the input is used as *Num* at the rule. This operation is repeated until all the input bits are completed. The Embedding algorithm is as List 6:

List 6. The embedding of a message

```
Convert message into binary as input bit stream
AST ← start symbol E
While exists any data in input stream
  expr ← Select a rule according to bits in stream and Huffman codes of rules
  if exist T rule then
     get 2 or 3 bit of stream, select a rule and replace it with T
  if exist Num in rule then
     get another 4 bit of stream and replace it with Num
AST  ← Apply expr in AST
Represent AST as mathematical expression
```

D. Extracting Message

A grammar-based methodology to parsing and extracting the message from mathematical expressions is proposed. A multi-party system is used to perform required operation for the proposed methodology. Input data is parsed using an extended grammar for a mathematical expression. The output of the parser is a tree data structure where it models the input data in a way that it can be used for data extracting. Each node of this tree represents a symbol or part of the input data, and its connection with other symbols where symbols can be operators, numbers, etc. The steps of our methodology are shown in Figure 8.

BŞEÜ Fen Bilimleri Dergisi
7(2), 739-754, 2020

BSEU Journal of Science
DOI: 10.35193/bseufbd.634578

e-ISSN: 2458-7575 (http://dergipark.gov.tr/bseufbd)

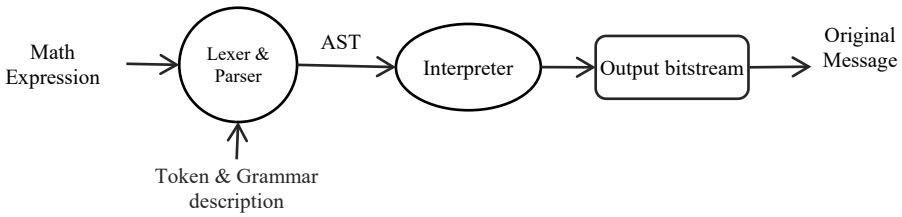

Figure 8. Phases and steps of the proposed methodology

Using these steps, one can implement a framework to work and extract the message from mathematical expressions. The operation of extraction of message bits can also be done at the time of AST production. In this case, at the time of parsing, when the parsing tree is created, in accordance with the rule used in the grammar, the corresponding Huffman code is extracted. There are several ways to create AST, one of which is the use of the compiler-compiler tool. In this paper, as an example, *JavaCC* is used to generate AST. For this purpose, in accordance with the proposed grammar, functions are developed.

Within each method, the corresponding operation is performed on expressions, and the result is returned, which is a string. List 7 shows some of the methods in *JavaCC*.

List 7. Some JavaCC functions for string-based message extraction

```
string parse() : { string a; }{
 a = expr(x) (<EOF> | <EOL>) { return a; }
}
string expr() : { string a, b; }{
   a = coef() <PLUS>  b = expr() { return "000"+binary(a)+b; }
 | a = coef() <MINUS> b = expr() { return "001"+binary(a)+b; }
 | a = coef() <MUL>   b = expr() { return "010"+binary(a)+b; }
 | a = coef() <DIV>   b = expr() { return "011"+binary(a)+b; }
 | <SIN><LPR> a=expr() <PPR><PLUS>  b = expr()
{ return "1000"+"00"+a+b;}
| <SIN><LPR> a=expr() <PPR><MINUS>  b = expr()
{ return "1000"+"010"+a+b;}
| <SIN><LPR> a=expr() <PPR><MUL>  b = expr()
{ return "1000"+"011"+a+b;}
 ...
}
string coef() : { Token t; }{
 t = <NUMBER><X> { return t.image; }
 | <X> { return ""; }
}
```

The parse function output in Table 8 is a string of extracted bits. By splitting the string into 8-bit strings and converting each one to the number representing the ASCII code of the message, the message can be obtained.

III. EVALUATION and RESULTS

In this paper, the text or any type of data is converted into a mathematical expression using grammatical rules, and somehow the data is hidden by the cover generation method. To evaluate the efficiency of the proposed system, we have implemented the proposed method using the Java programming language. Based on the random

BŞEÜ Fen Bilimleri Dergisi
7(2), 739-754, 2020

BSEU Journal of Science
DOI: 10.35193/bseufbd.634578

e-ISSN: 2458-7575 (http://dergipark.gov.tr/bseufbd)

structure presented in the proposed schema, a different mathematical expression is generated to conceal a specified text in every execution time. **Table 2** shows the production samples for some of data in different performances.

