cryptography

MDPI

Article

Combined and General Methodologies of Key Space Partition for the Cryptanalysis of Block Ciphers

Mijail Borges-Quintana [1,†], Miguel A. Borges-Trenard [2,†], Osmani Tito-Corrioso [3,†], Omar Rojas [4,†] and Guillermo Sosa-Gómez [4,*,†]

[1] Departamento de Matemática, Facultad de Ciencias Naturales y Exactas, Universidad de Oriente, Av. Patricio Lumumba s/n, Santiago de Cuba 90500, Cuba; mijail@uo.edu.cu
[2] Doctorate in Mathematics Education, Universidad Antonio Nariño, Bogotá 111321, Colombia; borgestrenard2014@gmail.com
[3] Departamento de Matemática-Física Aplicada, Facultad de Ingeniería Industrial, Universidad de Matanzas, Autopista a Varadero km 3.5, Matanzas 40100, Cuba; osmanitito94@gmail.com
[4] Facultad de Ciencias Económicas y Empresariales, Universidad Panamericana, Álvaro del Portillo 49, Zapopan 45010, Mexico; orojas@up.edu.mx
* Correspondence: gsosag@up.edu.mx; Tel.: +52-3313682200
† These authors contributed equally to this work.

Abstract: This paper proposes two new methods of key space partitioning for the cryptanalysis of block ciphers. The first one is called combined methodology of key space partition (CoMeKSPar), which allows us to simultaneously set some of the first and last consecutive bits of the key. In this way, the search is performed using the remaining middle bits. CoMeKSPar is a combination of two methods already proposed in the scientific literature, the Borges, Borges, Monier (BBM) and the Tito, Borges, Borges (TBB). The second method is called the general algorithm of key space reduction (GAKSRed), which makes it possible to perform a genetic algorithm search in the space formed by the unknown bits of the key, regardless of their distribution in the binary block. Furthermore, a method of attacking block ciphers is presented for the case where some key bits are known; the basic idea is to deduce some of the remaining bits of the block. An advantage of these methods is that they allow parallel computing, which allows simultaneous searches in different sub-blocks of key bits, thereby increasing the probability of success. The experiments are performed with the KLEIN (Small) lightweight block cipher using the genetic algorithm.

Keywords: optimization; genetic algorithm; key space partition; cryptanalysis; KLEIN

Citation: Borges-Quintana, M.; Borges-Trenard, M.A.; Tito-Corrioso, O.; Rojas, O.; Sosa-Gómez, G. Combined and General Methodologies of Key Space Partition for the Cryptanalysis of Block Ciphers. *Cryptography* **2024**, *8*, 45. https://doi.org/10.3390/cryptography8040045

Academic Editors: Jun Feng, Changqing Luo and Mamoun Alazab

Received: 28 August 2024
Revised: 9 October 2024
Accepted: 9 October 2024
Published: 11 October 2024

1. Introduction

The genetic algorithm (GA) is an optimization method that has been used in cryptography for various purposes in recent years. Some of the research carried out in this area is mentioned below. In [1], a novel image encryption algorithm based on a logistic sine map and the crossover operator of a GA is proposed. Logistic sine maps and crossover are used to generate the random session key for each image encryption. In [2], the authors use the GA to propose an extension to the advanced encryption standard cipher by improving the key generation process. The [3] paper establishes an algorithm using GA to encrypt and decrypt a message based on a symmetric key cryptosystem. Similarly, Ahmed S. Sakr and et al. in [4] propose an amino acid encryption model with two encryption keys, and the first key is randomly generated using a GA. For more information on the use of the GA in cryptography, see references such as [5–9].

This paper presents the results of attacks on the 64-bit key length variant of the KLEIN (*Small*) lightweight block cipher, KLEIN-64. The attack on this cipher involves the use of the genetic algorithm with two different fitness functions and different key space partitioning methods. Our main contribution is a methodology for attacking block ciphers in cases

where some bits of the key are known. The basic idea is to infer some of the remaining bits of the block. On the other hand, a key space partitioning methodology called CoMeKSPar (**Co**mbined **Me**thodology of **K**ey **S**pace **Par**tition) is proposed. This method allows us to simultaneously fix some of the first and last bits of the key while searching for the remaining central bits. This feature is the main advantage of the proposal. It is a combination of two methodologies already proposed in the scientific literature, the Borges, Borges, Monier (BBM) and the Tito, Borges, Borges (TBB). Finally, a second space partition methodology is proposed, called GAKSRed (**G**eneral **A**lgorithm of **K**ey **S**pace **Red**uction), which enables the search with the GA in the space formed by the unknown bits of the key, irrespective of their distribution in the binary block.

