Journal of Discrete Mathematical Sciences & Cryptography
ISSN 0972-0529 (Print), ISSN 2169-0065 (Online)
Vol. 29 (2026), No. 3, pp. 1219–1238
DOI : 10.47974/JDMSC-2195

Method for designing encoding algorithms based on AND/OR trees

Yuriy Shablya *
Vadim Polyuga †
Laboratory of Algorithms and Technologies for Discrete Structures Research
Tomsk State University of Control Systems and Radioelectronics
Tomsk 634050
Russia

Abstract

Information security is an important task in the digital age, and one of the ways to protect data is to use cryptographic methods. This paper discusses the problem of creating new combinatorial generation algorithms that can later be used as a mathematical basis for solving specific problems in the field of cryptography. For example, as a result of applying a ranking algorithm to a text over some alphabet, it will be encoded with the appropriate rank value. If some of the information that was used in this encoding process is declared as a secret, then the reverse decoding will become a difficult task. According to this idea, we proposed a method for designing encoding and decoding functions by using combinatorial generation algorithms that is based on applying AND/OR tree structures. Several examples of implementing the steps of the proposed method are also presented.

Subject Classification: *68P25, 05C05.*

Keywords: *Combinatorial set, Combinatorial generation algorithm, AND/OR tree, Ranking, Unranking, Encoding, Decoding.*

1. Introduction

The field of cryptography plays an important role due to the increasing demand for secure communication and data protection in the digital age [6,20,25]. Cryptographic encryption algorithms allow us to encode the original text (plaintext) so that it can only be recovered using some secret encryption key. Nowadays, there are simple implementations of cryptographic algorithms that find their application in situations of

* *E-mail:* **syv@fb.tusur.ru (Corresponding Author)**
† *E-mail:* **vadimiuspolyuga@gmail.com**

© Taru Publications

limited resources [4], as well as more complex ones that form entire cryptographic systems. The design of new cryptographic algorithms requires a deep understanding of different mathematical concepts that provide powerful tools for constructing algorithms with strong security properties. Cryptanalysis is also an important science because it is necessary to understand the cryptographic strength of the encryption algorithms used. For example, Mumtaz and Ping [16] overviewed various attacks on the RSA cryptosystem, which is now the standard of asymmetric encryption. Kant et al. [10] study the problem of identification of Indian languages from the plain and ciphered bit stream using different classifiers. Thus, due to the growth of computing power and the development of cryptanalysis methods, obtaining new mathematical foundations for cryptographic algorithms is a relevant and important task.

If we consider a text to be encrypted as a finite sequence of symbols in a finite alphabet, then the set of such words forms a combinatorial set for which various combinatorial algorithms are applicable. Combinatorial methods are widely used in designing and improving algorithms for encrypting and encoding information. For example, Saracevic, Adamovic, and Bisevac [19] presented a generator of pseudo-random numbers for obtaining cryptographic keys based on the Catalan numbers. They also presented a text encryption algorithm based on the application of the corresponding lattice paths. Benhamouda et al. [3] applied the methods of analytic combinatorics to Coppersmith methods aimed at finding small integer roots of polynomial equations. Wei et al. [24] proposed an improved version of the Blum-Kalai-Wasserman algorithm, which is a combinatorial algorithm and is actively used in quantum cryptography. Han et al. [7] considered different combinatorial designs and obtained optimal locally repairable codes with multiple repair sets. In [14] the authors applied combinatorial generation in steganography to find the best option for embedding information into the phase spectrum of the discrete Fourier transform. Sugimoto et al. [23] studied the cryptographic problem of two sheriffs and proposed an improved version of the combinatorial solution to this problem. In [9,11] the authors used combinatorial testing to detect Trojans. Moreover, combinatorial generation algorithms together with formal grammars can also be applied in the field of cryptography for format-preserving data encryption [2,15]. Thus, based on existing studies, we can conclude about the relevance of using combinatorial methods in solving problems related to cryptography. At the same time, there are a large number of tasks that have not been properly studied. This paper

discusses the problem of developing cryptographic algorithms based on applying combinatorial generation methods.

The main task of combinatorial generation is the generation of combinatorial objects that belong to a given set [12,18]. The generation of combinatorial objects can be realized in a random order or in accordance with a specific object's ordering. For the ordered case, it is necessary to specify the order of combinatorial objects, for example, by developing an appropriate ranking algorithm. Then the generation of a combinatorial object having a certain rank is performed using an unranking algorithm. Different general approaches are available for creating new combinatorial generation algorithms: the backtracking method [13], the ECO-method [1], the symbolic method [5], the method based on AND/OR tree structures [21,22], the method based on permutation languages [8], etc. This article is devoted to the application of the method that uses AND/OR trees to represent combinatorial sets.

2. Combinatorial generation based on AND/OR trees

The main idea of the method for developing combinatorial generation algorithms based on AND/OR trees is as follows: for a given combinatorial set, we build the corresponding AND/OR tree structure, where the number of AND/OR tree variants equals the number of combinatorial objects. An AND/OR tree is a tree structure where each internal node is an AND-node or an OR-node. An AND-node corresponds to the multiplication operation and is used to combine the child nodes. An OR-node corresponds to the addition operation and is used to select one of the child nodes. The graphical representation of an AND-node is distinguished by the presence of an additional arc connecting its edges to the child nodes. To obtain a variant of an AND/OR tree, it is necessary to leave only one child node for each OR-node. Figure 1 demonstrates an example of an AND/OR tree with its variants.

