Security analysis of an audio data encryption scheme based on key chaining and DNA encoding

Imad El Hanouti · Hakim El Fadili

Received: 30 September 2019 / Revised: 14 August 2020 / Accepted: 23 October 2020
Published online: 8 January 2021
© Springer Science+Business Media, LLC, part of Springer Nature 2021, corrected publication 2021

Abstract
Fairly recently, a new audio encryption scheme has been proposed. The cryptosystem is based on a substitution-permutation algorithm using DNA encoding. The key-generation of this proposed scheme is based on a key chaining mode, that generates a new key block for every plain block using the chaotic logistic map. After several statistical tests, handled by the authors of the scheme, they claimed that their cryptosystem is robust. In this paper, we scrutinize the cryptosystem from a cryptanalytic perspective, and we handle several security attacks to evaluate the immunity of the system, and to assess its possible adoption in real-world applications. We demonstrate two successful conventional attacks on the scheme, which are: the chosen ciphertext and chosen-plaintext attacks. The cryptosystem’s shuffling process design is scrutinized as well, and a cycle attack is described using the drawn results. Lessons learned from this cryptanalytic paper, are then outlined in order to be considered in further designs and proposals.

Keywords Cryptanalysis · Audio data security · Chosen-plaintext attack · Cycle attack · DNA encoding · Logistic map

1 Introduction
To convey modern applications’ requirements, and due to some intrinsic characteristics of multimedia data (e.g., videos, audios…) [16], including bulk data size, a strong correlation between uncompressed neighbor data-units, high presence in real-time systems and the necessity for fast processing and transmission [9, 23]. Multimedia data processing and transmission have become a challenging topic of research [14]. Information security and data privacy constitute a crucial requirement in modern network applications and data transmission. Several cyber-attacks
threaten public networks due to the high and random accessibility to their internal structure. Multimedia data security, as any data type security, is one of the most important processes to be ensured before establishing any information transfer over public networks. The necessity for fast multimedia processing applications that respond to real-time requirements makes multimedia application development a hot topic in the recent research arena [5, 8, 12, 19, 25].

Cryptography and cryptographic protocols [27] are one underlying process for any data type security; cryptography is the art of designing highly secure systems. It aims to convert original data from a readable state to an unrecognizable form, except for entities that own the secret key. Conventional cryptosystems have been shown, in some related works, to be insufficient to respond to all mentioned requirements [17]. That’s why designing new cryptosystems with high security and fast processing capabilities has become a primordial issue in the recent literature [6, 11, 31, 34]. Two of the most relevant requirements in modern cryptography that have to be met for designing robust symmetric cryptosystems are confusion and diffusion, depicted by C. Shannon in 1949 [32]. Confusion refers to making the relationship between the key and the ciphertext as complex as possible while diffusion refers to the property that the redundancy in the statistics of the plaintext is dissipated in the statistics of the ciphertext. It has been observed that chaos theory exhibits similar proprieties as confusion and diffusion in cryptography theory [2].

Chaos theory is the branch of mathematics studying the strange behaviors of dynamic systems. Chaotic attractors have two main characteristics: sensitivity to initial conditions (i.e., small changes in initial conditions leads to significant changes in the orbit of the strange attractor), and ergodicity [2]. Chaos theory has been widely used in designing new cryptosystems [5, 8, 11, 19, 25, 26, 31, 34]. Nevertheless, every proposal’s security level is still in doubt until carrying out a full cryptanalytic study to evaluate the cryptosystem’s immunity, and that is the main role of cryptanalysis [7, 21]. Some of the proposals have already been cryptanalyzed [10, 22, 24, 36]. Besides, many recommendations and guidelines have been proposed to assess some basic level of security for the upcoming new works [2, 29].

From multimedia to big data, the need for complex calculations and large storage space pushes researchers to look for other computing implementations [1]. DNA computing is a rising filed of new computing generation, making use of biochemical reactions techniques to perform computations across a medium of biological DNA molecules [30]. DNA computing has three main advantages over traditional silicon computing, which are inherited basically from some intrinsic characteristics of DNA molecules: a high degree of parallelism, low energy consumption and a vast amount of information storage [38].

In [28], a new proposed cryptosystem dedicated especially for audio data is designed based on DNA encoding and the chaotic logistic map. The new proposed system is a block cipher that operates on blocks of 32-byte size. It is based on a key chaining algorithm that generates different new key block for each plain block. The scheme’s design is based on two main operations: substitution operation using a DNA XOR (i.e., exclusive-or) operation, followed by a shuffling process to permute bytes within each block. The experimental and statistical tests done by the authors of the scheme shows good statistical performances [28]. However, in this paper we will scrutinize this cryptosystem, we will evaluate its immunity against conventional attacks, and we will draw some lessons and recommendations, based on this cryptanalytic paper, to be considered for further proposals.

This paper’s organization is as follows: the second section will describe the audio data cryptosystem under study. Section 3 outlines some vulnerabilities and security flaws found in the scheme. In Section 4, we will illustrate our proposed chosen-plaintext attack as well as a
small discussion about a chosen-ciphertext attack. In Section 5, we will illustrate our so-called cycle attack which is a customized version of chosen-plaintext attack. In Section 6, we will depict some lessons learned from this cryptanalytic paper to avoid designing systems with the same outlined vulnerabilities in further proposals. Finally, in Section 7, we will discuss our results and conclude the paper.

2 Description of the audio data encryption scheme

The scheme under study [28], is labeled in this paper as DECS-AU scheme (DNA Encoding and Channel Shuffling Audio Encryption scheme). It is based on the logistic map [18] and DNA encoding to handle different operations of encryption and decryption. The DECS-AU is a block cipher system that operates on blocks of 32-bytes. In this section, we equivalently describe all the process of the encryption operation as detailed in [28], followed by a tiny description of the decryption process. For more details about the scheme’s full description, one can refer to the paper in [28]. DECS-AU could be described in three basic steps: key scheduling and chaining, substitution operation and shuffling operation.