2. Materials and Methods

KLEIN cipher

KLEIN is a lightweight block cipher of the permutation substitution network type proposed in [10]. It takes a 64-bit plaintext block as input and returns a 64-bit ciphertext block. Three variants, called KLEIN-64, KLEIN-80, and KLEIN-96, differ in key size: 64, 80, and 96 bits, respectively. All variants internally process 64-bit text blocks. The number of rounds is also different: 12, 16, and 20 for each variant. This algorithm has been extended in several ways. For example, in [11], the key expansion algorithm was modified and a more advanced scheme called N-KLEIN was introduced. In addition, a quantum circuit was implemented on the S-box using the in-place method, which reduced the width and depth of the circuit, thereby improving the implementation efficiency of the quantum circuit. For more details on this block cipher, see, for example, [12–16].

Genetic Algorithm

This section briefly describes the GA scheme used in this paper. In Algorithm 1, the individuals of the population are elements of the key space taken as binary blocks. By **selecting** the s parents, a subset S of P_i is obtained. The parents are selected using the tournament method, where two individuals are randomly chosen and the one with the highest fitness is selected. Elements of the set S are crossed, and the descendants are added to P_i if they are not already members. For **crossover**, the two-point crossover is used, and the probability of two individuals crossing over was set to 0.6 for all experiments. The **mutate** operation randomly mutates up to three components of binary blocks, with a mutation ratio of 0.2 for all experiments.

Let F be a fitness function; an individual x_1 of the population is better adapted than another individual x_2, if it has greater fitness, i.e., if $F(x_1) > F(x_2)$. The following fitness functions are used (see [17]). Let

$$E : \{0,1\}^m \times \{0,1\}^n \to \{0,1\}^n, \tag{1}$$

where $m, n \in \mathbb{N}$ and $m \geq n$ are block ciphers, T is plaintext, K is a key, and C is the corresponding ciphertext, such that, $C = E(K, T)$. The function, based on the Hamming distance d_H, for a certain individual X of the population is,

$$F_1(X) = \frac{n - d_H(C, E(X, T))}{n}, \tag{2}$$

which measures the closeness between the ciphertexts C and the text obtained from encrypting T with the probable key X.

The next fitness function is based on measuring the closeness between ciphertexts, but on their representation in decimal rather than binary. Let Y_d be the corresponding decimal conversion of the binary block Y; the function is

$$F_4(X) = \frac{2^n - 1 - |C_d - E(X, T)_d|}{2^n - 1}. \tag{3}$$

For the GA specification on block ciphers and other details on the values of the operators and parameters used in the experiments of this paper, see Section 3 of [18] and Section 2.1 of [17].

Algorithm 1 Genetic Algorithm

Input: m (number of individuals in the population), F (fitness function), g (number of generations), s (number of individuals selected to mate).
Output: the individuals with the highest fitness function as the best solution.
1: **Generate** randomly an initial population P_i with m individuals.
2: **Calculate** $F(x)$, $\forall x \in P_i$ (the fitness of each individual of P_i).
3: **while** no solution found or g generations not reached **do**
4: **Select** s parents of P_i.
5: To apply the **Crossover** operator to the s selected elements and generate offspring pairs.
6: **Mutate** each of the resulting descendants.
7: **Compute** the fitness function F for each of the offspring and their mutations.
8: Using the Tournament Method between two, based on the aptitudes of the parents and offspring, decide what will be the new population P_{i+1} for the next generation, selecting two individuals at random each time and choosing the higher fitness.
9: **end while**

BBM and TBB key space partition methodologies

The BBM and TBB key space partitioning methods allow the GA to work on a particular subset of the set of admissible solutions as if it were the full set. The partitioning into equivalence classes allows it to use this algorithm in parallel, in multiple classes simultaneously and independently.

Let $\mathbb{F}_2^{k_1}$ be the key space of length $k_1 \in \mathbb{Z}_{>0}$ and $k_2, k_d \in \mathbb{Z}_{>0}$ be such that $1 \leq k_2 < k_1$, $k_d = k_1 - k_2$, and $Q = \{0, 1, 2, \ldots, 2^{k_d} - 1\}$. Then, in both methodologies, the formulas to represent the elements of $\mathbb{F}_2^{k_1}$ are identical, i.e.: $q\,2^{k_2} + r$ with $q \in Q$, $r \in \mathbb{Z}_{>0}$.

Both methodologies involve running the GA on a subset of the key space rather than the entire key space. In the case of BBM, the subset is associated with the class of keys that share the same quotient (q). The TBB method works with a subset defined by keys that share the same remainder (r), where the elements of each class are distributed across the set of keys. The parameters q, r, k_2, and k_d have a dual role in both methods. To avoid ambiguity in the notation, we refer to the parameters of the BBM methodology as q, r, k_2, and k_d. While $\hat{q}$, $\hat{r}$, $\hat{k}_2$ and $\hat{k}_d$ are used to refer to the same parameters in the TBB methodology. See [17,18] for more details on these methods.

In [17], the authors work with the BBM and TBB methodologies; furthermore, the study of certain parameters that intervene in GAs was carried out, such as the time it takes to execute a certain number of iterations; several fitness functions were introduced; and which ones led to better results was analyzed. The experiments were carried out with the block ciphers AES(t). Conversely, in the present investigation, BBM and TBB methodologies are combined to crate a new methodology for key space partitioning; in addition, other methodologies are proposed.

