Article

E-ART: A New Encryption Algorithm Based on the Reflection of Binary Search Tree

Bayan Alabdullah [1,2,*], Natalia Beloff [1] and Martin White [1]

1 Department of Informatics, University of Sussex, Falmar BN1 4GE, UK; n.beloff@sussex.ac.uk (N.B.); m.white@sussex.ac.uk (M.W.)
2 Computer Science Department, Princess Nourah Bint Abdul Rahman University, Riyadh 11543, Saudi Arabia
* Correspondence: B.Alabdullah@sussex.ac.uk

Abstract: Data security has become crucial to most enterprise and government applications due to the increasing amount of data generated, collected, and analyzed. Many algorithms have been developed to secure data storage and transmission. However, most existing solutions require multi-round functions to prevent differential and linear attacks. This results in longer execution times and greater memory consumption, which are not suitable for large datasets or delay-sensitive systems. To address these issues, this work proposes a novel algorithm that uses, on one hand, the reflection property of a balanced binary search tree data structure to minimize the overhead, and on the other hand, a dynamic offset to achieve a high security level. The performance and security of the proposed algorithm were compared to Advanced Encryption Standard and Data Encryption Standard symmetric encryption algorithms. The proposed algorithm achieved the lowest running time with comparable memory usage and satisfied the avalanche effect criterion with 50.1%. Furthermore, the randomness of the dynamic offset passed a series of National Institute of Standards and Technology (NIST) statistical tests.

Keywords: security; cryptography; dynamic key; binary tree; ASCII

Citation: Alabdullah, B.; Beloff, N.; White, M. E-ART: A New Encryption Algorithm Based on the Reflection of Binary Search Tree. *Cryptography* **2021**, *5*, 4. https://doi.org/10.3390/cryptography5010004

Received: 16 December 2020
Accepted: 26 January 2021
Published: 29 January 2021

1. Introduction

The emergence of social media platforms and smartphone applications has resulted in the generation of vast volumes of data, referred to as "big data." Big data is defined as massive and diverse datasets that exceed the computational, storage, and communication capabilities of traditional methods or systems [1]. These data are used for further analysis to provide insights into applications related to domains such as healthcare, banking, and finance. The collection and storage of potentially sensitive information raises serious security and privacy concerns.

Breaches of sensitive data expose organisations to many threats, including reputational risk, financial penalties, and other fines for non-compliance with regulations that require high levels of security to protect sensitive data [1,2]. For example, the General Data Protection Regulation (GDPR) [3] outlines a specific set of rules for transferring and storing personal data to protect individual privacy. One way to reduce the burden on GDPR is to encrypt personal data. For example, organizations that encrypt their personal data gain the benefit of not having to notify data subjects in cases of data breach. Hence, storing data in encrypted format could reduce the cost of personal data and information systems.

Organizations of all sizes are adapting different security and privacy preservation techniques to ensure data security and privacy compliance requirements, protect intellectual property, and secure their customers' personally identifiable information and company information. There is no universally suitable technique; instead, solutions will depend on the specific needs of an organization.

1.1. Motivation and Research Goals

As the applications and capabilities of big data have been expanding dramatically, data privacy and security have become significant issues given, for example, the shift from clients to cloud servers, the rise of the Internet of Things, and the shift from desktop computers to mobiles and other small devices. Although robust cryptographic solutions are available [4–7], their application to an ever-increasing volume, variety, and speed remains challenging [8]. Most existing cryptographic systems rely on increasing the key size and the number of rounds to enhance security and reduce the risk of cryptanalysis attacks. This causes overheads in terms of latency and computational resources for real-time applications. For example, the Advanced Encryption Standard (AES) [4] requires several iterations over a round function, which negatively impacts the performance of the system. As a result, many current applications have abandoned data encryption in order to reach an adoptive performance level [9]. Therefore, there is a need for an effective cryptographic algorithm to fulfil the requirements of big data, such as speed and security, in a cost-effective manner. This paper focuses on designing and implementing a new and alternative cipher scheme that can overcome the disadvantages of existing ciphers such large key and complex computation. The overall goal is to achieve a satisfactory level of security with shorter execution times and fewer resources.

1.2. Contribution

This work addresses the problem of transmission and storage security of sensitive data while maintaining low computational and latency overheads. We propose a new efficient, flexible, and secure encryption algorithm that adopts the dynamic key concept along with a balanced binary search tree data structure. The proposed algorithm uses a single round, which requires few processes and ensures good cryptographic performance.

The proposed cryptographic approach adopts several operations during the encryption process to achieve a high level of efficiency and security, as follows:

(1) A balanced tree data structure along with American Standard Code for Information Interchange (ASCII) values of text characters to encode data. This makes searching for a particular character value more efficient, as there is no need to visit every node when searching for a specific value. Thus, higher computational efficiency is achieved.

(2) Dynamic keys based on a pseudo-random generator. Each character in the text document is encrypted with a different cryptographic key. The character's position is used as a seed in the random number generation function to produce the pseudo-random number. This ensures a high level of security against classical and modern powerful attacks, which is traditionally ensured by increasing the key size, without scarifying performance.

The performance and security of the proposed algorithm were compared to benchmarked symmetric encryption algorithms AES-128 and Data Encryption Standard (DES). Performance was compared in terms of processing time and memory usage. Security was assessed through the avalanche effect, frequency analysis, and the National Institute of Standards and Technology (NIST) test.

The rest of the paper is organized as follows. Section 2 reviews related work. Section 3 introduces the system framework. Section 4 presents the experimental evaluation. Section 5 discusses the results. Section 6 concludes the paper.

2. Background and Related Work

Cryptography is the ancient practice of securing data for transmission and storage. It is a set of processes or functions using keys to encrypt plain text so that only those for whom it is intended can read and process it. Cryptographic algorithms can be divided into two main categories: symmetric and asymmetric key encryption. Symmetric key encryption uses one secret key to encrypt and decrypt data, whereas asymmetric key encryption requires two different keys (public and private keys). Classic ciphers are character-oriented ciphers which can only be used to encrypt text. These can be divided into substitution ciphers and transposition ciphers. In a substitution cipher, each plaintext character is replaced by another character, either using a fixed replacement structure (monoalphabetic ciphers) or variable replacement structure (polyalphabetic ciphers). In transposition ciphers, plaintext characters are shifted depending on a given mapping key [10]. These ciphers are highly susceptible to a cryptanalysis attack [11]. However, there have been some recent enhancements [12,13] to these ciphers to overcome these attacks and to increase security and maintain performance. For example, Marzan and Sison [14] enhanced the key security of the Playfair Cipher Algorithm using a combination of a 16 $\times$ 16 matrix, XOR, two's complement, and bit swapping. Aung and Hal [15] combined a Vigenère cipher with an Affine cipher to increase the level diffusion and confusion properties.

