Security Analysis of a Color Image Encryption Scheme Based on Dynamic Substitution and Diffusion Operations

George Teșeleanu$^{1,2}$

$^1$Advanced Technologies Institute, 10 Dinu Vintilă, Bucharest, Romania
$^2$Simion Stoilow Institute of Mathematics of the Romanian Academy, 21 Calea Grivitei, Bucharest, Romania

Abstract: In 2019, Essaid et al. proposed an encryption scheme for color images based on chaotic maps. Their solution uses two enhanced chaotic maps to dynamically generate the secret substitution boxes and the key bytes used by the cryptosystem. Note that both types of parameters are dependent on the size of the original image. The authors claim that their proposal provides enough security for transmitting color images over unsecured channels. Unfortunately, this is not the case. In this paper, we introduce two cryptanalytic attacks for Essaid et al.’s encryption scheme. The first one is a chosen plaintext attack, which for a given size, requires 256 chosen plaintexts to allow an attacker to decrypt any image of this size. The second attack is a a chosen ciphertext attack, which compared to the first one, requires 512 chosen ciphertexts to break the scheme for a given size. These attacks are possible because the generated substitution boxes and key bits remain unchanged for different plaintext images.

1 INTRODUCTION

Because the use of social media is growing exponentially, the protection of digital images has become a sensitive topic. Therefore, the protection of images against theft and illegal distribution has attracted much attention. As a result, many researchers have proposed a variety of image encryption schemes. One of the most popular types of image encryption schemes are those based on chaotic maps, due to their high sensitivity to the previous states, initial conditions or both. This property makes them highly desirable because their sensitivity makes it difficult to predict their behaviour or outputs. Hence, novel chaos based cryptographic algorithms have been proposed over the years. We refer the reader to (Zolfaghari and Koshiba, 2022; Muthu and Murali, 2021; Hosny, 2020) for some surveys of such proposals. Unfortunately, insufficient security analysis and a lack of design guidelines have led to the discovery of serious security flaws in a substantial number of chaos based image encryption schemes. To illustrate our point we further present a list of broken schemes in Table 1. Note that the list is not comprehensive.

In (Essaid et al., 2019) a chaos based encryption scheme is proposed. The authors use the Enhanced Logistic Map (ELM) and Enhanced Sine Map (ESM) as pseudorandom number generators (PRNGs). Using these two PRNGs, Essaid et al. randomly generate two substitution boxes (s-boxes), which are then used to compute the rest of the s-boxes required by the cryptosystem. Then, the PRNGs are combined to create the necessary key bytes. Since ELM and ESM are simply used as PRNGs and the scheme’s weakness is independent of the employed generators, we omit their description and simply consider the two s-boxes and the key bytes as being randomly generated.

In this paper we conducted a security analysis on the Essaid et al. scheme. More precisely, we propose a chosen plaintext attack and a chosen ciphertext attack that allow an attacker to decrypt all images of a given size. In order to achieve this, we need the corresponding ciphertexts of 256 chosen plaintexts or the corresponding plaintexts of 512 chosen ciphertexts. For completeness, we also analysed the Essaid et al. scheme when all the s-boxes are randomly generated. This should be the most secure version of their scheme, since there are no relationships between the s-boxes that would allow an attacker to filter out the correct key. Unfortunately, even when the s-boxes are random, our proposed attacks succeed in recovering the encrypted images except two or four pixels, de-
Table 1: Broken chaos based image encryption algorithms.