2.1 Key scheduling and chaining

Every single plain block\(^1\) \(b_i\) of 32-bytes is encrypted with a corresponding key \(K_i\) derived basically from the previous key block \(K_{i-1}\) using the chaotic logistic map depicted in Eq. (2). The original 32-bytes key (i.e., the one chosen by the user) is denoted in this paper as \(MK\) (master key) and used to generate the first key block \(K_1\) for the encryption of the first plain block \(b_1\). As mentioned in [28], consecutive blocks use definitely distinct keys. The key \(K_i\) is generated using the previous key \(K_{i-1}\) as follows:

\[
K_i(j) = \left\lfloor x_j \times \left( K_{i-1}^2(j) + K_{i-1}(j-1) \right) \right\rfloor \mod 256, \quad j = 1, 2, ..., 32
\]  

(1)

Where \(K_i(j)\) is the \(j\)-th byte of the key block \(K_i\), \(\mod\) denotes modulo operation, the floor function \(\lfloor \cdot \rfloor\) calculates the greatest integer less than or equal (.), we define \(K_{i-1}(0)\) as \(K_{i-1}(0) = K_{i-1}(1)\), and we define \(K_0\) as \(K_0 = MK\), finally, \(x_j\) is defined using the logistic map equation as:

\[
x_j = 3.99 \times x_{j-1} \times (1-x_{j-1})
\]  

(2)

Where \(x_0\) is calculated by the following equation:

\[
x_0 = \frac{\sum_{j=1}^{32} MK(j) \times (j+1)}{2^{21}}
\]  

(3)

remind that \(MK\) is the master key (as discussed above), and \(MK(j)\) is the \(j\)-th byte of the master key.

That is, the initial condition \(x_0\) is calculated first from the master key \(MK\), and then the logistic map with a fixed parameter \(\lambda = 3.99\) is iterated according to the position of the key element. The last operation, in the key scheduling process, is to update the generated values according to the following updating operation:

\(^1\) Some notations from the original paper are modified without affecting its main meaning.
\[ K_i(j) = \begin{cases} 
K_i(j), & \text{if } K_i(j) \neq K_{i-1}(j) \\
\lfloor x_j \times 256 \rfloor, & \text{if } K_i(j) = K_{i-1}(j) 
\end{cases} \]  

(4)

This updating operation ensures that two successive key blocks are byte-wise distinct. This *key chaining* ensures a good diffusion of the master key \( MK \) along all key blocks.

### 2.2 Substitution operation

Substitution operation aims to confuse the plaintext\(^2\) using the key generated at the key scheduling stage. Here, the same process for one block is done for every block \( b_i \) of a plaintext, denoted as \( P \). The substitution operation could be described in three steps:

1. **DNA encoding operation:** Each byte from the block \( b_i \) denoted as \( b_i(j) \) (i.e., the \( j \)-th byte of the plain block \( b_i \)) is encoded into its DNA sequence according to a specific rule \( r_j \); this rule is calculated as follows:

\[
r_j = \lfloor x_j \times (K_i^2(j) + K_i(j-1)) \rfloor \mod 8
\]

(5)

so, every plain byte has its own calculated rule, which depends on its position and to the key block as well as to the logistic map iterations.

The encoding map used in [28] is depicted in Table 1. The same process is considered to encode the entire key \( K_i \) to its corresponding DNA nucleotide sequence. Note that, according to [28], the same rule is used to encode the same \( j \)-th byte from both \( b_i \) and \( K_i \). We denote the encoding function that encodes a single byte \( d \) according to the rule \( r_j \) as:

\[
DNAe_{r_j}(d) = N_4N_3N_2N_1; \quad N_i \in \{A, T, G, C\}
\]

(6)

in other words, Eq. (6) transforms any single byte-value to its equivalent quadruplet of nucleotides.

2. **DNA XOR operation:** After the encoding of bytes using DNA encoding rules, DNA XOR operation is carried out between the DNA encoded plain block and the DNA encoded key block. In [28], the definition of DNA XOR operation is based on the binary XOR operation (bitwise EX-OR or XOR operation). For any given rule, the DNA code for a DNA XOR operation is defined as the DNA encoded bitwise-XOR operation of the two operands in their binary form. Table 2 depicts the DNA XOR operation according to the rule \( r_j = 1 \).

3. **DNA decoding operation:** After calculating the DNA XOR operation between key entities and block entities, the results are decoded back to the binary form according to the same rule used for the encoding of each byte. The decoding function according to the rule \( r_j \) decodes a

\[^2\text{The two terms “plaintext” and “plain blocks” are used alternatively in this paper, the same for “keys” with “key blocks”, and for “ciphertext” with “cipher blocks.”}\]
quadruplet of nucleotides back to a single binary byte \( d \), and it is denoted as:

\[
DNAd_{rj}(N_4N_3N_2N_1) = d, \quad N_i \in \{A, T, G, C\} \tag{7}
\]

That is, Eq. (7) transforms back each quadruplet of nucleotides to its equivalent decimal value (a single byte-value).

The overall substitution process could be formulated mathematically as:

\[
S_i(j) = DNA_{d_{ij}}(DNA_{e_{ij}}(b_i(j)) \oplus DNA_{e_{ij}}(K_i(j))), \quad j = 1, \ldots, 32 \tag{8}
\]

Where \( S_i(j) \) is the \( j \)th byte of the \( i \)th substituted block, the substituted block is the intermediate block resulting from the application of only the described substitution process. \( DNA_{d_{ij}} \) and \( DNA_{e_{ij}} \) are previously defined in Eqs. (6) and (7) respectively, and \( \oplus \) denotes DNA XOR operation.

### 2.3 Shuffling operation

The last operation, in the encryption process, is the bytes shuffling within each substituted block \( S_i \). The shuffling algorithm calculates shuffling indexes for each substituted block, and it depends on the corresponding key block. Shuffling indexes could be seen as a permutation map, so within this paper, shuffling indexes and permutation map are equivalent terms. The calculation of the shuffling indexes is depicted in Fig. 1 to get acquainted with, but, for more details about this process, we suggest to refer to Algorithm 1 in [28]. Odd indexes are stored in one vector array, called the right channel, and denoted as \( Rc = \{Rc(j) \in 2\mathbb{N} + 1 \}_{j=1}^{16} \), and even indexes are stored in another vector array called the left channel \( Lc = \{Lc(j) \in 2\mathbb{N} \}_{j=1}^{16} \). And every byte \( S_i(Lc(j)) \) is swapped with its corresponding byte \( S_i(Rc(j)) \). The algorithm used to