Elmogy et al. [16] have proposed a new character-oriented cipher based on ASCII Code and the relationship between plaintext characters. The cipher uses simple operations like addition and subtraction and each character is encrypted differently based on the position of the previous character to avoid a frequency analysis attack. However, the cipher can be further improved by adding a random key generator. Similarly, Yadav et al. [17] propose a new symmetric algorithm to use ASCII conversion and the length of the plaintext to create a square matrix. The key is randomly generated based on the order of the square matrix.

Modern ciphers on the other hand are bit-oriented ciphers that can be used to encrypt any form of data. Simple examples of these ciphers are XOR ciphers, rotation ciphers, and S-boxes. Simple modern ciphers have led to a new form of cipher called a product cipher or round cipher [18]. Product ciphers combine two or more transformations such as S-box, permutation and modular arithmetic [18]. The concept of product ciphers introduced by Shanoun [19] establishes two main properties for the design of cryptographic algorithms: confusion and diffusion. Confusion obscures the relationship between the plaintext and ciphertext, whereas diffusion dissipates the statistical structure of plaintext over the bulk of ciphertext. These two properties can be achieved by producing a product cipher with multiple iterations. Each iteration works by combining different transformations to construct a complex encryption function. There are various implementations of these techniques, such as DES [7], 3DES [20], AES [2–5], and BLOWFISH [21].

Symmetric algorithms are classified into two groups: block and stream ciphers. Block ciphers encrypt a fixed size of n-bits of plaintext at once (e.g., 64 bits) while stream ciphers encrypt 1 bit or byte of plaintext at a time [22]. The block cipher algorithm is preferred to the stream cipher for faster computations [23]. While symmetric key algorithms have been considered a cryptographic solution for emerging big data applications, the cumbersome key management and distribution of this approach does not provide a suitable level of scalability. Therefore, more lightweight and practical alternatives need to be developed [24].

Several systems and approaches have been proposed to reduce the required computational resources and latency to overcome the limitations of big data encryption. To address key generation and management issues, Aljawarneh et al. [25] proposed a multithreaded encryption system for securing big data that generates the key from the plaintext. In this work, the encryption algorithm combined the Feistel network, AES with substitution boxes, and a genetic algorithm. First, the input file is divided into several equally sized blocks, and then each block is split into plaintext and key parts. The Feistel network produces a cipher key that is used in the AES component and the genetic algorithm. The algorithm was evaluated using medical-based multimedia big data and compared to existing standard encryption algorithms in terms of runtime and avalanche effect with promising results. Dawood et al. [26] proposed a new symmetric block cipher model for securing big data. It uses a 512-bit block size and a key length of 128 bits, which can be expanded to up to 512 bits. The cipher supports high key agility and relatively fast encryption speed. However, it is designed with the heavy weight of eight degrees of polynomial equations and three layers of four iterated stages, which makes the encryption a heavyweight process. Lightweight Dynamic Crypto (LWDC) [27] is another block cipher that was proposed to address the speed requirements of modern applications. Encryption and decryption use simple XOR operations followed by substitutions and transpositions along with Cryptographically

Secure Pseudo Random Number Generators (CSPRNG) to generate and share the shared value. The cipher outperforms AES in execution time and the CSPRNG puts the cipher on the level of modern symmetric encryptions. Selective data encryption is considered a way of reducing computing cost while protecting data in clouds. For example, Gai et al. [9] attempt to address the privacy concern that arises from unencrypted transmissions of large amounts of data. They propose a new model that aims to maximize the privacy protection scope by using a selective encryption strategy within the required execution time requirements. To deal with the computation workload caused by large-volume data, this method gives encryption priority to data that carry sensitive information. It uses a Dynamic Encryption Determination (DED) algorithm to dynamically select data packages that can be encrypted under different timing constraints.

Dynamic keys have been used in several encryption approaches to overcome multi-round computational complexity [28–30]. These approaches follow a dynamic structure where the structure of all cipher primitives, such as substitution and permutation tables, changes depending on the dynamic key, which allows a reduction in the number of rounds, leading to a reduction in computational overhead without lowering the security level [28]. A single-round structure cipher to encrypt two blocks at a time was proposed in [31]. The mode of operation is based on the dynamic key approach, whereby blocks are selected and mixed according to a dynamic permutation table. A similar approach was adopted in [32], where a lightweight cipher schema generated a dynamic key for each input message by hashing the session key.

Chaos theory was recently used in cryptosystem design [30] due to its desirable features, such as pseudo-randomness, complexity, and sensitivity to initial parameter changes [8]. Jallouli et al. [33] proposed a stream cipher that uses a combination of multiple chaotic maps for improved robustness, security, and complexity. However, most of these schemas have various limitations, such as vulnerability to classical attacks [34] and complexity of floating computation and hardware implementation [35]. Recently, Ding et al. [36] attempted to overcome the chaos problem by proposing a chaos-based algorithm that utilizes a logistic map alongside nonlinear feedback shift registers. The algorithm has been analyzed using conventional cryptanalysis as well as a statistics-based experiment and the results were encouraging. Other studies have attempted to hybridize chaos theory with existing encryption algorithms, such as AES [37] and S-AES [8] and DNA encoding techniques [38]. In these studies, chaos theory was used to produce the encryption keys or the permutation tables used for encryption.

3. Proposed Schema

In this work, we propose a novel technique called Encryption technique based on ASCII values of Reflection Tree (E-ART). The binary tree supports the English language using ASCII, as shown in Figure 1. The tree nodes represent the ASCII character values, which range from 0 to 127. The E-ART algorithm uses a 128-bit symmetric key that is divided into a static and a dynamic part. The static part changes for every session, and the dynamic part changes for each character. Employing a dynamic key provides adequate protection against classical and modern cryptanalysis attacks. The key derivation process is explained in the next section and all the notations used are shown in Table 1.