<table>
<thead>
<tr>
<th>Scheme</th>
<th>(Matoba and Javidi, 2004)</th>
<th>(Wang et al., 2012)</th>
<th>(Huang et al., 2014)</th>
<th>(Khan, 2015)</th>
<th>(Wong and Quc, 2015)</th>
<th>(Chen et al., 2015)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Broken by</td>
<td>(Wang et al., 2019)</td>
<td>(Arroyo et al., 2011)</td>
<td>(Wei et al., 2021)</td>
<td>(Akhmar et al., 2021)</td>
<td>(Wei et al., 2021)</td>
<td>(Hua et al., 2017)</td>
</tr>
<tr>
<td>Scheme</td>
<td>(He et al., 2017)</td>
<td>(Nayal et al., 2017)</td>
<td>(Hua and Zhou, 2017)</td>
<td>(Pak and Huang, 2017)</td>
<td>(Li et al., 2020)</td>
<td>(Yosefnezhad Irani et al., 2019)</td>
</tr>
<tr>
<td>Broken by</td>
<td>(Li et al., 2019)</td>
<td>(Li et al., 2019)</td>
<td>(Yu et al., 2021)</td>
<td>(Wang et al., 2018)</td>
<td>(Hu et al., 2020)</td>
<td>(Wen et al., 2019)</td>
</tr>
<tr>
<td>Scheme</td>
<td>(Sibgatullina et al., 2019)</td>
<td>(Wei et al., 2019)</td>
<td>(Wu et al., 2019)</td>
<td>(Xiong et al., 2019)</td>
<td>(Khan and Maroud, 2019)</td>
<td>(Mendal et al., 2021)</td>
</tr>
<tr>
<td>Broken by</td>
<td>(Zaizou et al., 2019)</td>
<td>(Chen et al., 2020)</td>
<td>(Liu et al., 2020)</td>
<td>(Fan et al., 2021)</td>
<td>(Li et al., 2019)</td>
<td>(Li et al., 2021)</td>
</tr>
</tbody>
</table>

Depending on the type of attack.

Structure of the Paper. We provide the necessary preliminaries in Section 2. In Section 3 we show how an attacker can recover the private key and secret s-boxes in a chosen plaintext scenario. We also provide a key and s-boxes recovery attack in a chosen ciphertext attack in Section 4. We conclude in Section 5.

Algorithm 1: Encryption algorithm.

```
Input: A plaintext p, an s-box list s, a secret seed seed and a secret key k
Output: A ciphertext c
1 for i ∈ [0,H] and j ∈ [0,W] do
2   if i = 0 and j = 0 then
3     c_{ij} ← (seed ⊕ k_{ij} ⊕ s_{ij}(p_{ij})) mod 256
4   else if j = 0 then
5     c_{ij} ← (c_{i-1,j} ⊕ k_{ij} ⊕ s_{ij}(p_{ij})) mod 256
6   else
7     c_{ij} ← (c_{i,j-1} ⊕ k_{ij} ⊕ s_{ij}(p_{ij})) mod 256
8 return c
```

2 PRELIMINARIES

Notations. In this paper, the subset \{1, ..., s − 1\} ∈ N is denoted by \{1, s\}. The action of selecting a random element x from a sample space X is represented by \( x \overset{\text{rand}}{\leftarrow} X \), while \( x \leftarrow y \) indicates the assignment of value y to variable x. In the case of matrices, the \( \leftarrow \) operator assigns the values position by position and the \( \Rightarrow \) operator tests the equality between all positions of the two matrices. We use the C++ language operator & as reference to a variable. By H and W we denote an image’s height and width. Also, we denote by \( L = H + W - 1 \). Hexadecimal numbers will always contain the prefix 0x.

2.1 Essaid et Al. Image Encryption Scheme

In this section we present Essaid et al.’s encryption (Algorithm 1) and decryption (Algorithm 2) algorithms as described in (Essaid et al., 2019). We further consider two cases

- s-boxes \( s_0 \) and \( s_1 \) are randomly generated and the remaining ones are generated using Algorithm 3;
- all the s-boxes in list \( s \) are randomly generated.

The first version is according to the original paper (Essaid et al., 2019), while the second one is introduced to show that the scheme remains broken even if the s-boxes are chosen at random. Note that the seed and the key bytes \( k_{i,j} \) are randomly generated.