### Table 1 DNA encoding and decoding according to each rule, as used in [28]

| Rule \( r_j \) | A | T | G | C |
|---------------|---|---|---|---|
| \( r_j = 1 \)  | 00 | 11 | 10 | 01 |
| \( r_j = 2 \)  | 00 | 11 | 01 | 10 |
| \( r_j = 3 \)  | 11 | 00 | 10 | 01 |
| \( r_j = 4 \)  | 11 | 00 | 01 | 10 |
| \( r_j = 5 \)  | 10 | 01 | 00 | 11 |
| \( r_j = 6 \)  | 10 | 01 | 11 | 00 |
| \( r_j = 7 \)  | 01 | 10 | 00 | 11 |
| \( r_j = 8 \)  | 01 | 10 | 11 | 00 |

### Table 2 DNA XOR operation according to rule 1 [28]

| DNA XOR \( \oplus \) | A | T | G | C |
|-----------------------|---|---|---|---|
| A                     | A | T | G | C |
| T                     | T | A | C | G |
| G                     | G | C | A | T |
| C                     | C | G | T | A |
generate the two left and right channels is depicted in Fig. 1. The shuffling operation is mathematically equivalent to a permutation:

\[ C_i(j) = S_i(j^*) \]

where \( C_i(j) \) is the \( j \)-th byte of the \( i \)-th cipher block, and \( j^* \) is defined as:

\[ j^* = f(j) \]

where \( f \) is a bijective function (i.e., a one-to-one correspondence) from the set \( M \) to \( M \) such that \( M = R_c \_ L_c = \{ p_i \}_{i=0}^{31} \) where \( A \_ B \) denotes the concatenation of the set \( A \) and the set \( B \). \( f \) is defined as:

\[ f(p_i) = p_{i+16 \mod 32}, \quad p_i \in M \]

in other words, we swap each couple of two bytes from the substituted block to get the cipher block. This swapping is based on the shuffling indexes as calculated and depicted in Fig. 1. The byte in the position \( p_i \) is swapped with the byte in the position \( p_{i+16 \mod 32} \) as denoted in Eq. (11).

According to Kerchhoff’s principle [20], the underlying algorithm and all the design of the cryptographic system should be known to the public, and the secrecy of the system must rely only on the secret key. The overall process of the described encryption scheme could be formulated mathematically as:

\[ C_i(j) = D N A d_{ij^*}(D N A e_{ij^*}(b_i(j^*))) \_ D N A e_{ij^*}(K_i(j^*)), \quad j = 1, \ldots, 32 \]

where \( C_i \) is the \( i \)-th cipher block, \( b_i \) is the \( i \)-th plain block, \( K_i \) is the \( i \)-th key block, and \( j^* \) is defined in Eq. (10).
The decryption process is done by taking encryption steps in reverse order. Firstly, using the same key chaining process to de-shuffle the cipher block, and then de-mask it according to the same substitution operation.

3 Security flaws in the DECS-AU scheme

In this section, we outline some of the critical observed vulnerabilities in the DECS-AU scheme. Some of the discovered vulnerabilities directly affect the overall system’s security, while others are considered as weaknesses in the design. Therefore, they could be exploited to enhance some attacks or partially reveal some secret parts of data or keys. Our observed security flaws are described as follows:

1. **A bad confusion/diffusion**: this results from the fact that, in key chaining operation, only the previous key is considered, neither the plain blocks nor the cipher blocks are related to the key generation and chaining process. This is a crucial vulnerability in the system, thus, knowing only the first key block will immediately allow us to recover any desired key block by applying the known algorithm of the key scheduling and chaining as described in the previous section. This vulnerability opens the door to many cryptanalytic attacks (e.g., chosen-plaintext attack and differential attack (Section 4.1), chosen-ciphertext attack (Section 4.2), known-plaintext attack …). From this fact, we depict the following Proposition:

Proposition 1: Since the key blocks in the DECS-AU are related only to the previous key block, then, knowing the first key-block $K_1$ is sufficient to recover all the keys $K_i$ used for the encryption of any plaintext (under the same master key $MK$).

According to this last Proposition, we will focus, in our attacks, on recovering only the first key block. Thus, all our attacks demonstration will be focused only on recovering the first key-block.

2. **No security effect is added by the DNA encoding**: The DNA computing is a growing field of computing where the medium of operations is biological DNA molecules which adds more benefits to the world of computing (due to some intrinsic molecule’s characteristics) such that high degree of parallelism, low energy consumption and vast storage space [38]. In recent years, several proposals have been published to implement some computing operations using DNA strands such as making logic gates [13, 35], and implementing arithmetic operations [4, 15]. However, in the DECS-AU scheme, the methodology adopted to encode binary data and perform DNA XOR operation is, by definition, based on the conventional binary XOR operation. Thus, the adopted DNA operations are no more than just an encoding/decoding operation. The definition of the DNA XOR operation as explicitly depicted in [28] could be equivalently formulated for any given rule $r_j$ as:

$$DNAd_j(DNAe_j(d_1) ⊕ DNAe_j(d_2)) = d_1 ⊕ d_2 \quad (13)$$

where $d_1$ and $d_2$ are two bytes and $⊕$ denotes the bitwise EX-OR operation.

From this last equation, we can say that the DNA XOR operation in the DECS-AU is equivalent to a simple bitwise EX-OR operation in the binary format. That is, this operation is
not based on DNA reactions and properties, but only on encoding/decoding of binary data to DNA symbols, and no security effect is envisaged to be added by this mean. This fact will allow us to mount several successful attacks, as we will see later in Sections 4 and 5.

3 **Period-2 shuffling map**: shuffling (swapping) an element twice, regains its original position. The shuffling operation is a permutation process that shuffles bytes inside each substituted block $S_i$. Generally, the total number of possible permutations for a block of 32 distinct bytes is $n = 32! \approx 2^{118}$ permutations, and the range (or the period) of a permutation, which is the smallest integer $R$ that if the permutation is applied $R$ times, the data will regain its original form (no permutation occurred), should be as large as possible and close to $n$. The algorithm (shown in Fig. 1) used by authors of the DECS-AU to generate permutation maps is very weak, and all permutations generated by the algorithm are of range $R = 2$. This critical vulnerability makes all the permutation maps (that could be considered as permutation keys) weak keys and makes the overall system weak against cycle attack (to be detailed more in Section 5).