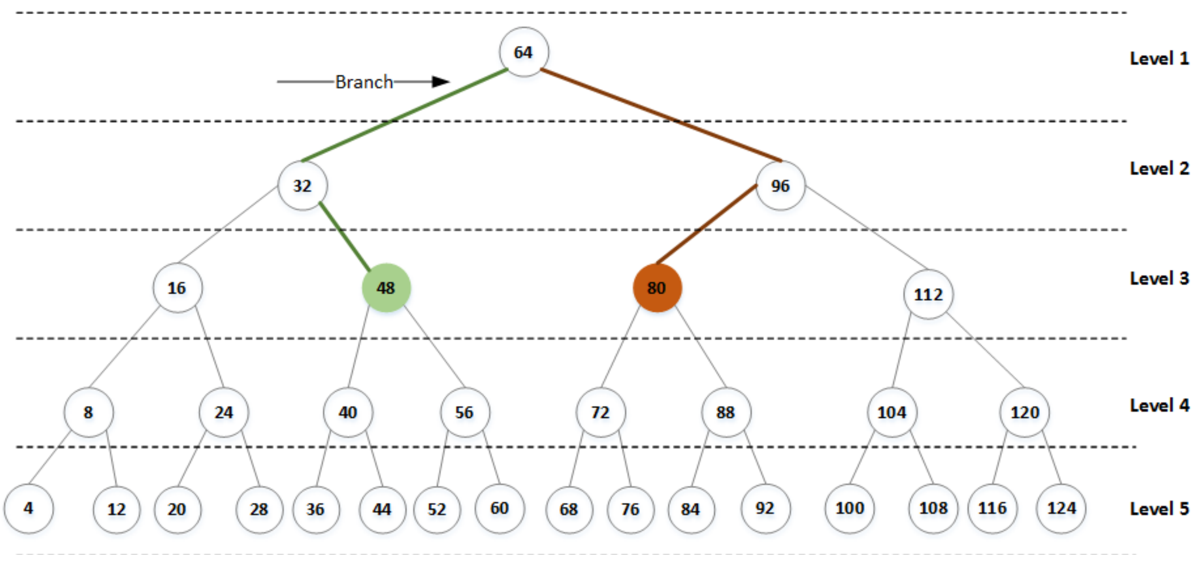

Figure 1. Binary tree of Encryption technique based on ASCII values of Reflection Tree (E-ART).

Table 1. Summary of notations.

Symbol	Definition
Len_{max}	Maximum ASCII character range (default is 127)
Val_{org}	Original ASCII character value
$Val_{initial\ ref}$	Reflected value of the original ASCII character value
$Offset_{const}$	Constant offset value used to avoid non-printable characters (32 by default)
Val_{ref}	Reflected value of the original ASCII character value after adding the offset
$Offset_{var}$	Variable offset computed based on the properties of the tree
R	The root node of the tree
N_L	Adjusted value of N so that it is within the range of maximum value
N_R	Reflection value of N_L
Pseudo	Pseudo-random number generated based on the character's position in the text
$Character_{value}$	Equivalent ASCII character for a given value

3.1. A. Key Derivation

In the key derivation process, the initial key is used to construct two offsets that are used during the data encryption process. The first one, called the variable offset, remains static during the encryption/decryption process, and the second one is a dynamic offset that changes with each character value. The inclusion of the dynamic offset ensures considerable randomness in the key, thus guaranteeing the security of the encryption process. The key derivation process is as follows:

3.1.1. Initial Key

This is a secret key shared between legal entities that can be renewed after each session or depending on the system's configuration. However, key management between legal entities is not the focus of this work. An initial key consists of two values, N and *Variance*, both of which are used to generate the offsets. N represents an integer value composed of 64 or 128 bits. This value will be the input to calculate the variable offset. *Variance* is used to compute the pseudo-random number value as part of the dynamic offset. The offsets used are as follows:

- Variable offset. It is calculated mathematically using the proposed tree properties. The left and right nodes are shown in Figure 1. It uses the N value derived from the initial key to calculate N_L and its reflection node N_R and then generate the value of $Offset_{var}$. This value is added to the initial reflection value according to Equations (3) and (4) to add more complexity and prevent cryptanalysis attacks that take advantage of one-to-one mapping. It is computed as follows:

$$
\begin{gathered}
N_L = N \; mod \; Len_{max} \\
N_R = (Len_{max} - N_L) + 1 \\
Offset_{var} = \begin{cases} R \times N_L \; mod \; N_R & \text{if } N_L < \text{Root} \\ R \times N_R \; mod \; N_L & \text{if } N_L > \text{Root.} \end{cases}
\end{gathered} \tag{1}
$$

- Dynamic offset. It is produced automatically using a pseudo-random number and the second part of the initial key (*Variance*). The pseudo-random generator uses each character's position in the text as a seed to generate a pseudo-random number of 64 or 128 bits. The pseudo-random number is then adjusted using the *Variance* value. This offset is added in the last step to produce the final encrypted characters and is changed for each character. This results in a high degree of robustness and resistance to known powerful attacks. The dynamic offset is calculated as follows:

$$\text{Dynamic offset } = (\text{Pseudo}) \text{ mod Variance.} \tag{2}$$

3.1.2. E-ART Structure

The main novelty of the E-ART algorithm is the use of the reflection property of a balanced binary tree data structure to enhance data search efficiency. Binary trees can be explained as follow. The root node has a value X. The left subtree of the main tree contains all values less than X, while the right subtree includes values greater than X. Thus, the binary tree has a search complexity of $O(log(n))$ when searching for a particular value.

An example of a binary tree based on E-ART is shown in Figure 1. E-ART has five levels. Node 64 is the root, with children 32 and 96. Node 32 has two children, as does node 96, and so on. The branch of the tree can be defined as the link between parent–child nodes. For instance, one of the branches is 64–32, and another branch is 88–92, marked with a red line in Figure 1.

Let us consider a text containing the character "P," whose ASCII code is 80. This character is on level 3 of the binary tree and is located in one right and one left branch (1R1L) of the root node. Its reflected character is at 1L1R, which is 48. On the ASCII table, 48 is the code for the character "0," so the initial reflected value of character "P" is 0. Thus, any character in the plaintext can be encoded using a reflection tree. As a result, a technique with higher search efficiency is achieved. For a balanced binary tree, as shown in Figure 1, the initial reflected value $Val_{initialref}$ can be computed from the original value Val_{org} as follows:

$$Val_{Initial\ ref} = (Len_{max} - Val_{org}) + 1 \tag{3}$$

However, some issues with the initial reflected value remain:

(1) The presence of special characters, such as space, carriage return, and other text formatting characters, which range from 0 to 32 in the ASCII table, is not addressed.
(2) It is vulnerable to cryptanalysis attacks, such as frequency analysis attacks [17,26], due to the one-to-one mapping of characters.

To tackle these issues, we propose adding multiple offsets to the initial reflected value. First, to avoid the appearance of non-printable ASCII characters, a constant offset value of 32 is added to the initial reflected value. For example, let us consider the character "H," whose ASCII code is 104 and initial reflected value based on Equation (1) is (127 − 104) + 1 = 24, which represents a non-printable character. The newly calculated reflected value after adding the constant offset is 24 + 32 = 56. Second, to prevent a cryptanalysis attack and add more complexity, we propose another offset, which we call variable offset $Offset_{var}$. It is computed based on the E-ART tree's root node R, left node N_L, and right node N_R values, as shown in Equation (1). The general equation for the reflected value is

$$Val_{Ref} = \begin{cases} (X \,\%\, Len_{max}) + Offset_{const} & , X > Len_{max} \\ X & , X \le Len_{max} \end{cases} \quad (4)$$

where $X = Val_{intial\ ref} + Offset_{var} + Offset_{const}$.