The focus of this research is to propose different methodologies for partitioning the key space. These methodologies can be used with different ciphers and different search algorithms because the main utility is that they allow parallel search in different classes simultaneously. The procedure is adaptable and can be applied to any search algorithm, including brute force search with parallel techniques. For this reason, no comparisons are made between KLEIN and other ciphers, nor with GA, because the same would apply to the others. For this reason, a black box attack is used in the thesis. The goal of the work is to focus on the methods, and they are applied to the search in KLEIN using GA as an example. In an exhaustive search, it is guaranteed that the sought key can be found if the

whole space can be traversed in a reasonable time. Genetic algorithms (GAs) are often more efficient than an exhaustive search for several reasons:

1. Efficient exploration of the search space
   - Evolutionary mechanism: GAs use principles of natural selection, which allow them to explore the search space more efficiently by focusing on the most promising solutions and combining features of those solutions.
   - Populations: Instead of evaluating a single solution at a time, GAs work with a population of solutions, allowing multiple regions of the search space to be explored simultaneously.
2. Eliminating Unpromising Solutions
   - Natural Selection: Through selection, GAs eliminate less effective solutions and focus on those that perform better. This significantly reduces the number of evaluations required compared to an exhaustive search, which evaluates all possible solutions.
3. Genetic Operators.
   - Crossover and Mutation: Crossover and mutation operators allow GAs to generate new solutions from existing ones, making it easier to explore unexplored areas of the search space. This can lead to innovative solutions that would not have been found by an exhaustive search.
4. Fast Convergence.
   - Local Optimization: GAs can quickly converge to optimal or suboptimal solutions by focusing on the most promising parts of the search space, whereas an exhaustive search can take a long time to find a suitable solution.
5. Adaptability.
   - Adaptation to change: GAs can adapt to changes in the problem or environment without having to restart the entire process, unlike an exhaustive search, which would have to start from scratch.
6. Reduced Computational Complexity.
   - Fewer evaluations: For complex problems, the number of possible solutions can be immense. GAs reduce the need to evaluate all possible combinations, saving time and computational resources.

Genetic algorithms use an adaptive and evolutionary approach that allows for smarter and faster exploration of the search space, eliminating the need to evaluate each possible solution individually. This makes them a powerful tool for solving complex problems where the solution space is large and difficult to manage.

Proposed key space partition methodologies

Next, as an introduction to the proposed key space partition methodologies, an application of the TBB and BBM methodologies is demonstrated for attacking block ciphers when some bits of the key are known. The basic idea is to fill in some of the remaining bits. Note that the selection of $\hat{k}_2$ (k_d) and the class in which the search is performed is equivalent to setting the final $\hat{k}_2$ bits of the key in the TBB methodology and the initial k_d bits in the BBM. In that context, if the first or last l bits of the key were known (or wanted to be set as known, which is common in the context of cryptanalysis), then the partition would be created by implementing $\hat{k}_2 = l$ or $k_d = l$. However, the most frequent issue is that some non-consecutive bits are identified, while others are missing to complete the block. In this case, the first (or last) l bits should be filled in by finding all combinations of the components that are unknown in that sub-block of length l.

For example, suppose that for a certain key, K, the first 19 bits are known (the left-most significant bit is taken), $b_1, \ldots, b_{19}$:

$$K = [1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, \underline{b_{20}}, \underline{b_{21}}, \ldots, \underline{b_{64}}], \quad (4)$$

with $b_i \in \mathbb{F}_2$, $i = \overline{1,64}$. So, to find the remaining bits, and therefore, the complete key, the partition can be made by taking $k_d = 19$, and the chosen class is the conversion to decimal of the sub-block formed by the first 19 bits,

$$q = [1,0,1,0,0,1,0,1,0,0,1,1,0,1,0,1,0,0,0]_d. \tag{5}$$

Following the idea of completing bits, the size of the class in which the search is performed could be reduced by completing one or more of the following bits. In this case, since the bits can only take two values, and therefore, the next component, b_{20}, can only be 1 or 0, we could take $k_d = 20$ and search in two classes (with fewer elements):

$$q_1 = [1,0,1,0,0,1,0,1,0,0,1,1,0,1,0,1,0,0,0,\mathbf{0}]_d, \tag{6}$$
$$q_2 = [1,0,1,0,0,1,0,1,0,0,1,1,0,1,0,1,0,0,0,\mathbf{1}]_d. \tag{7}$$

Note that in these cases, key space partition methodologies are advantageous because they allow parallel searching of several classes at the same time. In the case where some bits are known discontinuously, one can proceed in a similar way. Suppose the key has the following structure:

$$K = [1,\underline{b_2},0,1,0,1,\underline{b_7},0,\underline{b_9},1,0,1,0,0,1,0,1,0,0,1,\underline{b_{21}},\underline{b_{22}},\ldots,\underline{b_{64}}]. \tag{8}$$

where it can be seen that 17 bits of the total 64 are known. The largest number of known bits are in the first part of the block, so the BBM methodology must be used again and, therefore, partitioned by choosing $k_d = 20$, which is the range between the first and last known bit. In this range, the bits b_2, b_7, and b_9 are missing. Filling in these bits, there are $2^3 = 8$ classes in which to search for the key. It is clear that the key is in one of these classes. Note that the above is true for both methods. The examples were performed by setting the first few bits. In the case of the last bits, the TBB method would be used in a similar way, where the class would be chosen as $\hat{r}$ equal to the decimal conversion of the last sub-block of the key.