**Proposition 2**: the range $R$ (or the period) of all permutation maps (shuffling maps) in the DECS-AU is $R = 2$. That is, considering Eq. (11), for any $p_n \in M$, $f(f(p_n)) = p_n$. In other words, for any block $b_i$, the application of only the shuffling process twice will make no effect on the block $b_i$.

**Proof:**

Let two elements $p_n, p_m \in M$, such that $f(p_n) = p_m$, we have to prove that $f(f(p_n)) = p_n$.

From Eq. (11) we have:

$$f(f(p_n)) = f(p_m) = p_{m+16 \mod 32}$$  \hspace{1cm} (14)

And we have:

$$m = n + 16 \mod 32$$  \hspace{1cm} (15)

Thus:

$$m + 16 \mod 32 = ((n + 16 \mod 32) + 16) \mod 32 = n + 16 + 16 \mod 32 = n + 32 \mod 32 = n, \text{ since } n \in \mathbb{Z}/32\mathbb{Z}$$

Thus, we have proved that for any given $p_n \in M$:

$$f(f(p_n)) = p_n$$  \hspace{1cm} (16)

4 **Fixed $\lambda$ parameter of the chaotic logistic map**: the parameter $\lambda$ of the chaotic logistic map is by definition belongs to the range $(0, 4]$ of real numbers. However, the logistic map exhibits chaotic behavior only when the parameter $\lambda$ belongs to some distinct regions within $(0, 4]$ but not for all the interval. The determination of these chaotic regions needs deep analysis of the bifurcation diagram to exclude all non-chaotic periodic windows. Nevertheless, in [28], only one fixed value of parameter $\lambda = 3.99$ is used in every call to the chaotic map. It is recommended to make this parameter as a secret key. In this case, when the parameter $\lambda$ is secret, the system will surely be more robust and immune against some attacks.
No padding method is defined for the DECS-AU: if the padding of non-32-bytes blocks is based on zero-padding (i.e., use null bytes to pad the block of data), then information about the last key block will be clearly manifested (in a shuffled form) in the last cipher block. So, it’s better worth to define a good method for padding blocks.

The applicability of the DNA XOR is not ensured: the possibility to implement the DNA XOR operation as defined in [28], is seriously questionable. The XOR DNA is dependent on the rule at each operation. DNA XOR gates such as in [35] are based on oligonucleotide strands; thus, every binary state (i.e., 0 or 1) is encoded using an oligonucleotide strand, which allows making some DNA logic gates. The feasibility of the DNA XOR operation equivalently to binary XOR, as used in the DECS-AU, should be at least simulated. At this point, we consider that the DNA XOR operation is equivalent to an encoding process assessed with computer.

4 Differential chosen-plaintext attack and chosen-ciphertext attack

4.1 Differential chosen-plaintext attack

The chosen-plaintext attack is an attack in which the attacker gains temporary access to the encryption machinery. Thus, he could choose some intentionally made plaintexts and get their corresponding ciphertexts under an unknown key. The objective of the attack is to recover the key or the equivalent key used for the encryption. As depicted in Proposition 1, if we can recover the first key-block $K_1$, all subsequent keys could be recovered. In this attack, we will demonstrate that only two chosen plaintexts of 32-bytes size are sufficient to recover the first key block $K_1$ and thus recovering the full equivalent key used in the encryption process.

In this section, we denote the first chosen-plaintext as $b_1$ and the second chosen-plaintext as $b_2$, both are blocks of 32-bytes. Their corresponding substituted blocks, according to Eq. (8), are denoted as $S_1$ and $S_2$ respectively. And their corresponding ciphertexts are denoted as $C_1$ and $C_2$ respectively. Fig. 2 illustrates a flowchart of our proposed differential chosen-plaintext attack. As shown in this Figure, our proposed attack could be described in two main steps:

1. Recovering the shuffling map of the first block:

   In order to recover the shuffling map of the first block, we have to satisfy two conditions:
   - Make a unique identifier for each byte position in the plaintext block.
   - Cancel out the substitution process, thus forcing the substituted block to be the plain block, $S_i = b_i$ for $i = 1, 2$.

   Let us detail these two conditions in depth: we have to force the shuffling map to be clearly manifested in our outputs. The intuitive idea to do so is to input an ordered block of bytes (i.e., byte-values goes from 1 to 32) and then observe the new position of each byte-value in the output, which will allow us to deduce the shuffling map. But first, we have strictly to force the system to discard the substitution process and settle for only the shuffling process. In order to achieve that, we will combine a
differential attack with a chosen-plaintext one to cancel out the substitution effect as described in what follows.

The substitution process is based on the DNA XOR operations, and from Eq. (13), we have that DNA XOR operation is equivalent to binary bit-wise XOR operation. Let us define the difference between the two ciphertext blocks as:

$$\Delta C_1; C_2(\pi) = C_1 \oplus C_2$$

From Eq. (12) we have for \( j = 1, \ldots, 32 \):

$$\Delta(C_1, C_2)(\pi) = DNA_{d_{j'}}(DNA_{e_{j'}}(b_1(\pi))) \boxplus DNA_{e_{j'}}(K_1(\pi))) \boxplus DNA_{d_{j'}}(DNA_{e_{j'}}(b_2(\pi))) \boxplus DNA_{e_{j'}}(K_1(\pi)))$$

where \( j^* \) is defined in Eq. (10).

Thus, using Eq. (13) we have:

$$\Delta(C_1, C_2)(\pi) = b_1(\pi) \oplus K_1(\pi) \oplus b_2(\pi) \oplus K_1(\pi) = b_1(\pi) \oplus b_2(\pi)$$

Thus, we can write the difference in ciphertext blocks by means of the difference in plaintext blocks as:

\[ \Delta(C_1, C_2)(\pi) = C_1 \oplus C_2 \]
$\Delta(C_1, C_2)(j) = \Delta(b_1, b_2)(j^*)$ (20)

where $j^*$ is defined in Eq. (10).

In other words, we cancel out the substitution effect and only the shuffling effect remains (because, as we see in Eq. (20), only ciphertexts, plaintexts and shuffling operation remain in the equation). By this mean, we have satisfied the second condition (mentioned at the beginning of this sub-section), and we could easily deduce the permutation map if we choose carefully the two plaintexts $b_1$ and $b_2$ to satisfy the first condition in such a way to have unique identifier value for each byte in $\Delta(b_1, b_2)$ plain block.