If the value X is less than the maximum character space Len_{max}, then it is considered the reflected value. If the reflected value is greater than Len_{max}, then it is computed in the range of $\{0, Len_{max}\}$, and the constant offset is added to avoid the appearance of non-printable characters. Figure 2 shows a flowchart of the encryption process.

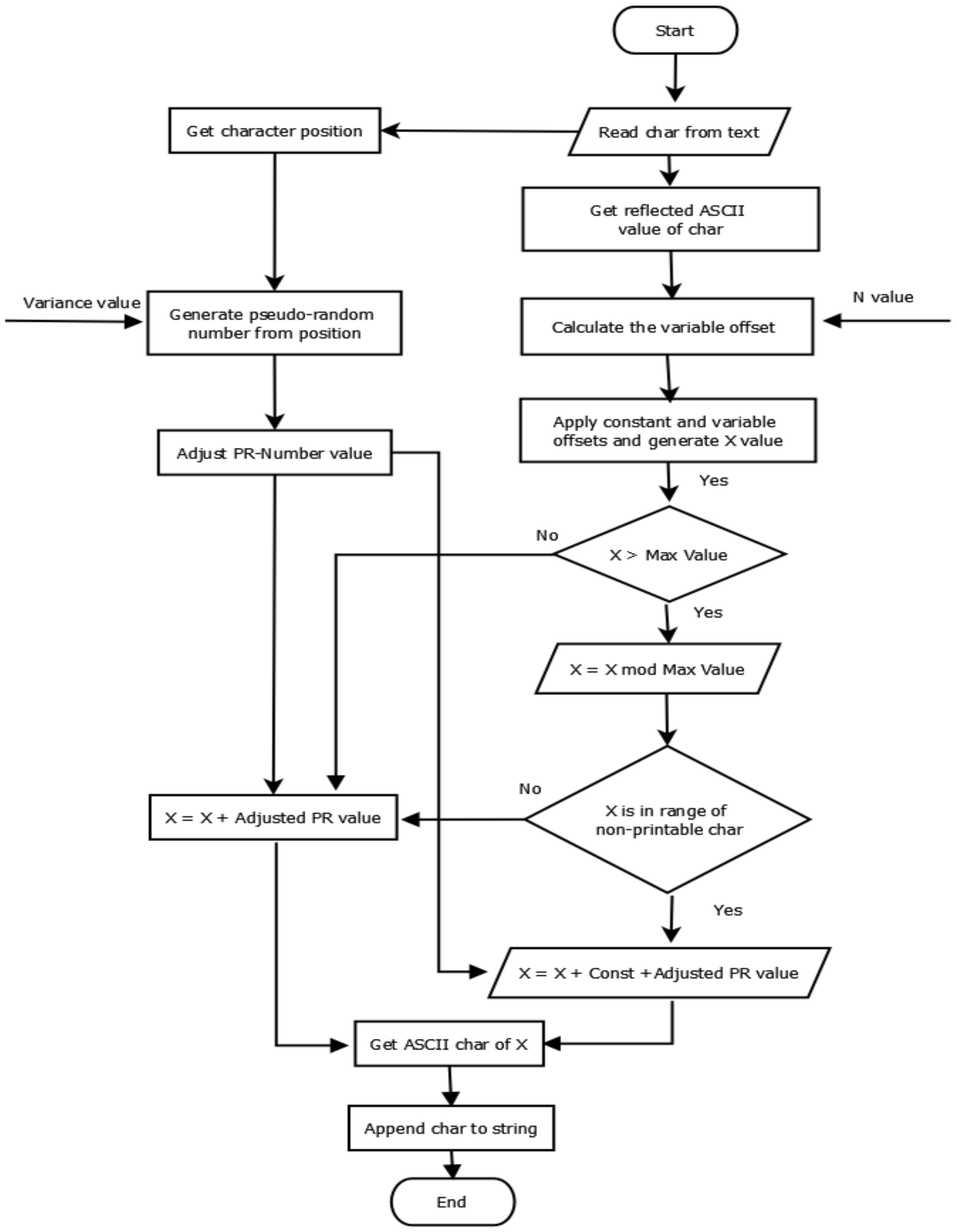
Start
Read char from text
Get character position
Get reflected ASCII value of char
Variance value
Generate pseudo-random number from position
Calculate the variable offset
N value
Adjust PR-Number value
Apply constant and variable offsets and generate X value
Yes
No
X > Max Value
Yes
X = X mod Max Value
No
X is in range of non-printable char
X = X + Adjusted PR value
Yes
X = X + Const + Adjusted PR value
Get ASCII char of X
Append char to string
End

Figure 2. Flowchart of the E-ART encryption process.

The second contribution of this work is the use of a dynamic offset based on the characters' position in the plaintext. It is generated automatically for each character using a pseudo-random generator and the character's position, as shown in Equation (2). This dynamic offset is produced as a function of the initial key, the character's position, and the pseudo-random number. This makes the relation between the plaintext and the ciphertext more random and complicated, which strengthens protection against cryptanalytic attacks since the encryption/decryption process becomes dynamic and different for each character. Combining Equations (3) and (4) yields (5) as follows:

$$Val_{Ref} = Val_{Ref} + dynamic\ offset \tag{5}$$

Pseudo codes for the E-ART algorithm are shown in Algorithm 1. We describe the main steps of Algorithm 1 as follows:

- Initially, the input textual data are stored in an array of characters (plaintext list).
- Each character in the list is converted into its corresponding ASCII values and stored in $Val_{org.}$
- The variable offset $Offset_{var}$ is generated using N, the first value of the initial key, and the properties of the tree—R, N_L, and N_R—as shown in Equation (1).
- For each character in the list, the initial reflected value $Val_{Initial\ ref}$ for each character is calculated using Equation (3).
- For each character in the list, the dynamic offset is generated by a pseudo-random generator using *Variance* on the second value of the initial key and characters' positions, as shown in Equation (2).
- Then, value X is generated by adding the variable offset $Offset_{var}$ and constant offsets $Offset_{const}$ to the initial reflected value using Equation (4).
- Value X changes based on the maximum length (Len_{max}) and non-printable character range. If the value of X is greater than Len_{max}, then apply the mod operation and then add $Offset_{const}$, as shown in Equation (4).
- Then, the dynamic offset is added to the value of X using Equation (5) to generate the final reflection value Val_{Ref}.
- Val_{Ref} is converted to the equivalent ASCII character to produce the encrypted character.
- Append the character to encrypted list.
- Once all characters in the plaintext are encrypted, the encrypted text file is generated.