Combined Methodology of Key Space Partition (CoMeKSPar)

With the information provided in the previous section, one can appreciate the usefulness of a tool for cryptanalysis when certain bits of the key are known or need to be fixed. The proposal reflects the complementary nature of the BBM and TBB methodologies. However, there is a problem with knowing both the start and end bits at the same time. The BBM and TBB methodologies would not solve the situation based on the information revealed so far. By using either of them, knowledge of the first or last bits would be sacrificed. This is because when a portion of the block is fixed, the search is performed on the remaining portion as a whole.

A possible alternative would be to use a fitness function that takes advantage of knowledge of certain components of the key. However, this approach would have two challenges. First, it would limit the number of fitness functions the GA could use within the same partition. On the other hand, the fitness function takes into account the known bits, but the search performed by the GA is blind to this information. In other words, it is as if no bit of the part of the key that was not set during the partition (where the GA searches) is known.

With regard to this problem, a key space partitioning methodology is proposed in this section. This methodology allows us to fix the first and the last bits of the key simultaneously. In this way, the search is performed in the remaining central bits, which is the main benefit of this proposal. This is a combination of the BBM and TBB methodologies, which is referred to as **CoMeKSPar** (**Co**mbined **Me**thodology of **K**ey **S**pace **Par**tition).

The general idea is to first apply the TBB methodology to the entire key space and then apply the BBM methodology to the subset Q of the TBB over which $\hat{q}$ moves. When we use the TBB methodology, the elements of the space $\mathbb{Z}_{2^{k_1}}$ are obtained by the expression

$$\hat{q}\,2^{\hat{k}_2} + \hat{r},\ \hat{q} \in Q, \tag{9}$$

where $\hat{r} \in \mathbb{Z}_{2^{\hat{k}_2}}$ and $Q = \{0, 1, 2, \ldots, 2^{k_1 - \hat{k}_2} - 1\}$; in particular, note that $|Q| = 2^{k_1 - \hat{k}_2} = 2^{\hat{k}_d}$. Now, in this methodology, $\hat{r}$ is set to choose the class, which is equivalent to setting the last $\hat{k}_2$ components, and then $\hat{q}$ varies by Q to move through the elements of said class.

The next thing is to apply the BBM methodology on the set Q as if it were the entire space. Let $k_3 \in \mathbb{Z}_{>0}$, $0 < k_3 < \hat{k}_d$. Dividing $|Q| = 2^{\hat{k}_d}$ by 2^{k_3}, the set Q is divided into $2^{\hat{k}_d - k_3}$ subsets, with 2^{k_3} elements each. Taking

$$q \in [0, 2^{\hat{k}_d - k_3} - 1] \subset \mathbb{Z}_+, \text{ and, } r \in [0, 2^{k_3} - 1] \subset \mathbb{Z}_+, \tag{10}$$

then, an element $\hat{q} \in Q$ is expressed as

$$\hat{q} = q\,2^{k_3} + r. \tag{11}$$

Note that q and r are the same parameters of the BBM methodology, only the space has been reduced to Q. With q, the subinterval (or class) is fixed, and with r, the position within it. Now the search is performed in the set $[0,\ 2^{k_3} - 1]$, where r is free. Note that by choosing q, the first k_3 bits of the key are being set to $\mathbb{Z}_{2^{k_1}}$.

Now, to recover the complete key in $\mathbb{Z}_{2^{k_1}}$, we substitute (11) in (9), from which we obtain

$$(q\,2^{k_3} + r)\,2^{\hat{k}_2} + \hat{r}, \tag{12}$$

where k_3, $\hat{k}_2$, q, and $\hat{r}$ are fixed, and only r varies by $[0,\ 2^{k_3} - 1]$. With the above, it is guaranteed to be able to fix the last $\hat{k}_2$ and the first k_3 bits of the key. It is interesting to note that it is equivalent to applying the BBM methodology first and then the TBB. However, it does not make sense to use the same methodology twice, since it would be equivalent to using it only once.

General Algorithm of Key Space Reduction (GAKSRed)

The most general case is when the known bits of the key are distributed over the block of length k_1. This includes that some components are continuous. This section proposes a key space partitioning methodology that allows searching with the GA in the space created by the unknown bits of the key, regardless of their distribution in the binary block. This methodology is referred to as GAKSRed (**G**eneral **A**lgorithm of **K**ey **S**pace **Red**uction).