In order to construct an AND/OR tree, we need to have a formula for calculating the cardinality of a given combinatorial set and this formula must satisfy the following requirements: it is allowed to use addition and multiplication operations (represented by OR-nodes and AND-nodes) and positive integers (represented by an OR-node with the corresponding number of child leaf nodes). In addition, the composition of AND/OR trees (including recursive composition) is also allowed, i.e. when a subtree of some node represents the structure of another AND/OR tree. The

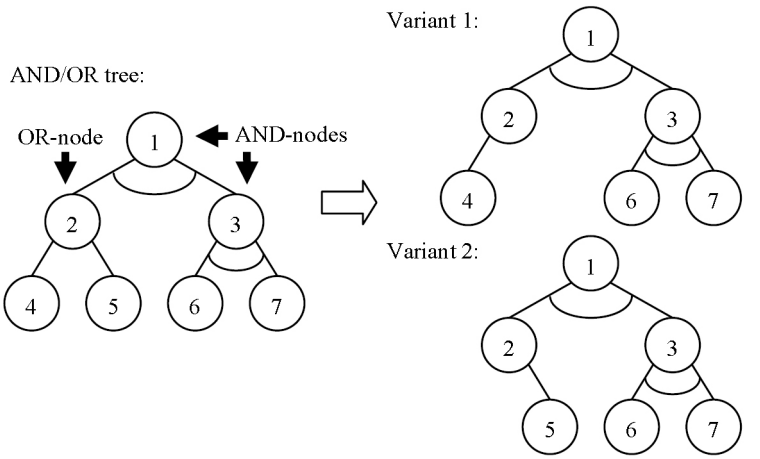


Figure 1

Example of AND/OR tree structure with its two variants

composition of AND/OR trees is represented by a node with a triangle, and this node is the root of another AND/OR tree.

If we have constructed an AND/OR tree structure that corresponds to a given combinatorial set, then it is necessary to define a bijection between this combinatorial set and the set of AND/OR tree variants. Then, for the set of AND/OR tree variants, combinatorial generation algorithms for their ranking and unranking can be constructed. At the same time, these combinatorial generation algorithms can be used for the corresponding combinatorial set due to the bijection between these sets.

The algorithm for ranking an AND/OR tree variant is based on the function $l(z)$ that calculates the value of the subtree rank for a given node z. The rules for calculating the value of $l(z)$ depend on the type of node z (an AND-node, an OR-node or a leaf node). A detailed description of the rules for calculating the value of $l(z)$ can be found in [21]. The algorithm for unranking an AND/OR tree variant is based on inverse transformations.

Note that for the same combinatorial set it is possible to construct different corresponding AND/OR tree structures by using equivalent expressions for the cardinality of the set.

3. Method for designing encoding algorithms

Let consider a finite alphabet A, which is a finite set of n symbols a_i ($a_i \in A$, $i = \overline{1,n},$, i.e. $A = \{a_1, a_2, \ldots, a_n\}$ and n is the cardinality of A (or

$|A|=n$). Using the symbols of this alphabet A, it is possible to compose some text t of length m $(m \geq 0)$ by concatenating these symbols, i.e. $t = t_1 t_2 \cdots t_m$ where $t_j \in A,\ j = \overline{1,m}$. If $m = 0$, we have the empty text.

Let $T_{n,m}$ denote the set of all texts $t = t_1 t_2 \cdots t_m$ of length m over a given alphabet A with $|A|=n$. Since each t_j $(j = \overline{1,m})$ corresponds to one of the symbols a_i $(i = \overline{1,n})$ and repetitions are allowed, the number of all possible texts is defined by the following formula:

$$|T_{n,m}| = n^m. \tag{1}$$

Thus, for fixed parameters n and m, we have a finite set of objects and rules for constructing such objects. Therefore, combinatorial generation algorithms can be applied to this combinatorial set $T_{n,m}$. For example, as a result of applying a ranking algorithm to a text t, it will be encoded with the appropriate rank r. If some of the information that was used in this encoding process is declared as a secret, then the reverse decoding will become a difficult task. The following steps present the main idea of the proposed method for designing encoding algorithms by using combinatorial generation algorithms that are based on AND/OR trees:

Step 1: Selecting a formula for calculating the cardinality of the combinatorial set, that satisfies the requirements of the method based on AND/OR trees (it can be (1) or any other equivalent expression), and constructing the corresponding AND/OR tree structure D. The number of variants v of the AND/OR tree D will be equal to the number of texts t in the combinatorial set $T_{n,m}$.