Algorithm 1: E-ART Algorithm

```
Input: $R, Offset_{const}, Len_{max}, input\_text, N, Variance$
Output: Encrypted text
1: Initialization
2: input_list = Read all words from input file
3: Get the $Val_{org}$ for each character
4: Get the $Offset_{var}$ from Equation (1)
5: while all words in input_list are not iterated, do
6:     word = pop word from input_list
7:         for each character in word do
8:             Get $Val_{Initial\ ref}$ character from Equation (3)
9:             Get Dynamic offset from Equation (2) with
10:              Let X = $Val_{Initial\ ref} + Offset_{var} + Offset_{const}$
11:              if X is greater than $Len_{max}$
12:                          $Val_{Ref}$ [($X$ mod $Len_{max}$) + $Offset_{const}$ + Dynamic offset]
13:              else
14:                          $Val_{Ref}$ [$X$ + Dynamic offset]
15:              end if
16:              end for
17:              append $Character_{value}$ of $Val_{Ref}$ to EncryptedWord
18:                          append word or EncryptedWord to EncryptedList
19:      end while
20:      Write all values from EncryptedList to output document
```

The decryption process can be considered a reversible form of the encryption algorithm. We describe the main steps of the decryption process in Algorithm 2 as follows:

- Initially, the input data are stored in an array of characters (ciphertext list).
- Each character in the list is converted into its corresponding ASCII value and stored in Val_{org}.
- The variable offset is generated using N, the first value of the initial key, and the properties of the tree—R, N_L, and N_R—as shown in Equation (1).
- For each character in the list, the dynamic offset is regenerated by a pseudo-random generator using the same parameters, *Variance*, with the second value of the initial key and the characters' positions as shown in Equation (2).
- Value X is generated by subtracting the dynamic offset from Val_{org}.
- Then, we check: if subtraction of variable offset $Offset_{var}$ and constant offset $Offset_{const}$ from X is less than 0, then set *Quotient* to be equal to 1; otherwise, set *Quotient* value to be equal to 0.
- Generate the Val_{Ref} by multiplying Len_{max} and *Quotient* and then subtract X, variable offset $Offset_{var}$ and constant offset $Offset_{const}$.
- Generate decrypted value by subtracting Val_{Ref} from Len_{max} plus 1.
- Decrypted value is converted to the equivalent ASCII character to produce the decrypted character.
- Append the character to the decrypted list.
- Once all characters in the ciphertext are decrypted, the decrypted text file is generated.

Algorithm 2: Data Decryption Algorithm

```
Input: $R, Offset_{const}, Len_{max}$, Encrypted Text, N, Variance
Output: decrypted text
1: Initialization
2: input_list = Read all word from input file
3: Get the $Val_{org}$ or each character
4: Get the $Offset_{var}$ from Equation (1)
5: while all words in input_list are not iterated, do
6:   word = pop word from input_list
7:       for each character in word do
8:           Get Dynamic offsetrom Equation (2)
9:           Let $X = Val_{org} -$ Dynamic offset
10:           if $(X - Offset_{var} - Offset_{const} < 0$
11:               Quotient = 1
12:       else
13:               Quotient = 0
14:           $Val_{Ref}\ Len_{max}[(xuotient + X) - Offset_{var}\ Offset_{const}]$
15:                       decrypted_value = $(Len_{max-}Val_{Ref}) + 1$
16:                           end if
17:                           end for
18:               append $Character_{value}$ of decrypted_value to decryptedWord
19:                       append word or decryptedWord to decryptedList
20:   end while
21:   Write all values from decryptedList to output document
```

4. Experimental Evaluation

The evaluation of E-ART have been done in a similar way adopted in different recent and relevant studies of cryptographic algorithms [14,23,25,39]. To evaluate its performance and efficiency, the proposed algorithm was implemented on the Java platform using NetBeans 8.2 and compared to AES-128 and DES. All experiments were conducted on the same platform using a Windows-based machine equipped with 32 GB of memory and an Intel i7 3.4 GHz CPU. We performed NIST statistical tests to verify randomness.

4.1. A. Performance Analysis

For an encryption algorithm, processing time is the most important criterion after security, especially in large-sized and real-time applications, where heavy processing and long runtimes are not preferred. Thus, the execution time (encryption/decryption) and memory usage metrics were used in the performance evaluation. The proposed algorithm results were compared to two well-known symmetric encryption algorithms, AES-128 and DES, using file sizes ranging from 200 to 2000 KB. The results, displayed in Tables 2–4, clearly showed that the proposed algorithm outperformed AES and DES in terms of processing time with comparable memory usage for file sizes between 200 and 1000 KB and slightly higher for 2000 KB. We can thus conclude that the implementation of the reflection of the balanced tree along with the dynamic offset achieves high computational speed with minimal memory usage.

Table 2. Advanced encryption standard (AES) performance.

File Size (KB)	Encryption		Decryption	
	Processing Time (ms)	**Memory (MB)**	**Processing Time (ms)**	**Memory (MB)**
200	1433	17	1378	17
400	1956	21	1363	18
600	2118	27	1637	25
800	2335	30	1645	31
1000	2528	33	1995	35
2000	3616	55	1754	56

Table 3. Data encryption standard (DES) performance.

File Size (KB)	Encryption		Decryption	
	Processing Time (ms)	**Memory (MB)**	**Processing Time (ms)**	**Memory (MB)**
200	1838	18	1992	19
400	2067	22	2444	25
600	2190	27	2750	29
800	2575	31	3183	37
1000	3034	34	3658	41
2000	4537	55	5500	49

Table 4. E-ART Performance.

File Size (KB)	Encryption		Decryption	
	Processing Time (ms)	**Memory (MB)**	**Processing Time (ms)**	**Memory (MB)**
200	123	14	162	7
400	189	23	250	15
600	253	23	320	20
800	413	29	385	29
1000	479	32	460	36
2000	1854	65	775	68

4.2. Security Analysis

4.2.1. Avalanche Effect

The avalanche effect is a desirable feature of any cryptographic algorithm that tries to reflect the idea of high nonlinearity [40]. If a significant level is not demonstrated during an avalanche test, then the algorithm's randomization is inadequate, which can allow a cryptanalyst to make predictions about the plaintext only from the given ciphertext. For an algorithm to satisfy the avalanche criterion, a slight change in the input (flipping a single bit in either the plaintext or the key) produces a significant change in the output (at least half of the bits are flipped) [41]. The avalanche effect is measured using the hamming distance. The hamming distance between two texts of equal length is the number of positions at which corresponding characters are different. To measure the avalanche effect, we used 10 pair keys that differed only in one bit to encrypt 10 pair files with each encryption technique. Then, we calculated the hamming distance to obtain the number of bits that differed between each pair of files. The avalanche effect is calculated as

$$AvalancheEffect = \frac{HammingDistance}{TotalNoCharacters} \times 100 \tag{6}$$

Table 5 shows the average hamming distance and the avalanche effect for 10 pairs of files encrypted using E-ART, AES, and DES. The average avalanche effect of our proposed algorithm was 50.11%, compared to 49.2% for AES and 49.3% for DES. These results showed that the proposed encryption system satisfies the avalanche effect criterion and thus provides strong protection against differential cryptanalysis.