Let $\mathbb{F}_2^{k_1}$ be the space in keys of length $k_1 \in \mathbb{Z}_+^*$. Let $B \in \mathbb{F}_2^{k_1}$ be a key. From now on, $b_{[j]} \in \mathbb{F}_2$ denotes the value of the component that occupies the position $1 \leq j \leq k_1$ in the binary block. Suppose that from B, the following l bits distributed randomly throughout the block are known:

$$b_{[\delta_1]},\ b_{[\delta_2]},\ \ldots,\ b_{[\delta_{l-1}]},\ b_{[\delta_l]}, \tag{13}$$

with $b_{[\delta_i]} \in \mathbb{F}_2$, $\delta_i \in \{1, 2, \ldots, k_1\}$ and $i = \overline{1, l}$. Let us denote as B^+ the list formed by these components, in that order,

$$B^+ = [b_{[\delta_1]},\ b_{[\delta_2]},\ \ldots,\ b_{[\delta_{l-1}]},\ b_{[\delta_l]}], \tag{14}$$

and as B^+_{idx}, to the list of the indexes:

$$B^+_{\text{idx}} = [\delta_1,\ \delta_2,\ \ldots,\ \delta_{l-1},\ \delta_l]. \tag{15}$$

The components are always taken in ascending order in relation to their position in the block, i.e., $i < j$ if and only if $\delta_i < \delta_j$. On the other hand, the first position is occupied by the leftmost bit of B, and the last bit is the rightmost bit of B.

By fixing the l bits of (13), the entire key is recovered if all combinations of the remaining $k_1 - l$ bits that are not known are found and evaluated. This would give a total of 2^{k_1-l} possible combinations. They would go from making all (unknown) bits equal to 0 to all equal to 1. Therefore, in decimal base, it would be equivalent to searching from 0 to $2^{k_1-l} - 1$. The only problem would be that these unknown components are not continuous, but are scattered throughout B forming several sub-blocks, for this reason, it is important to pay attention to the position occupied by the bits in the block.

Let $b_{[\omega_1]}, b_{[\omega_2]}, \ldots, b_{[\omega_{k_1-l-1}]}, b_{[\omega_{k_1-l}]}$ be the unknown bits of B. Where $b_{[\omega_j]} \in \mathbb{F}_2$, $\omega_j \in \{1, 2, \ldots, k_1\}$, and $j = \overline{1, (k_1 - l)}$. As explained above, a sub-block of $k_1 - l$ components is formed by concatenating these bits. Let us denote as B^- the list formed by these components, in that order,

$$B^- = [b_{[\omega_1]}, b_{[\omega_2]}, \ldots, b_{[\omega_{k_1-l-1}]}, b_{[\omega_{k_1-l}]}], \tag{16}$$

and as B^-_{idx}, to the list of the indexes:

$$B^-_{\text{idx}} = [\omega_1, \omega_2, \ldots, \omega_{k_1-l-1}, \omega_{k_1-l}]. \tag{17}$$

Now the space in which the search is performed is obtained by calculating the combinations of this sub-block. This is equivalent to searching the space $\mathbb{U} = \{0, 1, \ldots, 2^{k_1-l} - 1\}$.

Note that $\mathbb{U}$ is isomorphic to $\mathbb{Z}_{2^{k_1-l}}$; however, the notation change is due to the different way of obtaining the set $\mathbb{U}$. In this sense, what has been carried out so far is (1) separating from B the $k_1 - l$ unknown bits, saving their positions in the block; (2) concatenating these components; (3) and performing the search, using the GA (or another algorithm), in the set $\mathbb{U}$. In other words, with $\mathbb{U}$, we are referring to all the possible combinations of the unknown components of B. The elements of $\mathbb{U}$ represent the decimal conversion of blocks of length $k_1 - l$. We select a class by fixing the l known bits of B, and the search space is reduced to $\mathbb{U}$.

To retrieve B from B^+ and B^-, the function $Tog(B^+, B^+_{\text{idx}}, B^-, B^-_{\text{idx}})$ is suggested. This function creates a binary block B' of length k_1 in which it places the components of B^+ and B^-, taking into account the place (indexed by B^+_{idx} and B^-_{idx}, respectively) corresponding to each bit. In the programming of this methodology, it is possible to have, in the pre-calculation phase, B' with the components of B^+ in their position; therefore, in Tog, it would not be necessary to create a new variable B' each time.

A way to obtain an element B could be as follows. The k_1 bits of B are traversed sequentially: $B[i]$, $i \in \{1, 2, \ldots, k_1\}$. Now, if $i \in B^+_{\text{idx}}$, then first return the place that i occupies in B^+_{idx}, its index: $I = index(i, B^+_{\text{idx}})$, then $\vec{B}[i] = B^+[I]$. Otherwise, if $i \in B^-_{\text{idx}}$, then $I = index(i, B^-_{\text{idx}})$, and then $B[i] = B^-[I]$.