Step 2: Defining the rules for bijection between the combinatorial set $T_{n,m}$ and the set of variants of the AND/OR tree D. In this case, it is important to fix the order of the symbols of the alphabet A. Let B denote these bijection rules and $V(D)$ denote the set of variants v of the AND/OR tree D, then $B : T_{n,m} \leftrightarrow V(D)$. These bijection rules B are implemented by applying the following two functions:

$$\texttt{TextToVariant} : T_{n,m} \to V(D), \quad v := \texttt{TextToVariant}(t, A, D, B);$$

$$\texttt{VariantToText} : V(D) \to T_{n,m}, \quad t := \texttt{VariantToText}(v, A, D, B).$$

Step 3: Developing combinatorial generation algorithms for ranking and unranking the variants v of the AND/OR tree D:

$$\texttt{RankVariant}: V(D) \to \mathbb{N}_0, \quad r := \texttt{RankVariant}(v, D);$$

$$\texttt{UnrankVariant}: \mathbb{N}_0 \to V(D), \quad v := \texttt{UnrankVariant}(r, D).$$

Step 4: Determining the information that will be a secret (an encryption key). For example, such secret information could be: the alphabet A, the order of symbols in the alphabet A, the length m of the text t, the AND/OR tree D, the bijection rules B, or a combination of such parameters.

Step 5: Constructing encoding and decoding algorithms. The ciphertext c will be the value of the rank r calculated for the AND/OR tree variant v associated with the plaintext t. In general, the constructed encoding and decoding algorithms will have the following form:

$$c := \texttt{Encode}(t, A, D, B) = \texttt{RankVariant}(\texttt{TextToVariant}(t, A, D, B), D);$$

$$t := \texttt{Decode}(c, A, D, B) = \texttt{VariantToText}(\texttt{UnrankVariant}(c, D), A, D, B).$$

Thus, performing the steps presented above allows us to obtain encoding and decoding algorithms, which are based on the use of combinatorial generation algorithms. Moreover, if we use different formulas for calculating $|T_{n,m}|$, then different AND/OR tree structures will be obtained. This will lead to a variety of calculations in combinatorial generation algorithms, i.e. the result will be various encoding algorithms.

4. Examples of appropriate AND/OR tree structures and corresponding algorithms

Next, we consider several appropriate formulas for calculating $|T_{n,m}|$ and the corresponding AND/OR tree structures that can be used for designing the corresponding algorithms.

4.1 *AND/OR tree structure for* $|T_{n,m}|$ *based on (1)*

Let us represent Formula (1) in the following form that satisfies the requirements of the method based on AND/OR trees:

$$|T_{n,m}| = n^m = \prod_{j=1}^{m} n. \tag{2}$$

Figure 2 shows the corresponding AND/OR tree structure for (2).

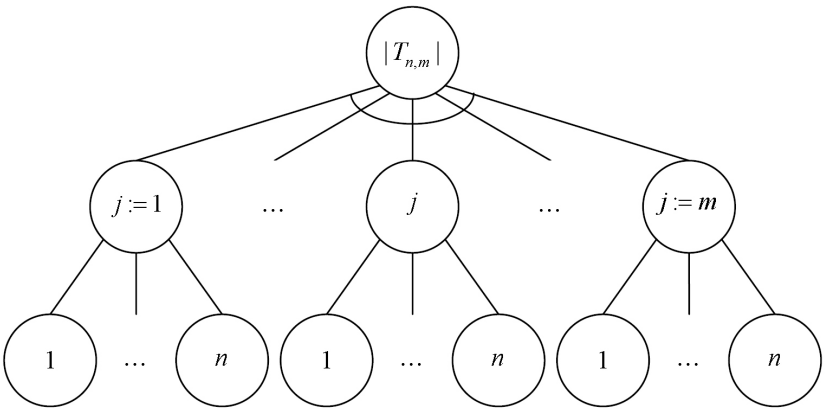


Figure 2

AND/OR tree structure for $|T_{n,m}|$ based on (2)

The rules for bijection between the combinatorial set $T_{n,m}$ and the set of variants of the constructed AND/OR tree are defined as follows:

- The subtree of an OR-node labeled j defines the value of t_j in the text $t = t_1 t_2 \cdots t_m$;
- Selecting a child node of an OR-node labeled j shows the symbol a_i of the ordered alphabet $A = (a_1, a_2, \ldots, a_n)$, which is the value of t_j.

For the constructed AND/OR tree structure, we also develop combinatorial generation algorithms for ranking and unranking its variants. In these algorithms, a variant v of the AND/OR tree is given by a pointer to its root node.

Algorithm 1: Ranking variant v of the AND/OR tree from Figure 2.

```
RankVariant_A(v, n, m)
begin
    if m = 0 then r := 0
    else
            (v_1,…,v_m) := Get pointers to child nodes of v
            k := Get the label of the selected child node of v_m
            r := k − 1
            for j := m − 1 to 1 do
                    k := Get the label of the selected child node of v_j
                    r := (k − 1) + n · r
```

```
            end
        end
        return r
    end
```

Algorithm 2: Unranking variant v of the AND/OR tree from Figure 2.

```
UnrankVariant_A(r, n, m)
begin
    Create the root node and its pointer v
    if m > 0 then
            Create m child nodes (v_1,...,v_m) for the AND-node v
            for j := 1 to m do
                    k := r mod n
                    Create a child node labeled k+1 for the node v_j
                    r := ⌊r/n⌋
            end
    end
    return v
end
```

4.2 *AND/OR tree structure for* $|T_{n,m}|$ *based on recurrence for (1)*

Let us represent Formula (1) in the following recurrent form that satisfies the requirements of the method based on AND/OR trees:

$$|T_{n,m}| = n^m = n \cdot n^{m-1} = n \cdot |T_{n,m-1}|. \tag{3}$$

The obtained recurrence has the following initial condition: $|T_{n,m}| = 1$ for $m = 0$ since there is only one empty text.