One possible proposition is to make the plain blocks difference as:

$\Delta(b_1, b_2)(j) = j, \ j = 1, 2, ..., 32$ (21)

The two chosen-plaintexts could be chosen as:

$\begin{cases} b_1(j) = 0 \\ b_2(j) = j \end{cases}, \ j = 1, 2, ..., 32$ (22)

The shuffling map for the first block could be recovered according to Eq. (10) as:

$M = \{p_i\}_{i=0}^{31} = \{\Delta(b_1, b_2)(j)\}_{j=1}^{16} \sim \{\Delta(C_1, C_2)(j)\}_{j=1}^{16}$ (23)

This process is illustrated in Fig. 2, as we see in the figure, the first chosen-plaintext is an all-zero bytes block, while the second is an ordered bytes block, as depicted in Eq. (22), and after encryption, the difference gives us directly the shuffling map as depicted by Eq. (23). The rest of our attack will describe how to recover the exact full first-key block.

2 Recovering the key block $K_1$:

After the recovery of the shuffling map, the recovery of the key $K_1$ is quite a simple operation. From Eqs. (12), (13) and (22) we have:

$C_1(j) = b_1(j^*) \oplus K_1(j^*) = 0 \oplus K_1(j^*) = K_1(j^*)$ (24)

That is to say, encrypting an all-zero block of data will result in a shuffled version of the key block. Thus, $C_1$ is a shuffled version of the key block $K_1$. According to Proposition 2, we simply apply the shuffling map (recovered from the previous step) once on the ciphertext $C_1$ to recover the key block $K_1$. Generally, according to Proposition 2, we have:

$C_1(j) = K_1(j^*) \bowtie K_1(j) = C_1(j^*)$ (25)

In Fig. 2, we have shown this process by taking the cipher block of only the first chosen-plaintext, and applying the shuffling map as depicted in the previous point, by this way, we will get directly the de-shuffled key block $K_1$. Until this stage, still only to deduce the full equivalent key to conclude the attack.

3 Recovering all key chains:

After the recovery of the first key block $K_1$, and according to Proposition 1, we use the algorithm described in the key scheduling and chaining algorithm from Section 2
to recover as many key blocks as needed to decrypt any other plaintext encrypted under the same master key $MK$.

Figure 3 demonstrates our proposed differential chosen-plaintext attack: since audio data are audible, to show our attacks experimentally, we have chosen to adopt two audio representations that are the time-domain representation and frequency-domain representation. The time-domain plots of the two chosen-plaintexts $b_1$ and $b_2$ are shown respectively in Figs. 3a and b and their corresponding ciphertexts time-domain plots are shown respectively in Figs. 3c and d. The frequency-domain plot of the supposed unknown plaintext (i.e., a sine wave of frequency $F = 700\text{Hz}$) is shown in Fig. 3e and its corresponding ciphertext’s frequency-domain plot is shown in Fig. 3f. All the samples are encrypted under the same master-key.

Note that all keys used in this paper, are generated randomly using $\text{rand}$ function from Octave,\footnote{Octave freeware under GNU GPL license at https://gnu.org/software/octave/} and that all samples are mono-channel samples sampled under a sampling frequency of $fs = 8000\text{Hz}$ using 8-bits per sample. The recovered key is used to decrypt the ciphertext that its frequency-domain plot is shown in Fig. 3f and a portion of the time-domain as well as the frequency-domain plots of the recovered plaintext are shown respectively in Figs. 3g and h. The recovered plaintext matches the original one (see Figs. 3g and h), which proves the success of our described differential chosen-plaintext attack.

### 4.2 The chosen-ciphertext attack

For a chosen-ciphertext attack, we will demonstrate that only one chosen-ciphertext of 32-bytes is required to directly recover the first key block $K_1$, and thus, recovering any desired equivalent key according to Proposition 1. The flowchart of our proposed chosen-ciphertext attack is illustrated in Fig. 4. As shown in this figure, the chosen cipher-block is chosen to be an all-zero block $C = \{C(j) = 0\}_{j=1}^{32}$, thus, the shuffling-process decryption will have no effect (since all byte-values are with the same value zero). Thus, the substituted block is itself the cipher block $S = C$.

According to Eqs. (12) and (13), and since our chosen block is an all-zero block, we have:

$$C(j) = b(j) \oplus K_1(j) = 0, \quad j = 1, 2, \ldots, 32$$

(26)

Where $b$ is the corresponding plaintext of the chosen ciphertext $C$, and since $C$ is an all-zero block, we have:

$$b(j) = K_1(j), \quad j = 1, 2, \ldots, 32$$

(27)

That is, the key block $K_1$ is manifested directly in the decrypted plaintext $b$ as shown in Fig. 4. In other words, deciphering an all-zero ciphertext will give us directly the first key block. And according to Proposition 1, from that key block only, we can extract any desired number of key blocks.

To assess this theoretical demonstration with experimental simulations, we propose the following chosen-ciphertext attack scenario: let assume that we have gained restricted and temporary access to the decryption machinery so that we have decrypted our chosen all-zero ciphertext $C = \{C(j) = 0\}_{j=1}^{32}$. The time-domain plot of the decrypted block as...
well as its frequency-domain plot are shown in Figs. 5a and b respectively. We have a ciphertext that we want to decrypt, in Fig. 5c and d, we show its time-domain and its frequency-domain plots respectively. Keeping in mind that we don’t know the key used for the encryption, and we have loosed our access to the decryption machinery. Using the decrypted chosen plaintext, which is exactly the first key-block as we proved before, we generate as many key blocks as needed to decrypt our target ciphertext. The results of the decryption using this attack are shown in Figs. 5e and f.