Now, similar to the previous methodologies, given $v \in \mathbb{U}$, it is necessary to have a way to search for the element that represents v in $\mathbb{F}_2^{k_1}$. For this purpose, v must first be converted to a binary block of length $k_1 - l$. Let V be the binary block of v:

$$V = [b'_{[\omega_1]}, b'_{[\omega_2]}, \ldots, b'_{[\omega_{k_1-l-1}]}, b'_{[\omega_{k_1-l}]}]. \tag{18}$$

Each one of the bits of V is inserted in the corresponding components that occupy the positions ω_j in B, for they, together with the $b_{[\delta_i]}$ bits already known (and fixed from the beginning), form the element B' of $\mathbb{F}_2^{k_1}$ that is represented by v in $\mathbb{U}$. For this reason, the positions (indexes) of the known bits and the remaining bits must be saved. In other words, the idea is to apply the Tog function taking V as if it were B^-: $B' = Tog(B^+, B^+_{\text{idx}}, V, B^-_{\text{idx}})$.

The GAKSRed methodology also constitutes a formalization of the procedure of iterative fixing components of the keys used in [19] to design a method of cryptanalysis of PRESENT-80 using the genetic algorithm, progressively reducing the set of possible solutions.

Let us look at the following example. Suppose $l = 10$ bits of a key B of a hypothetical length $k_1 = 26$ bits are known:

$$B = [b_1, b_2, \underline{1}, b_4, \underline{0}, b_6, \underline{0}, b_8, b_9, b_{10}, b_{11}, b_{12}, \underline{0}, \underline{1}, b_{15}, b_{16}, \underline{0}, b_{18}, \underline{0}, b_{20}, b_{21}, \underline{1}, b_{23}, b_{24}, \underline{0}, \underline{1}]. \quad (19)$$

By separating the known bits and the rest, together with their corresponding indexes, we have

$$\begin{aligned} B^+ &= [\underline{1}, \underline{0}, \underline{0}, \underline{0}, \underline{1}, \underline{0}, \underline{0}, \underline{1}, \underline{0}, \underline{1}], & (20)\\ B^+_{\text{idx}} &= [3, 5, 7, 13, 14, 17, 19, 22, 25, 26], & (21)\\ B^- &= [b_1, b_2, b_4, b_6, b_8, b_9, b_{10}, b_{11}, b_{12}, b_{15}, b_{16}, b_{18}, b_{20}, b_{21}, b_{23}, b_{24}], & (22)\\ B^-_{\text{idx}} &= [1, 2, 4, 6, 8, 9, 10, 11, 12, 15, 16, 18, 20, 21, 23, 24]. & (23) \end{aligned}$$

Since B^- has $k_1 - l = 26 - 10 = 16$ components, then the search is performed in the space $\mathbb{U} = \{0, 1, 2, \ldots, 2^{16} - 1\}$. Suppose now that we have $v = 36\,222 \in \mathbb{U}$, which in binary would be $V = [1, 0, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0]$. Applying the *Tog* function, we obtain

$$\begin{aligned} B' &= Tog(B^+, B^+_{\text{idx}}, V, B^-_{\text{idx}}), & (24)\\ B' &= [1, 0, \underline{1}, 0, \underline{0}, 0, \underline{0}, 1, 1, 0, 1, 0, \underline{0}, \underline{1}, 1, 1, \underline{0}, 1, \underline{0}, 1, 1, \underline{1}, 1, 0, \underline{0}, \underline{1}]. & (25) \end{aligned}$$

where B' is the key that represents V in $\mathbb{F}_2^{26}$. Any variation in V is also made in B' in the components for which no information is known. Looping exhaustively through the set $\mathbb{U}$ is equivalent to searching for all possible combinations in B while keeping the known bits fixed.

Note that this methodology is not isolated from the other three. If the set $\mathbb{U}$ is still too large, it can be reduced again by applying either of the BBM or TBB methodologies (hence the CoMeKSPar). The above is possible because $\mathbb{U} \cong \mathbb{Z}_{2^{k_1-l}} \cong [0,\ 2^{k_1-l} - 1] \subset \mathbb{Z}_+$. This feature would enable parallel searches in different classes simultaneously. Note that these methodologies do not influence the size of the space (or subset) where the search is carried out with the GA. In other words, if d bits of a key are not known, the subset where the key is searched is $2^d - 1$ elements regardless of the methodology used. One methodology will be chosen based on the distribution of the unknown bits.

3. Experiments and Discussion

Attack with the BBM and TBB Methodologies

A personal computer (PC) laptop with the following specifications was used for the experiments: Intel(R) Celeron(R) CPU N3050 @1.60 GHz (2 CPUs), ∼1.6 GHz, and 4 GB of RAM memory. The attack is a known plaintext attack and a black box attack. A total of 60 attempts were made to find the key, with 30 attempts for each space partitioning methodology. Of the 30 attempts, 15 were made using each of the fitness functions F_1 and F_4. At the same time, out of the 15, 10 were made for $k_2 = \hat{k}_d = 14$ and 5 for $k_2 = \hat{k}_d = 16$. The total number of generations the GA has to go through is 163 for $k_2 = \hat{k}_d = 14$ and 655 for $k_2 = \hat{k}_d = 16$, and the number of individuals in the population is 100 (the population size) for all cases. In all trials, the classes in which the key was found were searched. The key was found in 38 out of 60 attempts, resulting in a success rate of $63.3\bar{3}\%$.