Figure 3 shows the corresponding AND/OR tree structure for (3).

The rules for bijection between the combinatorial set $T_{n,m}$ and the set of variants of the constructed AND/OR tree are defined as follows:

- Selecting a child node of the OR-node labeled n shows the symbol a_i of the ordered alphabet $A = (a_1, a_2, \ldots, a_n)$, which is the value of t_m in the text $t = t_1 t_2 \cdots t_m$;
- The subtree of the node labeled $|T_{n,m-1}|$ defines the values of $t_1, \ldots, t_{m-1}$ in the text $t = t_1 t_2 \cdots t_m$;
- Reaching the node labeled $|T_{n,0}|$ corresponds to the empty text and indicates the recursive calls to stop.

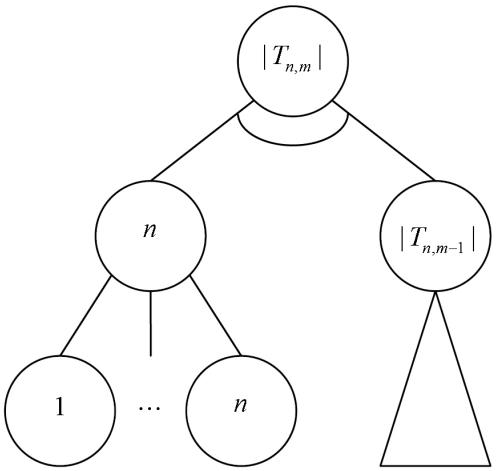


Figure 3

AND/OR tree structure for $|T_{n,m}|$ based on (3)

For the constructed AND/OR tree structure, we also develop combinatorial generation algorithms for ranking and unranking its variants. In these algorithms, a variant v of the AND/OR tree is given by a pointer to its root node. Note that the calculations in these algorithms are identical to the calculations in Algorithms 1-2 and differ only in the recursive organization of the loop and the reverse order viewing the text.

Algorithm 3: Ranking variant v of the AND/OR tree from Figure 3.

```
RankVariant_B(v, n, m)
begin
    if m = 0 then r := 0
    else
            (v1, v2) := Get pointers to child nodes of v
            k := Get the label of the selected child node of v1
            r := RankVariant_B(v2, n, m − 1)
            r := (k − 1) + n · r
    end
    return r
end
```

Algorithm 4: Unranking variant v of the AND/OR tree from Figure 3.

```
UnrankVariant_B(r, n, m)
begin
    Create the root node and its pointer v
    if m > 0 then
            Create 2 child nodes (v_1, v_2) for the AND-node v
            k := r mod n
            Create a child node labeled k+1 for the node v_1
            r := ⌊r/n⌋
            v_2 := UnrankVariant_B(r, n, m−1)
    end
    return v
end
```

4.3 *AND/OR tree structure for* $|T_{n,m}|$ *based on combinations*

Let us derive another formula for calculating the values of $|T_{n,m}|$ that also satisfies the requirements of the method based on AND/OR trees.

Theorem 4.1: *The number of all texts* $t = t_1 t_2 \cdots t_m$ *of length* m *over a given alphabet* A *with* $|A| = n$ *can be calculated using the following recurrence:*

$$|T_{n,m}| = \sum_{k=0}^{m} C_m^k \cdot |T_{n-1,m-k}|, \tag{4}$$

where $|T_{n,m}| = 1$ *for* $m = 0$ *or* $n = 1$ *and* C_m^k *is the number of k-combinations of the set of m elements.*

***Proof*:** Let k be the number of the symbol a_n in the text $t = t_1 t_2 \cdots t_m$. The minimum possible value of k is 0, i.e. when the text t does not contain the symbol a_n. The maximum possible value of k is m, i.e. when the text t consists only of a_n and $t = a_n a_n \cdots a_n$. Therefore, it is necessary to sum up the number of texts received for each possible value of k.

Next, for the fixed value of k, we need to determine k positions in the text $t = t_1 t_2 \cdots t_m$ for which $t_j = a_n$. The number of ways to do this corresponds to the number of k-combinations of the set $\{t_1, t_2, \ldots, t_m\}$, denoted by C_m^k and calculated as a binomial coefficient (A007318 in OEIS [17]). Then, each $t_j = a_n$ must be removed from the text $t = t_1 t_2 \cdots t_m$. The result will be the text of length $m-k$ that does not contain the symbol a_n. The number of such texts of length $m-k$ over the alphabet $A \setminus \{a_n\}$ is equal to $|T_{n-1,m-k}|$. Since for each of the C_m^k combinations there are

$|T_{n-1,m-k}|$ different ways, it is necessary to multiply C_m^k by $|T_{n-1,m-k}|$ to find the number of texts containing k of a_n.

The obtained recurrence has the following initial conditions:

- $|T_{n,m}|=1$ for $n=1$ since there is only one such text $t=t_1t_2\cdots t_m = a_1a_1\cdots a_1$;
- $|T_{n,m}|=1$ for $m=0$ since there is only one empty text.

Figure 4 shows the corresponding AND/OR tree structure for (4), where the subtree of the node labeled C_m^k also has an AND/OR tree structure.