Algorithm 2: Decryption algorithm.

```
Input: A ciphertext c, an inverse s-box list \( s^{-1} \), a secret seed seed and a secret key k
Output: A plaintext p
1 for i ∈ [0,H] and j ∈ [0,W] do
2   if i = 0 and j = 0 then
3     p_{ij} ← s^{-1}_{ij}(c_{ij} ⊕ k_{ij} ⊕ seed) mod 256
4   else if j = 0 then
5     p_{ij} ← s^{-1}_{ij}(c_{ij} ⊕ k_{ij} ⊕ seed) mod 256
6   else
7     p_{ij} ← s^{-1}_{ij}(c_{ij} ⊕ k_{ij} ⊕ seed) mod 256
8 return p
```

Algorithm 3: S-Box table generator.

```
Input: Two s-boxes \( s_0 \) and \( s_1 \)
Output: An s-box list s
1 for i ∈ [2,L] and j ∈ [0,256] do
2   s_{ij} ← \[ s_{i,j-1} | s_{i,j} | s_{i,j+1} \]
3 return s
```

3 CHOSEN PLAINTEXT ATTACK

In a chosen plaintext attack (CPA), the attacker A has temporary access to the encryption machine \( O_{enc} \) and can interrogate it on different inputs. Therefore, A constructs some plaintexts that are useful for his attack and then using \( O_{enc} \) obtains the corresponding ciphertexts.

We further show that Essaid et al.’s image encryption scheme is insecure in the chosen plaintext scenario, regardless of whether the generation method of the s-boxes is random or Algorithm 3 is used. The only difference between the two cases is the run time of the attack.

3.1 Randomly Generated S-Boxes

Before formally stating our attack, we first provide an example in order to provide the intuition behind our CPA attack.

Example 3.1. We further assume that we encrypt images of height 3 and width 4. We present in Figure 1...
how Essaid et al.’s encryption algorithm (see Algorithm 1) uses the generated s-boxes. We can see that the algorithm uses the same s-box for each cell on a given minor diagonal.

<table>
<thead>
<tr>
<th>s0</th>
<th>s1</th>
<th>s2</th>
<th>s3</th>
</tr>
</thead>
<tbody>
<tr>
<td>80</td>
<td>81</td>
<td>82</td>
<td>83</td>
</tr>
<tr>
<td>s1</td>
<td>s2</td>
<td>s3</td>
<td>s4</td>
</tr>
<tr>
<td>s2</td>
<td>s3</td>
<td>s4</td>
<td>84</td>
</tr>
</tbody>
</table>

Figure 1: Used s-boxes in an image with $H = 3$ and $W = 4$.

Let’s assume that the image $I_{pv}$, we want to encrypt has the same pixel value $pv$ everywhere. We further write $c_{ij}[pv]$ for the ciphertext byte $c_{ij}$ computed for $I_{pv}$. Then, if we write the ciphertext equations for the third minor diagonal we obtain

$$c_{0,2}[pv] \leftarrow (c_{0,1}[pv] \oplus k_{0,2} + s_2[pv]) \mod 256 \quad (1)$$

$$c_{1,1}[pv] \leftarrow (c_{1,0}[pv] \oplus k_{1,1} + s_2[pv]) \mod 256 \quad (2)$$

$$c_{2,0}[pv] \leftarrow (c_{3,1}[pv] \oplus k_{2,0} + s_2[pv]) \mod 256 \quad (3)$$

From Equations (1) and (2) we derive

$$c_{0,2}[pv] - c_{1,1}[pv] \equiv (c_{0,1}[pv] \oplus k_{0,2} - c_{1,0}[pv] \oplus k_{1,1}) \mod 256. \quad (4)$$