For the BBM methodology with functions F_1 and F_4, for $k_2 = 14$, the key was found in 7 and 4 out of 10 attempts, respectively. In both cases, for $k_2 = 16$, the key was found in 4 out of 5 attempts. In total, the BBM methodology found the solution in 19 out of 30 attempts, for $63.3\bar{3}\%$ of correct answers. On average, it took 87.1 generations and about 28.1 min to find the solution with F_1. The key was found using F_4 after 112.1 generations and 35.6 min.

In the case of the TBB methodology, out of 10 attempts for $\hat{k}_d = 14$, a positive solution was found in 5 attempts for the function F_1 and in 6 attempts for the function F_4. For $\hat{k}_d = 16$, the key was found in 5 and 3 tries. As in the first methodology, the solution was

found in 19 out of 30 attempts for 63.3$\bar{3}$% of correct answers. On average, it took about 90.8 generations and 27.9 min to find the solution with F_1. With F_4, the key was found in about 103.6 generations and 32.4 min. Note that even though more generations were needed than in BBM, the average time was slightly less in this case.

In both methodologies, in cases where the key was not found, the times were as follows. For $k_2 = \hat{k}_d = 14$ and $k_2 = \hat{k}_d = 16$, the GA took an average of 47 min and 3.37 h, respectively.

Attack with the CoMeKSPar methodology

The experiments were performed with the same parameters and conditions as before. Out of the 10 attempts for $k_3 = 14$, a positive solution was obtained in 5 and 9 attempts. As for $k_3 = 16$, the key was found in 5 and 4 attempts.

The solution was found in 23 out of 30 attempts for a success rate of 76.6$\bar{6}$%. On average, it took about 176.1 generations and 55.26 min to find the solution for F_1. The key for F_4 was found after 111.77 generations and 33.97 min. Note that F_1 required more generations than the BBM and TBB methodologies and, therefore, more time. The results for F_4 were similar. On the other hand, it is worth noting that the effectiveness of CoMeKSPar was greater than the results obtained with BBM and TBB. In cases where the key was not found, the times were similar to those obtained with the BBM and TBB methodologies. Interestingly, all of the methodologies complement each other in terms of utility by fixing some known bits of the key when performing the partition.

Experiments with GAKSRed would give similar results to CoMeKSPar. The difference is that GAKSRed would work with any distribution in the binary block of the known bits of the key, but the behavior in terms of experiments and the size of the search space would be similar to that of CoMeKSPar (depending on the number of bits assumed to be known).

These methodologies are new compared to the previous works, such as [17,18]. The choice of encryption is also different; in the case of this article, it is KLEIN, another encryption method with moderate parameters.

These experiments show that it is possible to divide the key space into classes and find the keys in use. At the same time, the results confirm the usefulness of the proposed methodologies. An advantage of these methodologies is that they allow the use of parallel computing. This allows us to search simultaneously in different sub-blocks of bits of the key, which increases the probability of success.

4. Conclusions

An attack methodology has been revealed for cases where some bits of the key are known. It involves completing the remaining components of the block. On the other hand, a key space partition methodology called CoMeKSPar has been proposed. This method allows the first and last bits of the key to be set simultaneously while the search is performed in the remaining central bits. This methodology is a combination of BBM and TBB. A second space partition methodology called GAKSRed has been proposed. It allows searching with the GA in the space created by the unknown bits of the key, regardless of their distribution in the binary block. This methodology is a generalization that allows fixing a number of bits of the key in any position. It makes it possible to perform experiments with arbitrarily distributed search positions of similar length. The important thing would be the value of the parameters for performing the search, not the positions themselves.

For future research, it would be interesting to perform attacks on KLEIN by increasing the size of the classes. This involves experimenting with values of k_2 and $\hat{k}_d$ greater than 16. In addition, the two proposed methodologies for analyzing the key space can be applied to other families of block ciphers.

Author Contributions: Conceptualization, M.B.-Q., M.A.B.-T., O.T.-C., and G.S.-G.; formal analysis, M.B.-Q. and O.T.-C.; investigation, M.B.-Q., M.A.B.-T., O.R., and G.S.-G.; methodology, M.B.-Q. and M.A.B.-T.; project administration, O.R. and G.S.-G.; supervision, O.R.; validation, M.B.-Q., O.T.-C., and G.S.-G.; writing—original draft, M.B.-Q. and M.A.B.-T.; writing—review and editing, O.R. and G.S.-G. All authors have read and agreed to the published version of the manuscript.

Funding: The research associated with the results presented in this publication received funds from the International Funds and Projects Management Office under the code PN223LH006-015, and also from Red CYTED 522RT0131 "Nuevas Herramientas Criptográfias para la E-comunidad".