As shown in Fig. 5, we have first decrypted our chosen ciphertext which is an all-zero cipher block, and we get, as we proved before, the first key block, shown in Figs. 5a and b. After that, we have targeted an unknown ciphertext shown in Figs. 5c and d using the recovered key block. And we have generated as many key blocks as needed to decrypt all the targeted cipher. The results of decryption are shown in Figs. 5e and f. We see from Fig. 5e that we have recovered a sine wave sound, and we see from Fig. 5f that its frequency is equal to 700 Hz. This simulation proves the success of our demonstrated chosen-ciphertext attack.
Weak keys are keys that lead to some undesirable results (e.g., no or bad encryption, self-decryption...). If, for example, encrypting some plaintext with the same key twice leads to recovering the original plaintext, then this key is considered a weak key. We extend this definition to our proposed cycle attack to consider a relatively small number of successive encryptions \( c \geq 2 \).

Our proposed cycle attack is based on repeating encryptions of a particular unknown plaintext \( c \) times under the same unknown key \( K \). Fig. 6 shows the general flowchart of a cycle attack, in general, the attacker can feed back the ciphertext (after each encryption process) to be re-encrypted, doing this until the plaintext is recovered after some successive number of encryptions (i.e., serial encryptions). This attack could be seen as a chosen-plaintext attack with \( c - 1 \) chosen plaintexts. The cycle attack could succeed (e.g., be feasible) only under weak keys, that’s why for any cryptographic system it is better worth to exclude all weak keys from the key space.

From Proposition 2, we know that the range (i.e., cycle) of all shuffling keys is equal to two: \( R = 2 \). Thus, for any key, if the encryption process is applied twice, the shuffling permutation will be canceled.
Lemma 1: for any even number $c \in 2\mathbb{N}$, applying $c$ successive DECS-AU encryptions on the same plaintext will cancel out the shuffling process effect.

Proof: trivially proved using Proposition 2: the period of the shuffling process is equal to two. So, by applying any multiple of two $c \in 2\mathbb{N}$ encryptions, the shuffling effect will be discarded. ■

Moreover, from Eq. (13), the substitution process is based on the XOR operation. Bit-wise XOR operation is an involutory operator, characterized by the fact that for any byte $a$ we have the following property: $a \oplus a = 0$. Thus, in order to cancel out the effect of XOR operation, we have to apply it twice. The XOR masking process is an involutory function (i.e., which is a function that is its own inverse).

Proposition 3: Under any master key $K$, the DECS-AU scheme is weak under our described cycle attack. And only $c = 4$ application of successive encryptions is sufficient to recover the unknown original plaintext.

Proof: Let $b$ be a plain block and $C$ is its corresponding cipher block under some key $K$. We define $b(k)$ as the $k-th$ byte of $b$, and we have to prove that after four encryptions we have: $C^4(k) = b(k)$ where $C^i$ is the cipher block after $i$-encryptions with the same key. According to Eqs. (12) and (13) we can write any encryption as follow:

$$
\begin{align*}
C^1(k^*) &= b(k) \oplus K(k) \\
C^i(k^*) &= C^{i-1}(k) \oplus K(k), \quad i \geq 2
\end{align*}
$$

Fig. 4 General flowchart of our proposed chosen-ciphertext attack
Fig. 5 Simulation of our chosen-ciphertext attack (a) time-domain plot of the decrypted chosen ciphertext (b) frequency-domain plot of the decrypted chosen ciphertext (c) time-domain plot of the target ciphertext (d) frequency-domain plot of the target ciphertext (e) time-domain plot of the recovered plaintext using chosen-ciphertext attack (f) frequency-domain plot of the recovered plaintext using chosen-ciphertext attack.
And according to Eq. (25) we can write the following equivalence:

\[ C^i(k^*) = C^{i-1}(k) \oplus K(k) \iff C^{i}(k) = C^{i-1}(k^*) \oplus K(k^*) \]  

(29)

So, we will be based on Eqs. (28) and (29) to write and simplify our ciphertexts in what follows. The first encryption is written as:

\[ C^1(k^*) = b(k) \oplus K(k) \]  

(30)

The simplified second-encryption could be written (after substituting \( C^i(k^*) \) as in Eq. (30)), as:

\[ C^2(k) = C^1(k^*) \oplus K(k^*) = b(k) \oplus K(k) \oplus K(k^*) \]  

(31)

The same process is repeated for the third encryption:

\[ C^3(k^*) = C^2(k) \oplus K(k) = b(k) \oplus K(k) \oplus K(k^*) \oplus K(k) = b(k) \oplus K(k^*) \]  

(32)

And finally, the fourth encryption is written as:

\[ C^4(k) = C^3(k^*) \oplus K(k^*) = b(k) \oplus K(k^*) \oplus K(k^*) = b(k) \]  

(33)

That is, we have proved that \( C^4 = b \) which implies that after four encryptions under the DECS-AU scheme, we will recover the original plaintext.

\[ \square \]
To experimentally demonstrate our proposed cycle attack, we consider a plaintext,\(^4\) (supposed to be unknown), where both time-domain and frequency-domain plots are shown in Figs. 7a and b respectively. Figs. 7c and d represent its time-domain and frequency-domain plots respectively after one encryption. Figs. 7e and f represent the same plots respectively after two encryptions. The same plots are presented in Figs. 7g and h, but after three encryptions. And Fig. 7i and j show the time-domain and frequency-domain plots, respectively, of the fourth encryption operation. As shown, in Fig. 7, the fourth encryption returns back to the original plaintext which demonstrates the success of our described attack experimentally.

6 Lessons learned and recommendations for further works

In this section, we outline some lessons learned from this cryptanalytic paper, as well as some recommendations to be considered in improving the DECS-AU system or in designing further proposals. However, the DECS-AU system is considered weak at this stage. Some recommendations for designing similar systems are outlined below:

1. Make strong confusion/diffusion between key blocks and plain blocks; thus, it is recommended to use some cryptographic modes (e.g., CBC, CFB, OFB …) in order to diffuse the plain blocks along with both cipher blocks and key blocks. Fig. 8 shows a diagram of a CBC (code block chaining) mode, in which each ciphertext block depends on all previous plaintext blocks (IV is an initialization vector used to encrypt the first block). As shown in Fig. 8, this chaining mode starts with an IV (initialization vector), and XOR it with the first plain-block before encrypting the result with the encryption machinery. The resulting cipher block is then XORed with the next plain block, and the result is then encrypted using the encryption machinery. This process is repeated until the last plain block gets encrypted. This chaining mode guarantees excellent properties of confusion/diffusion, and hence, the security of the encryption scheme will be enhanced and will be immune against several types of attacks. We recommend the use of chaining modes for any new proposals. A more detailed description of several chaining modes is presented in [Section 3, 33].