If we request $O_{mc}$ the ciphertexts for all 256 images $I_{0}, \ldots, I_{255}$ we obtain 256 equations of type Equation (4). By checking all the values $x$ and $y$ that satisfy

$$c_{0,2}[pv] - c_{1,1}[pv] \equiv (c_{0,1}[pv] \oplus x - c_{1,0}[pv] \oplus y) \mod 256,$$

for all $pv$ values, we find the correct key pair $(k_{0,2}, k_{1,1})$ and an equivalent key pair $(k_{0,2} \oplus 0x80, k_{1,1} \oplus 0x80)$. The second solution is derived from the following relations

$$c_{0,2}[pv] \equiv (c_{0,1}[pv] \oplus k_{0,2} + 128$$

$$+ s_2[pv] \oplus 0x80) \mod 256$$

which lead to

$$c_{0,2}[pv] - c_{1,1}[pv] \equiv (c_{0,1}[pv] \oplus x \oplus 0x80$$

After computing $k_{0,2}$ using Equation (1) we recover the correct s-box entries

$s_2[pv] \leftarrow (c_{0,2}[pv] - c_{0,1}[pv] \oplus k_{0,2}) \mod 256$

and from Equation (3) we obtain

$$k_{2,0} \equiv (c_{2,0}[0] - s_2[0] \mod 256) \oplus c_{1,3}[0].$$

We also obtain an equivalent s-box $\tilde{s}_2$ from

$$\tilde{s}_2[pv] \leftarrow (c_{0,2}[pv] - c_{0,1}[pv] \oplus k_{0,2} \oplus 0x80) \mod 256$$

and from Equation (3) we obtain

$$k_{2,0} \oplus 0x80 \equiv (c_{2,0}[0] - \tilde{s}_2[0] \mod 256) \oplus c_{1,3}[0].$$

We can easily see that both the correct key bytes $k_{0,2}, k_{1,1}, k_{2,0}$ and s-box $s_2$, and the equivalent key bytes $k_{0,2} \oplus 0x80, k_{1,1} \oplus 0x80, k_{2,0} \oplus 0x80$ and s-box $\tilde{s}_2$ can be used for decryption. Since we cannot determine the order of the correct and equivalent key bytes when we brute force $x$ and $y$, we assume that the first solution we obtain is the correct one.

We repeat the same procedure for the second, forth and fifth minor diagonals. The only keys bytes and s-boxes that we cannot recover using the above technique are $k_{0,0}, k_{2,3}, k_{0,4}$ and $s_5$.

The formal description of our chosen plaintext attack is provided in Algorithm 4 and is a generalization of the method presented in Example 3.1. For completeness, in Algorithm 4 we compute two possible key, s-boxes pairs. Note that in order to be able to use our attack we must have $H, W \geq 2$.

The complexity of Algorithm 4 is $O(2^{24}L + HW)$ and we need 256 oracle queries. For example, if we encrypt 2 megapixels images we obtain that the complexity of Algorithm 4 is $O(2^{24} \cdot 2^{12.45} + 2^{20.87}) = O(2^{35.45})$. In the case of 12 megapixels, we obtain $O(2^{24} \cdot 2^{12.77} + 2^{23.5}) = O(2^{36.77})$.

### 3.2 Essaid et Al.’s Generation Method for S-Boxes

As in the previous subsection, we begin with an example.

**Example 3.2.** Compared to Example 3.1, in the case of Essaid et al.’s generation method it is enough to compute $s_0, s_1, s_2, s_3, s_4, \text{and } s_5$ and the remaining s-boxes can be easily deduced using Algorithm 3. Note that in this case we can also compute $s_0$ and $s_5$.

The first thing that we do after deducing $s_0, s_1, s_2, s_3, s_4, s_5$ from the 256 encrypted images is to compute $s_{0,1}, s_{0,2}, s_{1,2}, s_{0,3,4}$ and $s_5$ and the remaining s-boxes can be easily deduced using Algorithm 3.

Let $pos \in \{0, 2\}$. After computing $s_{pos,1}, s_{pos,2}$ and $s_{pos,3}$, the remaining s-boxes can be calculated as follows:

$$s_{pos,4} = s_{pos,3} \oplus s_{pos,2}$$

$$s_{pos,5} = s_{pos,4} \oplus s_{pos,5}$$

$^1W \times H = 1600 \times 1200$

$^2W \times H = 4000 \times 3000$
Algorithm 4: CPA attack (randomly generated).