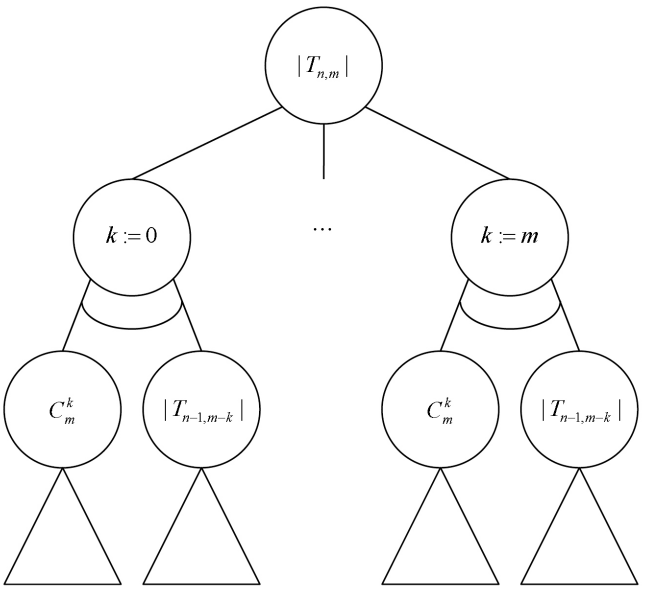


Figure 4

AND/OR tree structure for $|T_{n,m}|$ based on (4)

The rules for bijection between the combinatorial set $T_{n,m}$ and the set of variants of the constructed AND/OR tree are defined as follows:

- Selecting an AND-node labeled k shows the number of a_n in the text $t=t_1t_2\cdots t_m$;
- The subtree of the node labeled C_m^k defines the *k* positions in the text $t=t_1t_2\cdots t_m$ for which $t_j=a_n$;

- The subtree of the node labeled $|T_{n-1,m-k}|$ defines the text of length $m-k$ obtained after removing each $t_j = a_n$ from the text $t = t_1 t_2 \cdots t_m$ over the alphabet $A \setminus \{a_n\}$;
- Reaching the node labeled $|T_{1,m}|$ corresponds to the text $t = t_1 t_2 \cdots t_m = a_1 a_1 \cdots a_1$ and indicates the recursive calls to stop;
- Reaching the node labeled $|T_{n,0}|$ corresponds to the empty text and indicates the recursive calls to stop.

For the constructed AND/OR tree structure, we also develop combinatorial generation algorithms for ranking and unranking its variants. In these algorithms, a variant v of the AND/OR tree is given by a pointer to its root node.

Algorithm 5: Ranking variant v of the AND/OR tree from Figure 4.

```
RankVariant_C(v, n, m)
begin
    if m = 0 or n = 1 then r := 0
    else
        l := Get the label of the selected child node of v
        (v1, v2) := Get pointers to child nodes of the node labeled l
        r1 := RankVariant_Combination(v1, m, l)
        r2 := RankVariant_C(v2, n − 1, m − l)
        r := r1 + C_m^l · r2 + Σ_{k=0}^{l−1} C_m^k · (n − 1)^{m−k}
    end
    return r
end
```

Algorithm 6: Unranking variant v of the AND/OR tree from Figure 4.

```
UnrankVariant_C(r, n, m)
begin
    Create the root node and its pointer v
    if m > 0 then
        s := 0
        for k := 0 to m do
            if s + C_m^k · (n − 1)^{m−k} > r then
                l := k
                r := r − s
                break
```

```
                    end
                    s := s + C_m^k · (n − 1)^(m−k)
                end
                Create a child node labeled l for the node v
                Create 2 child nodes (v_1, v_2) for the node labeled l
                r_1 := r mod C_m^l
                r_2 := ⌊ r / C_m^l ⌋
                v_1 := UnrankVariant_Combination(r_1, m, l)
                v_2 := UnrankVariant_C(r_2, n − 1, m − l)
            end
            return v
        end
```

Note that Algorithms 5-6 require algorithms for ranking and unranking k-combinations of the set of m elements (for example, see [21]).

4.4 *AND/OR tree structure for* $|T_{n,m}|$ *based on set partitions*

Let us derive another formula for calculating the values of $|T_{n,m}|$ that also satisfies the requirements of the method based on AND/OR trees.

Theorem 4.2: *The number of all texts* $t = t_1 t_2 \cdots t_m$ *of length* m *over a given alphabet* A *with* $|A| = n$ *can be calculated using the following formula:*

$$|T_{n,m}| = \sum_{k=1}^{\min(n,m)} S_m^k \cdot C_n^k \cdot P_k, \tag{6}$$

where S_m^k *is the number of partitions of an m-element set into k parts,* C_n^k *is the number of k-combinations of an n-element set and* P_k *is the number of permutations of k elements.*

Proof: Let k be the number of different symbols a_i in the text $t = t_1 t_2 \cdots t_m$. The minimum possible value of k is 1, i.e. when the text t consists only of a certain a_i and $t = a_i a_i \cdots a_i$. The maximum possible value of k is $\min(n, m)$, i.e. when all t_j in the text $t = t_1 t_2 \cdots t_m$ are different $(m < n)$ or the text t contains all symbols of the alphabet A $(n < m)$.