Table 2. Some of sample texts and their production expressions

Secret Message	Generated Expression
steganography	$\tan(\cot(\left\lvert x+\sqrt{\cos\frac{x}{37}-x}+x\right\rvert+13x-x)+16x-x)$
	$\tan(\cot(\lvert x+36\rvert+\sqrt{\cos\frac{5x}{26}-x+22})+\frac{x}{21})$
Math steganography	$\left\lvert\sqrt{\frac{15x}{x}}*\tan\frac{x}{17}+x*19\right\rvert-\frac{\cot(4x-\sqrt{\sqrt{16x*x}}*(7x-(x-x)))}{\frac{\log(\cos(\cot\frac{3x}{\frac{x}{x}}-\cot\frac{x}{33}))}{\cot\frac{\cos x}{x}}}+x$
	$\left\lvert\sqrt{\frac{15x}{\tan\frac{7x}{25}}+32}*\cos(7x*x)+x\right\rvert-\log(\sqrt{\frac{\lvert 17x-x\rvert}{15x-\tan(\sqrt{x*x}+x)*x}})+\frac{3x}{\frac{x}{x}}$
The weather is very good today	$\tan(13x+\log(\cot(\tan\sin(\log\frac{\sqrt{\log_{\frac{x}{x}}\frac{x}{x}}}{31}+x)-\log(x*x)*\frac{4x}{27})))$
	$x-\left\lvert\frac{x}{\sqrt{\sqrt{10x*x}-\frac{\sqrt{\cos(\lvert x-21\rvert*x)}}{26}}}+\sqrt{\cot\frac{\cot\sqrt{\lvert x+38\rvert*x}}{x}+x-(12x*33)}\right\rvert$

The manufacturing samples given in Table 2 show that the proposed method is suitable for concealing a short text. If the text lengths are acceptable, the generated mathematical expressions look very natural. To test the capacity of the proposed method, we have experimented some of text instances in the implemented system and the results are shown in Table 3.

Table 3. Capacity analysis for some of sample texts and their production expressions

Secret Message Size	Generated expression size (average)	Capacity Ratio
13	112.3	11.58%
18	133.3	13.50%
30	78.2	38.63%
60	147.0	40.82%
90	162.2	55.49%
120	208.4	57.58%
Average (55.17)	140.233	39.342%

For each text sample, 10 executions were performed and their average capacity was given in Table 3. According to this table and compared with other methods of steganography, it is clear that the proposed capacity is significantly more than other methods. Table 4 compares the proposed method with some of the other methods.

Table 4. Comparing Capacity of the proposed method with some of other methods

Method	Stego Type	Capacity Ratio
Kashida Technique [25]	Text	1.22%
Diacritics Technique [26]	Text	3.27%
Ref [27]	Image	≈0.8bpp = 10%
Ref [28]	Image	≈1bpp = 12.5%
Proposed Method	Mathematical	39.34%

Despite the high ability offered by the proposed method in capacity, the main weakness of this approach is that it is only suitable for short text lengths. For longer-length text, it can be included in more mathematical expressions. One of the applications of this method is to hide private keys in cryptographic methods which must be securely transmitted short text as a key.

IV. DISCUSSION

This paper proposes a method for hiding text in mathematical expressions. In terms of the special structures of mathematical expressions, it is possible to produce mathematical expressions that are not doubtful, with a returnable methodology for the message. For this purpose, a suitable grammar is designed according to the proposed method. We transform the desired grammar into a probabilistic grammar, which can determine the probability of each rule depending on the type of the given mathematical expressions. In this way, we can examine a number of actual mathematical expressions with respect to the text.

To make the generated mathematical expressions more natural, we can define different classes. At production time, we can choose one of these classes randomly or under certain conditions. This issue is presented in the algorithm presented in the methodology in Figure 8. Table 5 shows a list of some supported forms that an algebraic expression can have.

Table 5. Various forms of covered expressions

Cover form	***Cover form***
exp	$f(x) = exp$
$f(exp) = exp$	$exp = exp$
$f(x) = exp = exp$	$f(exp) = exp = exp$
$exp\ ,\ exp$	$f(x) = exp\ ,\ exp$
$f(x)=exp,\ g(x)=exp$	$f(exp)=exp,\ g(x)=exp$
$f(exp)=exp,\ g(exp)=exp$	$exp=exp, exp=exp$
$f(x)=exp=exp,\ g(x)=exp=exp$	

By choosing some of these classes, the generated mathematical expressions look more natural, and the capacity of the inserted message can be increased. Other cases may also be considered for realizing mathematical expressions. For example, the generated expressions can also be converted to the forms shown in Table 6.

Table 6. Some forms of generated expressions

Expression form	Expression form
$\int exp\, dx$	$\lim_{x\to 0} exp$
$\iint exp\, dxdy$	$\oint \exp dx$

The proposed method, as a cover generation method, transforms the message into a mathematical expression. This transformation, different from cryptography, can be regarded as a special kind of steganography named cover generation method, and can be proposed as a new method of steganography. Because the previous steganography methods do not use mathematical expressions as a cover, no steganalysis instances that examine the properties of mathematical expressions are developed. This can be considered as the strength of the proposed method because it can safely send messages through this type of data.

REFERENCES

[1] Gupta, B.B., D. Agrawal, and S. Yamaguchi. Handbook of Research on Modern Cryptographic Solutions for Computer and Cyber Security. New York: IGI Global, 2016.