1. for $t \in [0, 256)$ do
2. for $i \in [0, H)$ and $j \in [0, W)$ do $p_{i,j} \leftarrow t$
3. Send the ciphertext $t$ from the encryption oracle $O_{enc}$.
4. Receive the ciphertext $t'$ from the encryption oracle $O_{enc}$.
5. $\epsilon_t \leftarrow \epsilon$
6. for $j \in [1, L - 1)$ do
7. $\alpha \leftarrow \max(0, j - (W - 1)); \beta \leftarrow \min(j, H - 1); pos \leftarrow 0$
8. for $x \in [0, 256)$ and $y \in [0, 256)$ do $\tilde{s}_x \leftarrow \alpha$
9. $ctr \leftarrow 0$
10. for $t \in [0, 256)$ do
11. $f \leftarrow \alpha_{i,j} - \alpha_{i+1,j} mod 256$
12. if $f \neq \alpha \oplus 1$ then $g \leftarrow (\alpha_{i,j} - \alpha_{i-1,j} \oplus x) mod 256$
13. else $g \leftarrow (\alpha_{i,j} - \alpha_{i-1,j} \oplus y) mod 256$
14. if $g = f$ then $ctr \leftarrow ctr + 1$
15. if $ctr = 256$ then
16. $k_{pos,i,j} \leftarrow x_{pos,i,j} \oplus g_{pos,i,j} \leftarrow y$
17. for $t \in [0, 256)$ do $f_{pos,i,j} \leftarrow (\alpha_{i,j} - \alpha_{i-1,j} \oplus x) mod 256$
18. for $t \in [0, 256)$ do $f_{pos,i,j} \leftarrow \tilde{s}_x mod 256$
19. for $t \in [\alpha + 1, \beta + 1)$ do $\tilde{s}_{x,i,j} \leftarrow t$
20. $\tilde{s}_{x,i,j} \leftarrow \left(((\tilde{s}_{x,i,j} - \tilde{s}_{x,i}) \oplus pos) mod 256 \oplus \tilde{s}_{x,i+1}ight)$
21. pos $\leftarrow \text{pos} + 1$
22. return $\tilde{s}, \tilde{s}^{-1}$.

\[ \tilde{s}_{pos,0} = \tilde{s}_{pos,2} \oplus \tilde{s}_{pos,1} \text{.} \]

Once all the $s$-boxes are known, we can easily compute the key $k_{pos}$.

We remark that only one of the two solutions is the correct one. We can see that from the following relation

\[ \tilde{s}_3[i] = \tilde{s}_3[i] \oplus 0x80 = \tilde{s}_2[i] \oplus 0x80 = \tilde{s}_2[i] \text{.} \]

Since $\tilde{s}_3$ is not generated by $\tilde{s}_2[i]$, we do not obtain the equivalent key, $s$-boxes pair, and thus the solutions will not decrypt images correctly.

The formal description of our chosen plaintext attack is provided in Algorithm 5 and is a generalization of the method presented in Example 3.2. The complexity of Algorithm 5 is $O(2^{25} + 2^6L + HW)$ and we need 256 oracle queries. For example, if we encrypt 2 megapixels images we obtain that the complexity of Algorithm 5 is $O(2^{24} + 2^{5} \cdot 2^{11.45} + 2^{20.87}) = O(2^{24.21})$. In the case of 12 megapixels, we obtain $O(2^{24} + 2^{5} \cdot 2^{12.77} + 2^{25.5}) = O(2^{24.85})$.

Note that according to (Essaid et al., 2019) the security of their scheme is $O(2^{228})$. Using 256 encrypted images and Algorithm 5, we lower the security strength of Essaid et al.’s scheme from 128 bits to approximately 24 bits.