Table 5. Avalanche effect performance.

Technique	Hamming Distance	Avalanche Effect
E-ART	57,852	50.1%
AES	39,345	49.2%
DES	39,425	49.3%

4.2.2. Bit Independence Criterion

Webster and Tavares defined another feature called the Bit Independence Criterion (BIC) for the cryptographic algorithm [41,42]. A function satisfies the BIC if any input bit i is inverted in the plaintext or the key, then the output bits j and k in the ciphertext must change independently. To measure the degree of independence between the pairs j and k of avalanche variables, first, the avalanche variables are converted to binary representation. Then, the correlation coefficient between j and k is calculated. The result shows a correlation value of 0.1863, which indicates very low dependence between the two avalanche variables. Hence, the proposed algorithm satisfies the BIC.

4.2.3. Frequency Analysis

In cryptanalysis, frequency analysis is the study of the frequency of letters or symbols in ciphertext. The frequency can be represented in a histogram [43]. An attacker can use frequency analysis to obtain the key or the plaintext. This type of attack is called a statistical attack. To prevent such attacks, the plaintext and ciphertext histograms should not be statistically similar. Therefore, the ciphertext histogram should be relatively uniform. The generated histograms of the plaintext and its corresponding ciphertext are shown in Figure 3. It can be seen that the histogram of the encrypted text is quite uniformly distributed and differs significantly from that of the plaintext.

4.2.4. Randomness Verification

To evaluate the quality of our dynamic offset, we used National Institute of Standards and Technology (NIST) statistical tests [44] with 1000 sequences, where each series was 128 bits. The NIST Statistical Test Suite is a statistical package consisting of 15 tests designed to assess the degree of randomness for binary sequences produced by cryptographic random number generators. These tests compute the p-value to determine the strength of the evidence against the null hypothesis. In each test, the p-value is the probability that an ideal random number generator generates a sequence less arbitrary than the series tested according to the types of non-randomness evaluated by the test. The significance level α, typically ranging from 0.001 to 0.01, can be selected to be (0.01). If $p \geq \alpha$, then the null hypothesis is accepted, and the sequence appears to be random.

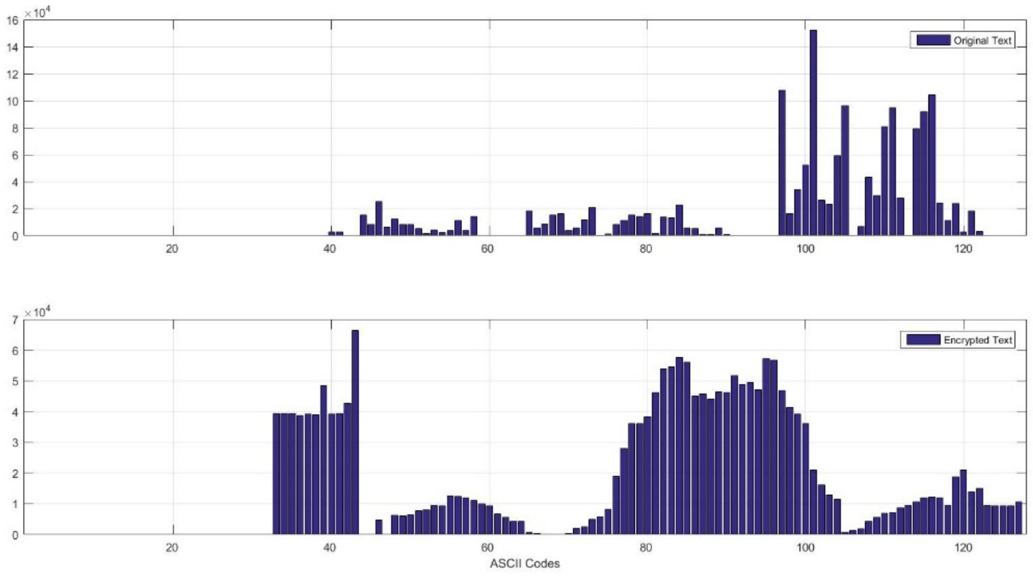

Figure 3. Histogram analysis.

For our experiment, we selected five tests to determine the randomness and non-uniformity of our dynamic key. The selection was based on the recommendation of input size stated by NIST user's guide, as we restricted our dynamic offset to 128 bits. The following tests are applicable to our key length:

- Frequency test
- Block frequency test
- Runs test
- Cumulative sums forward test
- Cumulative sums backward test

Using our proposed pseudo-random generator, we produced 1000 sequences. The results of the five statistical tests are presented in Table 6. A p-value of $\geq$0.01 in each test showed that our proposed algorithm can generate dynamic offsets that are truly random and infeasible to predict.

Table 6. Statistical test results.

Test	p-Value	Remarks
Frequency test	0.7399	Random
Block frequency test	0.7399	Random
Runs test	0.0668	Random
Cumulative sums forward test	0.1223	Random
Cumulative sums backward test	0.5341	Random

4.3. Security against Attacks

All of the experiments were carried out to prove that the proposed cipher is secure and efficient. In this section, a brief cryptanalysis discussion is presented to validate its security and to prove that it can resist various kinds of well-known attacks.

The proposed cipher includes a dynamic key structure that varies for each character. Therefore, no critical information can be captured from the collected encrypted messages since they are encrypted with different dynamic keys. The obtained ciphertext is also significantly different from the original message as shown in Figure 3. Thus, the adversary cannot establish any relationship among received encrypted messages. Consequently, the proposed scheme can be considered secure against chosen and known plain-text/cipher-text attacks.

To resist brute force attack, the key space should be sufficiently large. The size of the initial secret key is 128 bits and dynamic offset is 128 bits, which means 2^{128} possible initial secret keys, which is computationally infeasible. Furthermore, it is difficult for an adversary to get all the combination parameters (e.g., R, Pseudo, and *Variance*) used to generate the variable offsets and dynamic offsets. Thus, the cipher is computationally secure against brute-force attack.

Aside from the high random outcomes of the pseudo-random generator as measured by the NIST test, a high degree of randomness is ensured based on a BIC test by having a correlation coefficient close to zero, and the key sensitivity to any bit of the secret key is achieved according to an avalanche effect test. Consequently, the proposed scheme can be considered secure against statistical attacks.

5. Discussion

The main aim of this study was to design a cipher algorithm that can achieve a satisfactory level of security with shorter execution times and fewer resources. We used the properties of the binary search tree to reduce the computational complexity and the dynamic key technique to increase security.