Informed Consent Statement: Not applicable.

Data Availability Statement: The original contributions presented in the study are included in the article.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Gupta, M.; Gupta, K.K.; Shukla, P.K. Session Key based Image Cryptographic Algorithm using Logistic-Sine Map and Crossover Operator for IoT. *J. Sci. Res.* **2021**, *65*, 260–265. [CrossRef]
2. Bagane, P.; Kotrappa, D.S. Enriching AES Through The Key Generation From Genetic Algorithm. *Indian J. Comput. Sci. Eng.* **2021**, *12*, 955–963. [CrossRef]
3. Mittal, A. A New Cryptographic Technique Involving Genetic Algorithm. *PAIDEUMA J.* **2022**, *XV*, 31–41.
4. Sakr, A.S.; Shams, M.Y.; Mahmoud, A.; Zidan, M. Amino Acid Encryption Method Using Genetic Algorithm for Key Generation. *Comput. Mater. Contin.* **2022**, *70*, 123–134. [CrossRef]
5. Din, M.; Pal, S.K.; Muttoo, S.K.; Madan, S. A Hybrid Computational Intelligence-based Technique for Automatic Cryptanalysis of Playfair Ciphers. *Def. Sci. J.* **2020**, *70*, 612–618. [CrossRef]
6. Qobbi, Y.; Jarjar, A.; Essaid, M.; Benazzi, A. Image Encryption Algorithm based on Genetic Crossover and Chaotic DNA Encoding. *Soft Comput.* **2022**, *26*, 5823–5832. [CrossRef]
7. Sabonchi, A.; Akay, B. A survey on the Metaheuristics for Cryptanalysis of Substitution and Transposition Ciphers. *Comput. Syst. Sci. Eng.* **2021**, *39*, 87–106. [CrossRef]
8. Tito-Corrioso, O.; Borges-Trenard, M.A.; Borges-Quintana, M. Ataques a cifrados en bloques mediante búsquedas en grupos cocientes de las claves. *Cienc. Matemáticas* **2019**, *33*, 71–74.
9. Tiwari, M.; Pinheiro, D.; Shukla, S.; Poptani, S.; Natarajan, D. Cryptanalysis Using Genetic Algorithm. *Int. Res. J. Adv. Eng. Sci.* **2020**, *5*, 128–131.
10. Gong, Z.; Nikova, S.; Law, Y.W. KLEIN: A New Family of Lightweight Block Ciphers. In *International Workshop on Radio Frequency Identification: Security and Privacy Issues*; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2011; Volume 7055, pp. 1–18.
11. Yanjun, L.; Yaodong, G.; Qi, W.; Weiguo, Z.; Chen, L. Improved KLEIN algorithm and its quantum analysis. *J. Comput. Appl.* **2023**, *44*, 2810–2817. [CrossRef]
12. Alregabo, A.; Hikmat, Y. Block Cipher Performance and Risk Analysis. *Al-Rafidain J. Comput. Sci. Math. (RJCM)* **2023**, *17*, 23–33. [CrossRef]
13. Bhatiya, M.R. A Study and Analysis on Color Coded Cryptography on Textual Data. *J. Image Process. Intell. Remote Sens.* **2022**, *2*, 15–21. [CrossRef]
14. Ghorashi, S.; Zia, T.; Jiang, Y.; Bewong, M. Software optimisation of lightweight Klein encryption in the Internet of Things. *J. Inf. Secur. Cybercrimes Res.* **2021**, *4*, 159–172. [CrossRef]
15. İlter, M.; Selçuk, A. A New MILP Model for Matrix Multiplications with Applications to KLEIN and PRINCE. In Proceedings of the 18th International Conference on Security and Cryptography (SECRYPT 2021), SCITEPRESS, Online, 6–8 July 2021; Science and Technology Publications: Lda, Pakistan, 2021; pp. 420–427. [CrossRef]
16. Long, M.; Kong, M.; Long, S.; Zhang, X. An Improved Differential Fault Analysis on Block Cipher KLEIN-64. *Comput. Mater. Contin.* **2020**, *65*, 1425–1436. [CrossRef]
17. Tito-Corrioso, O.; Borges-Trenard, M.; Borges-Quintana, M.; Rojas, O.; Sosa-Gómez, G. Study of Parameters in the Genetic Algorithm for the Attack on Block Ciphers. *Symmetry* **2021**, *13*, 806. [CrossRef]
18. Borges-Trenard, M.; Borges-Quintana, M.; Monier-Columbié, L. An application of genetic algorithm to cryptanalysis of block ciphers by partitioning the key space. *J. Discret. Math. Sci. Cryptogr.* **2019**, *25*, 325–334. [CrossRef]
19. Donatien-Charón, A.; Borges-Trenard, M.; Borges-Quintana, M. Ataque al PRESENT-80 con el Algoritmo Genético mediante aproximaciones sucesivas de componentes fijas. *Rev. Cuba. Cienc. Inform.* **2023**, *17*, 1–15.