Algorithm 5: CPA attack (Essaid et al.’s generation method).

6. for $j \in [1, L - 1)$ do
7. $\alpha \leftarrow \max(0, j - (W - 1)); pos \leftarrow 0$
8. for $x \in [0, 256)$ and $y \in [0, 256)$ do $\tilde{s}_x \leftarrow \alpha$
9. $ctr \leftarrow 0$
10. for $t \in [0, 256)$ do
11. $f \leftarrow \alpha_{i,j} - \alpha_{i+1,j} \mod 256$
12. if $f \neq \alpha \oplus 1$ then $g \leftarrow (\alpha_{i,j} - \alpha_{i-1,j} \oplus x) \mod 256$
13. else $g \leftarrow (\alpha_{i,j} - \alpha_{i-1,j} \oplus y) \mod 256$
14. if $g = f$ then $ctr \leftarrow ctr + 1$
15. if $ctr = 256$ then $k_{pos,i,j} \leftarrow x_{pos,i,j} \oplus y$
16. for $t \in [0, 256)$ do $f_{pos,i,j} \leftarrow f \leftarrow (\alpha_{i,j} - \alpha_{i-1,j} \oplus x) \mod 256$
17. for $t \in [0, 256)$ do $f_{pos,i,j} \leftarrow f_{pos,i,j} \oplus pos$
18. for $t \in [\alpha + 1, \beta + 1)$ do $\tilde{s}_{x,i,j} \leftarrow t$
19. $\tilde{s}_{x,i,j} \leftarrow \left(((\tilde{s}_{x,i,j} - \tilde{s}_{x,i}) \oplus pos) \mod 256 \oplus \tilde{s}_{x,i+1}ight)$
20. pos $\leftarrow \text{pos} + 1$
21. for $j \in [1, L - 1)$ do $\tilde{s}_x \leftarrow \alpha$
22. for $t \in [0, 256)$ do $f_{pos,i,j} \leftarrow f_{pos,i,j} \oplus pos$
23. for $t \in [0, 256)$ do $f_{pos,i,j} \leftarrow f_{pos,i,j}$
24. for $pos \in [0, 256)$ do $\tilde{s}_{x,i,j} \leftarrow \left(((\tilde{s}_{x,i,j} - \tilde{s}_{x,i}) \oplus pos) \mod 256 \oplus \tilde{s}_{x,i+1}ight)$
25. pos $\leftarrow \text{pos} + 1$
26. return $\tilde{s}, \tilde{s}^{-1}$.

4 CHOSEN CIPHERTEXT ATTACK

Compared to the chosen plaintext attack, in a chosen ciphertext attack (CCA), $A$ has temporary access to the decryption machine $O_{dec}$. Therefore, $A$ constructs some ciphertexts that are useful for his attack and then using $O_{dec}$ obtains the corresponding plaintexts.

In this scenario, we provide two types of attacks against Essaid et al.’s cryptosystem, one for images with odd width and one for images with even width. Again the s-box generation method is irrelevant to the success of the attacks, the only thing that is affected is their run time.

4.1 Randomly Generated S-Boxes

We first provide an example for images with odd width and then one for images with even width. After,
we present the formal description of our CCA attack.

\[
C_0(t) = \begin{bmatrix}
    2 & 2 & 1 & 2 & 1 \\
    2 & 1 & 1 & 2 & 1 \\
    2 & 1 & 1 & 1 & 1 \\
\end{bmatrix}
\]

\[
C_1(t) = \begin{bmatrix}
    2 & 2 & 1 & 2 & 1 \\
    2 & 1 & 1 & 2 & 1 \\
    2 & 1 & 1 & 1 & 1 \\
\end{bmatrix}
\]

Figure 2: Ciphertext patterns for \( H = 3 \) and \( W = 5 \).