The comparison of the performance of E-ART with that of well-known cipher schemas, AES and DES, showed that E-ART achieved the shortest running time with comparable memory consumption. This makes it a promising solution for big data, delay-sensitive, and real-time applications. For security analysis, the avalanche effect was calculated. The integer numbers N and *Variance* represent the initial value secret key that was used to generate the offsets for the encryption algorithm. Flipping of one single bit in these initial keys will change the offsets value and consequently change the algorithm sequence, and the ciphertext will widely vary. Our avalanche effect analysis showed that E-ART changed on average half of the bits in the ciphertext when a single bit was changed in the initial keys, demonstrating that it is sufficiently sensitive to any change in the key, making key-related attacks considerably more difficult to succeed. The BIC analysis performed on two avalanche variables showed a satisfactory bit independence criterion as the correlation value obtained was close to 0. Therefore, it is difficult to predict one bit from other bits and make the cryptanalysis difficult. Further, the size of the of the initial key can be set to 64 or 128 bits, whereas the size of the proposed pseudo-random can be set to 64 or 128 bits. These sizes are large enough to make brute force attacks unfeasible.

The performed NIST tests showed that the dynamic offset has a satisfactory level of randomness and uniformity. As the dynamic offset changes for every character, algebraic [45], linear, and differential attacks [46] are also considerably harder to succeed. Overall, our results show that the proposed cipher is a good candidate for lightweight modern text data encryption.

6. Conclusions

Existing symmetric encryption algorithms are based on static keys and multi-round functions to reach the required security levels. This entails a trade-off between security and performance. The aim of this work was to propose a simple and efficient algorithm that reduces the required latency and resources without compromising security. The core novelties of the proposed algorithm are the use of the balanced binary search tree's properties along with ASCII values to reduce the computational complexity and the use of the dynamic key technique to strengthen security. To generate a dynamic key that that varies for each character, we used a pseudo random number generator that uses each character's position.

The performance of the algorithm was compared to that of AES and DES. The results showed that E-ART had the shortest running time and comparable memory usage for file sizes between 200 and 1000 KB and slightly higher for 2000 KB, as shown in Tables 2–4. For security analysis, the avalanche effect analysis showed that E-ART changed on average half of the bits in the ciphertext when a single bit was changed in the initial keys, demonstrating that it is sufficiently sensitive to any change in the key and satisfies the avalanche effect criterion. The BIC analysis performed on two avalanche variables showed a satisfactory bit independence criterion, as the correlation value obtained shows a correlation value of 0.1863. The histogram of the encrypted text is quite uniformly distributed and differs significantly from that of the plaintext, as shown in Figure 3.

The randomness of the dynamic offset was assessed using NIST statistical tests. The results showed that it is truly random and infeasible to predict. We therefore conclude that the proposed algorithm is suitable for securing big and real-time data.

Author Contributions: Conceptualization, B.A. and N.B.; methodology, B.A.; software, B.A.; validation, B.A., N.B. and M.W.; investigation, B.A.; resources, B.A.; data curation, B.A.; writing–original draft preparation, B.A.; writing–review and editing, M.W.; visualization, B.A.; supervision, N.B. and M.W. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: No new data were created or analyzed in this study. Data sharing is not applicable to this article.

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

References

1. Tankard, C. Encryption as the cornerstone of big data security. *Netw. Secur.* **2017**, *2017*, 5–7. [CrossRef]
2. Alabdullah, B. Rise of Big Data; Issues and Challenges. In Proceedings of the 2018 21st Saudi Computer Society National Computer Conference, Riyadh, Saudi Arabia, 25–26 April 2018; pp. 1–6.
3. European Union. European Union Regulation 2016/679. *Off. J. Eur. Communities* **2014**, *2014*, 1–88.
4. Editors, S.; Editors, A. *Daemen. Springer—The Design of Rijndael.pdf*; Springer: Berlin/Heidelberg, Germany, 2002; ISBN 9783642076466.
5. Abood, O.G.; Guirguis, S.K. A Survey on Cryptography Algorithms. *Int. J. Sci. Res. Publ.* **2018**, *8*, 8. [CrossRef]
6. Ostrovsky, R.; Sahai, A.; Waters, B. Attribute-based encryption with non-monotonic access structures. In Proceedings of the 14th ACM Conference on Embedded Network Sensor Systems CD-ROM, Alexandria, VA, USA, 29 October–2 November 2007; pp. 195–203.
7. Nie, T.; Zhang, T. A study of DES and blowfish encryption algorithm. In Proceedings of the TENCON 2009—2009 IEEE Region 10 Conference, Singapore, 23–26 January 2009; pp. 1–4.
8. Çavuşoğlu, Ü.; Kaçar, S.; Zengin, A.; Pehlivan, I. A novel hybrid encryption algorithm based on chaos and S-AES algorithm. *Nonlinear Dyn.* **2018**, *92*, 1745–1759. [CrossRef]
9. Gai, K.; Qiu, M.; Zhao, H.; Xiong, J. Privacy-Aware Adaptive Data Encryption Strategy of Big Data in Cloud Computing. In Proceedings of the 3rd IEEE International Conference on Cyber Security and Cloud Computing, Beijing, China, 25–27 June 2016; pp. 273–278.
10. Forouzan, B.A. *Cryptography and Network Security*; McGraw-Hill, Inc.: New Delhi, India, 2007; Volume 1025, ISBN 9780131873162.
11. Al-Kazaz, N.R.; Teahan, W.J. An automatic cryptanalysis of Arabic transposition ciphers using compression. *Int. J. Adv. Comput. Sci. Appl.* **2018**, *9*, 738–745. [CrossRef]
12. Agustini, S.; Rahmawati, W.M.; Kurniawan, M. Modified Vegenere Cipher to Enhance Data Security Using Monoalphabetic Cipher. *Int. J. Artif. Intell. Robot.* **2019**, *1*, 25. [CrossRef]
13. Amalia; Budiman, M.A.; Sitepu, R. File text security using Hybrid Cryptosystem with Playfair Cipher Algorithm and Knapsack Naccache-Stern Algorithm. *J. Phys. Conf. Ser.* **2018**, *978*, 012114. [CrossRef]
14. Marzan, R.M.; Sison, A.M.; Medina, R.P. An enhanced key security of Playfair cipher algorithm. *Int. J. Adv. Trends Comput. Sci. Eng.* **2019**, *8*, 1248–1253. [CrossRef]
15. Aung, T.M.; Hla, N.N. A Complex Polyalphabetic Cipher Technique Myanmar Polyalphabetic Cipher. In Proceedings of the 2019 International Conference on Computer Communication and Informatics, Coimbatore, Tamil Nadu, India, 23–25 January 2019.
16. Elmogy, A.; Bouteraa, Y.; Alshabanat, R.; Alghaslan, W. A New Cryptography Algorithm Based on ASCII Code. In Proceedings of the 2019 19th International Conference on Sciences and Techniques of Automatic Control and Computer Engineering (STA), Sousse, Tunisia, 24–26 March 2019; pp. 626–631.
17. Yadav, N.; Kapoor, R.K.; Rizvi, M.A. A Novel symmetric key cryptography using dynamic matrix approach. *Adv. Intell. Syst. Comput.* **2016**, *439*, 51–60.
18. Biryukov, A. *Encyclopedia of Cryptography and Security*; van Tilborg, H.C.A., Ed.; Springer: Boston, MA, USA, 2005; ISBN 978-0-387-23483-0.
19. Shannon, C.E.; Weaver, W. *The Mathematical Theory of Communication*; ACM: New York, NY, USA, 1964; Volume 8, ISBN 9780252725487.