2. Use some existing DNA computing operations. The used DNA encoding in the scheme under study is not a real DNA-computing process. It is just an encoding process that uses symbols inherited from DNA terminology. We recommend the use of real DNA-computing techniques for DNA cryptography proposals. We encourage the use of real DNA XOR operation as in [15, 35].

3. Change the shuffling method and use more robust permutation maps. For example, we propose generating entirely key-dependent permutation maps using other chaotic maps or any other methods to generate permutation maps. In addition to that, we recommend the use of both intra-block and inter-block permutations. That is, permuting bytes within a block and also from block to another one, according to some permutation maps. By this, we will avoid low periods (ranges) in the permutation process, and thus, the security of the scheme will be enhanced more.

\(^4\)The audio sample used for this illustration is available freely on: https://archive.codeplex.com/?p=audiotestfiles
Fig. 7 Demonstration of our proposed cycle attack: (a) the time-domain plot of the original plaintext, (b) the frequency-domain plot of the original plaintext, (c) the time-domain plot of the ciphertext after the first encryption, (d) the frequency-domain plot of the ciphertext after the first encryption, (e) the time-domain plot of the ciphertext after the second encryption, (f) the frequency-domain plot of the ciphertext after the second encryption, (g) the time-domain plot of the ciphertext after the third encryption, (h) the frequency-domain plot of the ciphertext after the third encryption, (i) the time-domain plot of the ciphertext after the fourth encryption, (j) the frequency-domain plot of the ciphertext after the fourth encryption.
4 Use other chaotic maps since the logistic map is not recommended for cryptographic use, as shown in [3]. We recommend using maps with robust chaos that are chaotic on the entire range of parameter space. For more details about robust chaos, one can refer to [37].

7 Discussion and conclusion

In this paper, we analyzed the security of a new audio-data cryptosystem proposed in [28], based on key chaining and DNA encoding. The key chaining algorithm aims to generate a new key block for every plain block as seen in Section 2. This process, as claimed by authors of the scheme, allows enhancing the security of the system. Nevertheless, we found that this process has two main drawbacks: the first one is depicted in Section 3 (vulnerability (1)), as this process takes into consideration neither the plain block nor the cipher block to generate new key blocks, which lead to a weak confusion/diffusion implementation. The second one is that the shuffling function has a period of two (see Section 3, vulnerability (3)), so repeating the...
shuffling process twice will discard the effect of the shuffling process. Besides, we have noticed that repeating the overall process of the key generation for every single plain block is time-consuming. Some other vulnerabilities found in the scheme are detailed in Section 3, including the weak cryptographic-effect of the DNA encoding, and the fixed parameter of the chaotic map, which contributes in major part to the deficiency of the system.

In addition to that, we have mounted three conventional attacks (i.e., differential chosen-plaintext attack, chosen-ciphertext attack and cycle attack) to evaluate the system’s security and robustness. We have found that the system is not secure against these attacks. The chosen-ciphertext attack is the simplest one since it has the lowest time complexity, but it requires temporary access to the decryption machinery under the target key, we have proved, that only one chosen-ciphertext of 32-bytes is required to mount this attack and recover any equivalent key successfully. The two other attacks are both considered as a chosen-plaintext attack, but they differ in some aspects. The first attack named differential chosen-plaintext attack; it is actually based on two combined attacks: differential attack and chosen-plaintext attack. We have proved that the minimum number of required chosen plaintexts for this proposed attack is two chosen-plaintexts with 32-bytes each. The second one called cycle attack, and it is a type of chosen-plaintext attack, but it does not require any intervention from the attacker, such that programming new pieces of code, making some customized sets of plaintexts or do calculations… All the attacker needs, is the ciphertext to be decrypted and a temporary access to the encryption machinery. That is, by re-encrypting any given ciphertext three successive encryptions using the same key, we end up recovering the original unknown plaintext. If needed, the key could be recovered (fully or partially) by applying a known-plaintext attack.

Our attacks’ demonstrations are shown visually based on time-domain plot, frequency-domain plot, or both of them. We have chosen to depict the most convenient plot type for every result to shed light on the main differences between results and be as representative as possible. For example, for sine waves, it is better to use frequency-domain plots to visualize the information data. Based on our results, we have detailed our observed vulnerabilities, and
we have addressed some recommendations deduced from this cryptanalytic paper to avoid such vulnerabilities in further works.