Next, for the fixed value of k, we need to determine the partition of the text $t = t_1 t_2 \cdots t_m$ into k subsets, where each subset contains t_j with the same values. The number of ways to do this corresponds to the number of partitions of the set $\{t_1, t_2, \ldots, t_m\}$ into k parts, denoted by S_m^k and calculated

as the Stirling numbers of the second kind (A008277 in OEIS [17]). Then, for each subset, we need to determine the corresponding symbol from the alphabet A. The number of ways to select k symbols from the alphabet $A = \{a_1, a_2, \ldots, a_n\}$ corresponds to the number of k-combinations of the set $\{a_1, a_2, \ldots, a_n\}$, denoted by C_n^k and calculated as a binomial coefficient (A007318 in OEIS [17]). An arrangement of the selected k symbols between the k subsets is determined by their permutation. The number of permutations of k elements is denoted by P_k and calculated as a factorial (A000142 in OEIS [17]). Therefore, to find the number of texts containing k different symbols a_i, it is necessary to multiply S_m^k, C_n^k and P_k.

Figure 5 shows the corresponding AND/OR tree structure for (5), where the subtrees of the nodes labeled S_m^k, C_n^k and P_k also have an AND/OR tree structure.

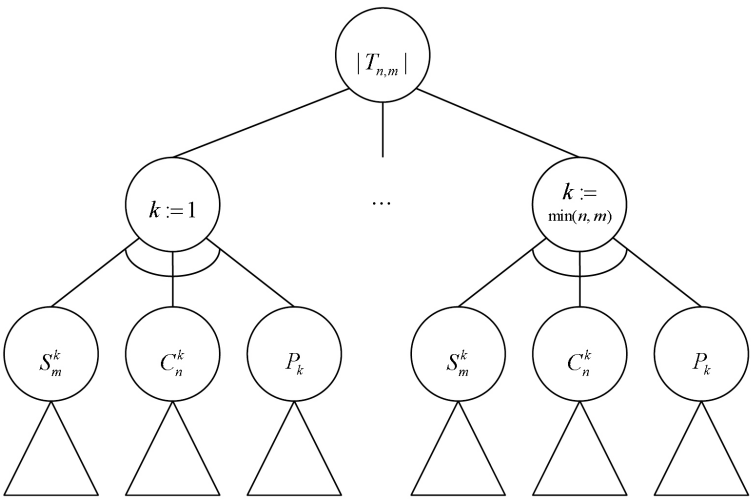


Figure 5

AND/OR tree structure for $|T_{n,m}|$ based on (5)

The rules for bijection between the combinatorial set $T_{n,m}$ and the set of variants of the constructed AND/OR tree are defined as follows:

- Selecting an AND-node labeled k shows the number of different symbols a_i in the text $t = t_1 t_2 \cdots t_m$;
- The subtree of the node labeled S_m^k defines the partition of the text $t = t_1 t_2 \cdots t_m$ into k subsets containing t_j with the same values;

- The subtree of the node labeled C_n^k defines the selection of k symbols from the alphabet $A = \{a_1, a_2, \ldots, a_n\}$;
- The subtree of the node labeled P_k defines the arrangement of the selected k symbols between the k subsets.

For the constructed AND/OR tree structure, we also develop combinatorial generation algorithms for ranking and unranking its variants. In these algorithms, a variant v of the AND/OR tree is given by a pointer to its root node.

Algorithm 7: Ranking variant v of the AND/OR tree from Figure 5.

RankVariant_D(v, n, m)
begin
  if $m = 0$ **then** $r := 0$
  else
    $l :=$ Get the label of the selected child node of v
    $(v_1, v_2, v_3) :=$ Get pointers to child nodes of the node labeled l
    $r_1 :=$ RankVariant_SetPartition$(v_1,\ m,\ l)$
    $r_2 :=$ RankVariant_Combination$(v_2,\ n,\ l)$
    $r_3 :=$ RankVariant_Permutation$(v_3,\ l)$
    $r := r_1 + S_m^l \cdot (r_2 + C_n^l \cdot r_3) + \sum_{k=1}^{l-1} S_m^k \cdot C_n^k \cdot P_k$
  end
  return r
end

Algorithm 8: Unranking variant v of the AND/OR tree from Figure 5.

UnrankVariant_D(r, n, m)
begin
  Create the root node and its pointer v
  if $m > 0$ **then**
    $s := 0$
    for $k := 1$ **to** m **do**
      if $s + S_m^k \cdot C_n^k \cdot P_k > r$ **then**
        $l := k$
        $r := r - s$
        break
      end
      $s := s + S_m^k \cdot C_n^k \cdot P_k$

```
            end
            Create a child node labeled l for the node v
            Create 3 child nodes (v_1, v_2, v_3) for the node labeled l
            r_1 := r mod S_m^l
            r := ⌊ r / S_m^l ⌋
            r_2 := r mod C_n^l
            r_3 := ⌊ r / C_n^l ⌋
            v_1 := UnrankVariant_SetPartition(r_1, m, l)
            v_2 := UnrankVariant_Combination(r_2, n, l)
            v_3 := UnrankVariant_Permutation(r_3, l)
        end
        return v
    end
```

Note that Algorithms 7-8 require algorithms for ranking and unranking partitions of an m-element set into k parts, k-combinations of the set of n elements and permutations of k elements (for example, see [21]).