[2] Bhattacharyya, S., I. Banerjee, and G. Sanyal. "A novel approach of secure text based steganography model using word mapping method (WMM)." International Journal of Computer and Information Engineering, 2010: Vol. 4, No. 2, pp.96–103.

[3] Lockwood, Robert, and Curran Kevin. "Text based steganography." International Journal of Information Privacy, Security and Integrity 3-2 (2017): 134-153.

[4] Li, J., C. YU, B.B. Gupta, and X. Ren. "Color image watermarking scheme based on quaternion Hadamard transform and Schur decomposition." Multimedia Tools and Applications, 1027: 1–17.

[5] Balaji, R., and Naveen Garewal. "Secure data transmission using video Steganography."IEEE International Conference on Electro/Information Technology. 2011.

[6] Nikam, G., A. Gupta, V. Kalal, and P. Waghmare. "A Survey of Video Steganography Techniques." Journal of Network Communications and Emerging Technologies (JNCET), 2017: 33-35.

[7] Cuttnel, J.D., and K.W. Johnson. Physics, 4th ed., Wiley, New York. 4th. New York: Wiley, 1998.

[8] Mishra, Shilpi, Yadav Virendra Kumar, Trivedi Munesh Chand, and ShrimaliTarun. "Audio Steganography Techniques: A Survey." Advances in Computer and Computational Sciences. Springer, 2018: 581-589.

[9] Johnson, Neil F., and JajodiaSushil. "Steganalysis of images created using current steganography software." International Workshop on Information Hiding. Berlin, 1998.

[10] Provos, Niels, and Honeyman Peter. "Hide and seek: An introduction to steganography." 2003. 32-44.

[11] Czaplewski, Bartosz. "Current trends in the field of steganalysis and guidelines for constructions of new steganalysis schemes." PrzeglądTelekomunikacyjny+ WiadomościTelekomunikacyjne, 2017: 1121-1125.

[12] Wayner, P. "Mimic functions." Cryptologia 16, no. 3 (Jul 1992): 193-214.

[13] Chang, Ching-Yun, and Clark Stephen. "Linguistic Steganography Using Automatically Generated Paraphrases." In Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics. 2010. 591-599.

[14] Sampat, Vivek, S. Karmokar, J. Madia, K. Dave, and P. Toprani. "Audio Steganography using Dynamic Cover Generation." IJCA Proceedings on National Conference on Advancement of Technologies. 2012. 26-30.

[15] Seifedine, Kadry, and Sara Nasr. "New Generating Technique for Image Steganography." Lecture Notes on Software Engineering 1, no. 2 (2013): 190-193.

[16] Johnson, Stephen C. Yacc: Yet another compiler-compiler. Vol. 32. Murray Hill, NJ: Bell Laboratories, 1975.

[17] Levine, John R., Mason Tony, and Brown Doug .Lex&yacc. O'Reilly Media, Inc., 1992.

[18] Kodaganallur, Viswanathan. "Incorporating language processing into java applications: A JavaCC tutorial." IEEE software 21, no. 4 (2004): 70-77.

[19] Ryan, Conor, O'Neill Michael, and Collins J. J. "Grammatical evolution: solving trigonometric identities." proceedings of Mendel, 1998.

[20] Newmeyer, Frederick J. "Grammar is grammar and usage is usage." Language 79, no. 4 (2003): 682-707.

[21] Donnelly, Charles, and Stallman Richard. "Bison. The YACC-compatible parser generator." 2000.

[22] Earley, Jay. "An efficient context-free parsing algorithm." Communications of the ACM, 1970: 94-102.

[23] Marlow, Simon, and Gill Andy. Happy: The parser generator for haskell. 2004. https://www.haskell.org/happy/.

[24] Hudson, S. E. JAVACUP: LALR parser generator for Java. User's manual, GVU Center, Georgia Institute of Technology, 1996.

[25] Adnan Gutub, LahouariGhouti, Alaaeldin Amin, TalalAlkharobi, andMohammad K. Ibrahim, "Utilizing Extension Character 'Kashida' With PointedLetters For Arabic Text Digital Watermarking", International Conference on Securityand Cryptography - SECRYPT, Barcelona, Spain, July 28 - 31, 2007.

[26] Aabed, M. A., Awaideh, S. M., Elshafei, A. R. M., &Gutub, A. A. (2007, November). Arabic diacritics based steganography. In Signal Processing and Communications, 2007. ICSPC 2007. IEEE International Conference on (pp. 756-759). IEEE.

[27] K. Qazanfari, R. Safabakhsh, A new steganography method which preserves histogram: Generalization of LSB++, Information Sciences, 277 (2014) 90-101.

[28] K. Muhammad, J. Ahmad, N.U. Rehman, Z. Jan, M. Sajjad, CISSKA-LSB: color image steganography using stegokeydirected adaptive LSB substitution method, Multimedia Tools and Applications, (2016) 1-30