20. Aleisa, N. A comparison of the 3DES and AES encryption standards. *Int. J. Secur. Appl.* **2015**, *9*, 241–246. [CrossRef]
21. Manku, S.; Vasanth, K. Blowfish encryption algorithm for information security. *ARPN J. Eng. Appl. Sci.* **2015**, *10*, 4717–4719.
22. Lamba, C.S. Design and Analysis of Stream Cipher for Network Security. In Proceedings of the 2010 Second International Conference on Communication Software and Networks, Singapore, 26–28 February 2010; Volume 76, pp. 526–567.
23. Rajesh, S.; Paul, V.; Menon, V.G.; Khosravi, M.R. A secure and efficient lightweight symmetric encryption scheme for transfer of text files between embedded IoT devices. *Symmetry* **2019**, *11*, 293. [CrossRef]
24. Hernández-Ramos, J.L.; Pérez, S.; Hennebert, C.; Bernabé, J.B.; Denis, B.; Macabies, A.; Skarmeta, A.F. Protecting personal data in IoT platform scenarios through encryption-based selective disclosure. *Comput. Commun.* **2018**, *130*, 20–37. [CrossRef]
25. Aljawarneh, S.; Yassein, M.B.; Talafha, W.A. A multithreaded programming approach for multimedia big data: Encryption system. *Multimed. Tools Appl.* **2018**, *77*, 10997–11016. [CrossRef]
26. Dawood, O.A.; Sagheer, A.M.; Al-Rawi, S.S. Design large symmetric algorithm for securing big data. In Proceedings of the 2018 11th International Conference on Developments in eSystems Engineering (DeSE), Cambridge, UK, 2–5 September 2018; pp. 123–128.
27. Al-Omari, A.H. Lightweight Dynamic Crypto Algorithm for Next Internet Generation. *Eng. Technol. Appl. Sci. Res.* **2019**, *9*, 4203–4208. [CrossRef]
28. Ngo, H.H.; Wu, X.; Le, P.D.; Wilson, C.; Srinivasan, B. Dynamic key cryptography and applications. *Int. J. Netw. Secur.* **2010**, *10*, 161–174.
29. Chunka, C.; Goswami, R.S.; Banerjee, S. An efficient mechanism to generate dynamic keys based on genetic algorithm. *Secur. Priv.* **2018**, e37. [CrossRef]
30. Noura, H.N.; Reem, M.; Mohammad, M.; Ali, C. Lightweight and secure cipher scheme for multi-homed systems. *Wirel. Netw.* **2020**, 1–18. [CrossRef]
31. Noura, H.N.; Chehab, A.; Couturier, R. Efficient & secure cipher scheme with dynamic key-dependent mode of operation. *Signal Process. Image Commun.* **2019**, *78*, 448–464.
32. Noura, H.; Chehab, A.; Couturier, R. Lightweight Dynamic Key-Dependent and Flexible Cipher Scheme for IoT Devices. In Proceedings of the 2019 IEEE Wireless Communications and Networking Conference (WCNC), Marrakesh, Morocco, 15–18 April 2019; pp. 1–8.
33. Jallouli, O.; Assad, S.E.; Chetto, M.; Lozi, R. Design and analysis of two stream ciphers based on chaotic coupling and multiplexing techniques. *Multimed. Tools Appl.* **2018**, *77*, 13391–13417. [CrossRef]
34. Wen, W.; Zhang, Y.; Su, M.; Zhang, R.; Chen, J.X.; Li, M. Differential attack on a hyper-chaos-based image cryptosystem with a classic bi-modular architecture. *Nonlinear Dyn.* **2017**, *87*, 383–390. [CrossRef]
35. Teh, J.S.; Alawida, M.; Sii, Y.C. Implementation and practical problems of chaos-based cryptography revisited. *J. Inf. Secur. Appl.* **2020**, *50*, 102421. [CrossRef]
36. Ding, L.; Liu, C.; Zhang, Y.; Ding, Q. A new lightweight stream cipher based on chaos. *Symmetry* **2019**, *11*, 853.
37. Arab, A.; Rostami, M.J.; Ghavami, B. An image encryption method based on chaos system and AES algorithm. *J. Supercomput.* **2019**, *75*, 6663–6682. [CrossRef]
38. Chai, X.; Fu, X.; Gan, Z.; Lu, Y.; Chen, Y. A color image cryptosystem based on dynamic DNA encryption and chaos. *Signal Process.* **2019**, *155*, 44–62. [CrossRef]
39. Dawood, O.A.; Khalaf, M.; Mohammed, F.M.; Almulla, H.K. *Design a Compact Non-linear S-Box with Multiple-Affine Transformations*; Springer: Berlin/Heidelberg, Germany, 2020; Volume 1, ISBN 9783030387525.
40. Zheng, Y.; Zhang, X.M. On relationships among avalanche, nonlinearity, and correlation immunity. *Lect. Notes Comput. Sci.* **2000**, *1976*, 470–482.
41. Lee, J.; Sultana, N.; Yi, F.; Moon, I. Avalanche and bit independence properties of photon-counting double random phase encoding in gyrator domain. *Curr. Opt. Photonics* **2018**, *2*, 368–377.
42. Webster, A.F.; Stafford, E.T. On the design of S-boxes. In *Conference on the Theory and Application of Cryptographic Techniques*; Springer: Berlin/Heidelberg, Germany, 1985; pp. 523–534.
43. Omran, S.S.; Al-Khalid, A.S.; Al-Saady, D.M. A cryptanalytic attack on Vigenère cipher using genetic algorithm. In Proceedings of the 2011 IEEE Conference on Open Systems, Langkawi, Malaysia, 25–28 September 2011; pp. 59–64.
44. Rukhin, A.; Soto, J.; Nechvatal, J. A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications. *Nist Spec. Publ.* **2010**, *22*. [CrossRef]
45. Roman'kov, V. Two general schemes of algebraic cryptography. *Groups Complex. Cryptol.* **2018**, *10*, 83–98. [CrossRef]
46. Blondeau, C.; Leander, G.; Nyberg, K. Differential-Linear Cryptanalysis Revisited. *J. Cryptol.* **2017**, *30*, 859–888. [CrossRef]