5. Discussion

The results presented in Section 4 describe possible ways to implement Steps 1-3 of the proposed method for designing encoding and decoding algorithms. Figure 6 shows examples of encoding the text $t = abb$ over the ordered alphabet $A = (c, a, b)$ by AND/OR tree variants based on AND/OR tree structures from Figure 2, Figure 3 and Figure 4. Figure 6 also shows the results of ranking these AND/OR tree variants by applying Algorithm 1, Algorithm 3 and Algorithm 5. To decode the text according to a given rank by applying an unranking algorithm, we need to know information about the used AND/OR tree structure, as well as the used bijection rules.

Next, it is necessary to determine the information that will be a secret (an encryption key). For example, such secret information could be: the alphabet, the order of symbols in the alphabet, the length of the text, the AND/OR tree, the bijection rules, or a combination of such parameters. Constructing a new appropriate AND/OR tree structure is a difficult task, this paper presents 4 possible such structures. Therefore, using only information about the AND/OR tree and its bijection rules as an encryption

key is not a very good idea (small number of possible keys and large amount of stored data). However, if a new appropriate AND/OR tree structure is obtained, then the corresponding encryption algorithm will be cryptographically strong until this AND/OR tree becomes known to third parties.

Using the length of the text as an encryption key makes breaking the encryption algorithm much more difficult, but it requires working with integers of arbitrary length for texts of large length. If we fix the length of the encoded text, we obtain a block cipher.

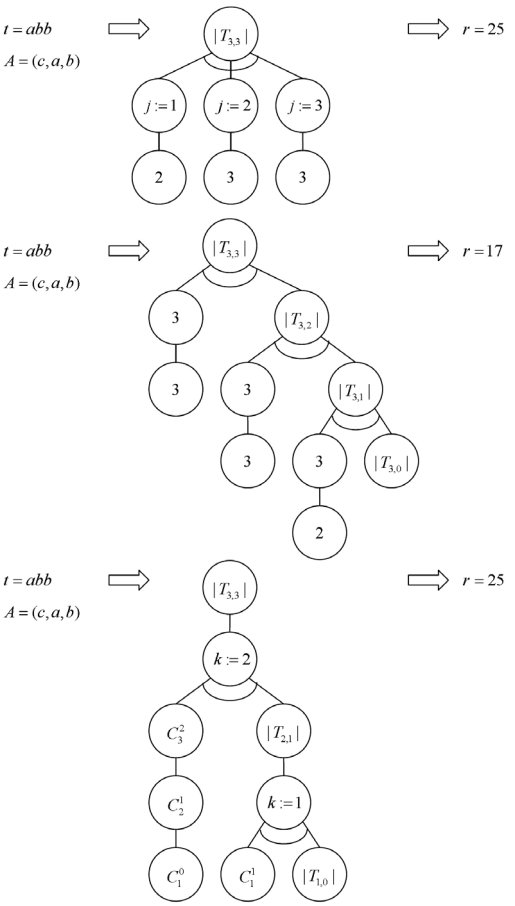


Figure 6

Examples of encoding the text $t = abb$ over the alphabet $A = (c, a, b)$ based on Algorithm 1, Algorithm 3 and Algorithm 5

In addition, for widespread use, it is desirable to fix the alphabet used. A universal option is to use the alphabet $A = \{0,1\}$, but in this case there are only 2 options for ordering this alphabet. To use the order of symbols in the alphabet as an encryption key, we need to have much more options. This can be done by using a sequence of zeros and ones as the alphabet symbol. For example, the alphabet $A = \{00,01,10,11\}$ has 4 symbols and $4! = 24$ options for ordering this alphabet. In general, if each symbol of the alphabet is a sequence of k zeros and ones, then the alphabet contains 2^k symbols and there are $(2^k)!$ options for ordering this alphabet.

Thus, the main result of this paper is the proposed method, which allows us to design new encoding and decoding algorithms applying combinatorial generation algorithms based on AND/OR trees. The obtained algorithms can later be used as a mathematical basis for solving specific problems in the field of cryptography. Further research can be aimed at creating a specific cryptographic algorithms based on the developed algorithms.

Acknowledgement

This research was funded by the Russian Science Foundation, grant number 22-71-10052.

References

[1] S. Bacchelli, E. Barcucci, E. Grazzini, and E. Pergola, "Exhaustive generation of combinatorial objects by ECO," *Acta Informatica*, vol. 40, pp. 585–602 (2004), doi: 10.1007/s00236-004-0139-x.

[2] M. Bellare, T. Ristenpart, P. Rogaway, and T. Stegers, "Format-preserving encryption," in *Lecture Notes in Computer Science*, vol. 5867, pp. 295–312 (2009), doi: 10.1007/978-3-642-05445-7_19.

[3] F. Benhamouda, C. Chevalier, A. Thillard, and D. Vergnaud, "Easing Coppersmith methods using analytic combinatorics: Applications to public-key cryptography with weak pseudorandomness," in *Lecture Notes in Computer Science*, vol. 9615, pp. 36–66 (2016), doi: 10.1007/978-3-662-49387-8_3.