References

1. Adleman LM (1994) Molecular computation of solutions to combinatorial problems. Sci (80). https://doi.org/10.1126.science.7973651
2. Alvarez G, Li S (2006) Some basic cryptographic requirements for chaos-based cryptosystems. Int J Bifurc Chaos 16:2129–2151. https://doi.org/10.1142/S0218127406015970
3. Arroyo D, Alvarez G, Fernandez V (2008) On the inadequacy of the logistic map for cryptographic applications. arXiv Prepr arXiv:08054355
4. Barua R, Misra J (2003) Binary arithmetic for DNA computers. Lect Notes Comput Sci (including Subser Lect Notes Artif Intell Lect Notes Bioinformatics). https://doi.org/10.1007/3-540-36440-4_11
5. Bensikaddour EH, Bentoutou Y, Taleb N (2020) Embedded implementation of multispectral satellite image encryption using a chaos-based block cipher. J King Saud Univ - Comput Inf Sci 32:50–56. https://doi.org/10.1016/j.jksuci.2018.05.002
6. Bhargava B, Shi C, Wang SY (2004) MPEG video encryption algorithms. Multimed Tools Appl 24:57–79. https://doi.org/10.1023/B:MTAP.0000033983.62130.00
7. Broemeling LD (2011) An account of early statistical inference in Arab cryptology. Am Stat 65:255–257. https://doi.org/10.1198/tas.2011.10191
8. Chen S, Zhong XX, Wu ZZ (2008) Chaos block cipher for wireless sensor network. Sci China, Ser F Inf Sci 51:1055–1063. https://doi.org/10.1007/s11432-008-0102-5
9. Du M, Wang K, Chen Y et al (2018) Big data privacy preserving in multi-access edge computing for heterogeneous internet of things. IEEE Commun Mag 56:62–67. https://doi.org/10.1109/MCOM.2018.1701148
10. El Hanouti I, El Fadili H, Zenkouar K (2020) Breaking an image encryption scheme based on Arnold map and Lucas series. Multimed Tools Appl. https://doi.org/10.1007/s11042-020-09815-4
11. El-Latif AAA, Li L, Niu X (2014) A new image encryption scheme based on cyclic elliptic curve and chaotic system. Multimed Tools Appl 70:1559–1584. https://doi.org/10.1007/s11042-012-1173-2
12. Gencoglu MT (2019) Embedded image coding using Laplace transform for Turkish letters. Multimed Tools Appl 78:17521–17534. https://doi.org/10.1007/s11042-018-7096-9
13. Genot AJ, Bath J, Turberfield AJ (2011) Reversible logic circuits made of DNA. J Am Chem Soc 133:20080–20083. https://doi.org/10.1021/ja208497p
14. Ghadi M, Laouamer L, Moulahi T (2016) Securing data exchange in wireless multimedia sensor networks: perspectives and challenges. Multimed Tools Appl 75:3425–3451. https://doi.org/10.1007/s11042-014-2443-y
15. Gupta V, Parthasarathy S, Zaki M (1999) Arithmetic and logic operations with DNA
16. Habib IW, Saadawi TN (2002) Multimedia traffic characteristics in broadband networks. IEEE Commun Mag 30:48–54. https://doi.org/10.1109/35.144777
17. Hraoui S, Gmira F, Jarar AO, Saaidi A (2013) Benchmarking AES and chaos based logistic map for image encryption. In: Proceedings of IEEE/ACS International Conference on Computer Systems and Applications, AICCSA. https://doi.org/10.1109/AICCSA.2013.6616441
18. Iqbal S, Malik M, Iqbal S, Muhammad O, Sher H (2008) Study of nonlinear dynamics using logistic map. In: LUMS 2nd international conference on mathematics and its applications in information technology (LICM08)
19. Janakiraman S, Thenmozhi K, Rayappan JBB, Amirtharajan R (2018) Lightweight chaotic image encryption algorithm for real-time embedded system: implementation and analysis on 32-bit microcontroller. Microprocess Microsystems 56:1–12. https://doi.org/10.1016/j.micropro.2017.10.013
20. Kerckhoffs A (1883) Auguste Kerckhoffs “La cryptographie militaire.” Des Sci Mil
21. Larew K, Kahn D (1968) The Codebreakers; the story of secret writing. Am Hist Rev 74:537. https://doi.org/10.2307/1853680
22. Li S, Li C, Chen G, Lo KT (2008) Cryptanalysis of the RCES/RSES image encryption scheme. J Syst Softw 81:1130–1143. https://doi.org/10.1016/j.jss.2007.07.037
23. Li H, Wang K, Liu X, Sun Y, Guo S (2017) A selective privacy-preserving approach for multimedia data. IEEE Multimed 24:14–25. https://doi.org/10.1109/MMUL.2017.4031322
24. Li C, Xie T, Liu Q, Cheng G (2014) Cryptanalyzing image encryption using chaotic logistic map. Nonlinear Dyn 78:1545–1551. https://doi.org/10.1007/s11071-014-1533-8
25. Lin Z, Liu J, Lian J, Ma Y, Zhang X (2019) A novel fast image encryption algorithm for embedded systems. Multimed Tools Appl 78:20511–20531. https://doi.org/10.1007/s11042-018-6824-5
26. Masood F, Ahmad J, Shah SA, Jamal SS, Hussain I (2020) A novel hybrid secure image encryption based on Julia set of fractals and 3D lorenz chaotic map. Entropy. https://doi.org/10.3390/e22030274
27. Mishra D, Vijayakumar P, Sureshkumar V, Amin R, Islam SKH, Gope P (2018) Efficient authentication protocol for secure multimedia communications in IoT-enabled wireless sensor networks. Multimed Tools Appl 77:18295–18325. https://doi.org/10.1007/s11042-017-5376-4
28. Naskar PK, Paul S, Nandy D, Chaudhuri A (2019) DNA encoding and channel shuffling for secured encryption of audio data. Multimed Tools Appl 78:25019–25042. https://doi.org/10.1007/s11042-019-7696-2
29. Özkaynak F (2018) Brief review on application of nonlinear dynamics in image encryption. Nonlinear Dyn 92:305–313. https://doi.org/10.1007/s11071-018-4056-x
30. Reif JH (1998) Paradigms for biomolecular computation. In: Unconventional models of computation. Springer, New york, pp. 72–93
31. Shakir HR (2019) An image encryption method based on selective AES coding of wavelet transform and chaotic pixel shuffling. Multimed Tools Appl 78:26073–26087. https://doi.org/10.1007/s11042-019-07766-z
32. Shannon CE (1949) Communication theory of secrecy systems. Bell Syst Tech J 28:656–715. https://doi.org/10.1002/j.1538-7305.1949.tb00928.x
33. Smekal D, Hajny J, Martinasek Z (2018) Comparative analysis of different implementations of encryption algorithms on FPGA network cards. IFAC-PapersOnLine 51:312–317. https://doi.org/10.1016/j.ifacol.2018.07.172
34. Ur RA, Liao X (2019) A novel robust dual diffusion/confusion encryption technique for color image based on Chaos, DNA and SHA-2. Multimed Tools Appl 78:2105–2133. https://doi.org/10.1007/s11042-018-6346-1
35. Yan H, Feng L, LaBean TH, Reif JH (2003) Parallel molecular computations of pairwise exclusive-or (XOR) using DNA “string tile” self-assembly. J Am Chem Soc 125:14246–14247. https://doi.org/10.1021/ja036670m
36. Yap WS, Phan RCW, Yau WC, Heng SH (2015) Cryptanalysis of a new image alternate encryption algorithm based on chaotic map. Nonlinear Dyn 80:1483–1491. https://doi.org/10.1007/s11071-015-1956-x
37. Zeraouilia E, Sprott JC (2011) Robust Chaos and its applications. World Scientific
38. Zhang Y, Wang Z, Wang Z, Liu X, Yuan X (2017) A DNA-based encryption method based on two biological axioms of DNA Chip and Polymerase chain reaction (PCR) amplification techniques. Chem - A Eur J 23:13387–13403. https://doi.org/10.1002/chem.201701411

Publisher’s note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.