[4] D. P. Bhatt, L. Raja, and S. Sharma, "Light-weighted cryptographic algorithms for energy efficient applications," *Journal of Discrete Mathematical Sciences and Cryptography*, vol. 23, no. 3, pp. 643–650 (2020), doi: 10.1080/09720529.2020.1729510.

[5] P. Flajolet, P. Zimmerman, and B. Cutsem, "A calculus for the random generation of combinatorial structures," *Theoretical Computer Science*, vol. 132, pp. 1–35 (1994), doi: 10.1016/0304-3975(94)90226-7.

[6] O. Goldreich, Foundations of Cryptography: Volume 1, Basic Tools. Cambridge, U.K.: Cambridge Univ. Press (2008).

[7] X. Han, G. Han, H. Cai, and L. Yin, "Locally repairable codes with multiple repair sets based on packings of block size 4," *Cryptography and Communications* (2023), doi: 10.1007/s12095-023-00681-z.

[8] E. Hartung, H. Hoang, T. Mutze, and A. Williams, "Combinatorial generation via permutation languages. I. Fundamentals," *Transactions of the American Mathematical Society,* vol. 375, pp. 2255–2291 (2022), doi: 10.1090/tran/8199.

[9] L. Kampel, P. Kitsos, and D. E. Simos, "Locating hardware Trojans using combinatorial testing for cryptographic circuits," *IEEE Access,* vol. 10, pp. 18787–18806 (2022), doi: 10.1109/ACCESS.2022.3151378.

[10] S. Kant, V. Sharma, N. Verma, and B. K. Dass, "Identification scheme for romanized Indian languages from their plain and ciphered bit stream," *Journal of Discrete Mathematical Sciences and Cryptography,* vol. 13, no. 3, pp. 329–345 (2010), doi: 10.1080/09720529.2010.10698298.

[11] P. Kitsos, D. E. Simos, J. Torres-Jimenez, and A. G. Voyiatzis, "Exciting FPGA cryptographic Trojans using combinatorial testing," in Proc. IEEE 26th Int. Symp. Software Reliability Engineering (ISSRE), pp. 69–76 (2015), doi: 10.1109/ISSRE.2015.7381800.

[12] D. E. Knuth, The Art of Computer Programming, Volume 4A: Combinatorial Algorithms, Part 1. Boston, MA, USA: Addison-Wesley Professional (2011).

[13] D. L. Kreher and D. R. Stinson, Combinatorial Algorithms: Generation, Enumeration, and Search. Boca Raton, FL, USA: CRC Press (1999).

[14] A. Melman, O. Evsutin, and Y. Shablya, "On the efficiency of combinatorial generation for adaptive image steganography," in Proc. 2021 Int. Conf. Engineering and Telecommunication (EnT) (2021), doi: 10.1109/EnT50460.2021.9681730.

[15] S. Miracle and S. Yilek, "Targeted invertible pseudorandom functions and deterministic format-transforming encryption," in *Lecture Notes in Computer Science*, vol. 13871, pp. 622–642 (2023), doi: 10.1007/978-3-031-30872-7_24.

[16] M. Mumtaz and L. Ping, "Forty years of attacks on the RSA cryptosystem: A brief survey," *Journal of Discrete Mathematical Sciences and Cryptography*, vol. 22, no. 1, pp. 9–29 (2019), doi: 10.1080/09720529.2018.1564201.

[17] OEIS Foundation Inc., The On-Line Encyclopedia of Integer Sequences, https://oeis.org.

[18] F. Ruskey, Combinatorial Generation (2003), https://page.math.tu-berlin.de/~felsner/SemWS17-18/Ruskey-Comb-Gen.pdf.

[19] M. Saracevic, S. Adamovic, and E. Bisevac, "Application of Catalan numbers and the lattice path combinatorial problem in cryptography," *Acta Polytechnica Hungarica*, vol. 15, pp. 91–110 (2018), doi: 10.12700/aph.15.7.2018.7.5.

[20] B. Schneier, Applied Cryptography: Protocols, Algorithms, and Source Code in C, Wiley (1996).

[21] Y. Shablya, D. Kruchinin, and V. Kruchinin, "Method for developing combinatorial generation algorithms based on AND/OR trees and its application," *Mathematics*, vol. 8, 962 (2020), doi: 10.3390/math8060962.

[22] Y. Shablya, A. Merinov, and D. Kruchinin, "Combinatorial generation algorithms for directed lattice paths," *Mathematics*, vol. 12, 1207 (2024), doi: 10.3390/math12081207.

[23] K. Sugimoto, T. Nakai, Y. Watanabe, and M. Iwamoto, "The two sheriffs problem: Cryptographic formalization and generalization," *Lecture Notes in Computer Science*, vol. 14461, pp. 512–523 (2023), doi: 10.1007/978-3-031-49611-0_37.

[24] Y. Wei, L. Bi, K. Wang, and X. Lu, "An improved BKW algorithm for solving LWE with small secrets," *Lecture Notes in Computer Science*, vol. 14411, pp. 578–595 (2023), doi: 10.1007/978-3-031-49187-0_29.

[25] Z. Zhiyong, Modern Cryptography: Volume 1, A Classical Introduction to Informational and Mathematical Principle, Springer Nature (2022), doi: 10.1007/978-981-19-0920-7.

Received April, 2024