**Example 4.1.** We further assume that we decrypt images of height 3 and width 5. In the first part of the attack we use ciphertexts of type \( C_0(t) \) (see Figure 2). If we exploit the relations for the fourth minor diagonal we obtain

\[
s_3[p_{0,3}[i]] \leftarrow (2 - t \oplus k_{0,3}) \mod 256 \tag{5}
\]

\[
s_3[p_{1,2}[i]] \leftarrow (1 - t \oplus k_{1,2}) \mod 256 \tag{6}
\]

\[
s_3[p_{2,1}[i]] \leftarrow (1 - t \oplus k_{2,1}) \mod 256. \tag{7}
\]

Since we consider all \( t \) values, in Equations (5) and (6) permutation \( s_3 \) iterates through all its values. Therefore, all we need is to find the \( t \) values for which \( p_{0,3} = p_{1,2} \). Therefore, we define

\[
tag{\text{tab}_0} p_{0,3}[i] \leftarrow t \quad \text{and} \quad \text{tab}_1 p_{1,2}[i] \leftarrow t,
\]

and using Equations (5) and (6) we obtain

\[
(2 - \text{tab}_0 p_{0,3}[i] \oplus k_{0,3}) \equiv (1 - \text{tab}_1 p_{1,2}[i] \oplus k_{1,2}) \mod 256,
\]

for all \( i \) values. By checking all the values \( x \) and \( y \) that satisfy

\[
(2 - \text{tab}_0 p_{0,3}[i] \oplus x) \equiv (1 - \text{tab}_1 p_{1,2}[i] \oplus y) \mod 256,
\]

for all \( i \) values, we find the correct key pair \((k_{0,3}, k_{1,2})\) and the equivalent key pair \( (k_{0,3} \perp 0x80, k_{1,2} \perp 0x80)\).

As in Example 3.1, we consider that the first solution is the correct one. Once the first two key bytes are known, we can easily compute the third s-box using Equation (5) and the third key byte using Equation (7).

We repeat the process for the second and sixth minor diagonals. In the second part of our attack, we use ciphertexts of type \( C_1(t) \) (see Figure 2) and then we use the same procedure as before for the third and fifth minor diagonal. The only key bytes and s-boxes that we cannot recover using the above technique are \( k_{0,0}, k_{2,4}, s_0 \) and \( s_9 \).

**Example 4.2.** For the even width case, we consider images of height 3 and width 6. In this case we use the same procedure as in Example 4.1, but instead of using \( C_0(t) \) and \( C_1(t) \) ciphertext patterns, we use \( C_2(t) \) and \( C_3(t) \) (see Figure 3). The only difference between the even and odd width cases is that in the even case we cannot recover \( k_{0,1}, k_{1,0} \) and \( s_1 \). This is because in the even case we could not put \( t \) on the last cell in the first line of \( C_2(t) \) without interfering with the recovery of the other bytes and s-boxes.

Note that we can recover the two key bytes and one s-box if we create one additional ciphertext pattern that satisfies the previously stated condition (see Figure 4). However, we have to ask 256 more oracle queries. Therefore, we decided that is more practical to lose the ability to decrypt two extra bytes, than to ask the additional queries, since images can still be recognized without them. For example, an emoji has a resolution of \( 32 \times 32 \) and removing the four pixels it does not affect the informational content.

\[
C_2(t) = \begin{bmatrix}
    t & 2 & 2 & 2 & 2 & 2 \\
    2 & 2 & 2 & 2 & 1 & 1 \\
    2 & 2 & 2 & 1 & 1 & 1 \\
    2 & 2 & 1 & 1 & 1 & 1 \\
    2 & 1 & 1 & 1 & 1 & 1 \\
\end{bmatrix}
\]

\[
C_3(t) = \begin{bmatrix}
    2 & 2 & 2 & 2 & 2 & 2 \\
    2 & 2 & 2 & 2 & 1 & 1 \\
    2 & 2 & 2 & 1 & 1 & 1 \\
    2 & 2 & 1 & 1 & 1 & 1 \\
    2 & 1 & 1 & 1 & 1 & 1 \\
\end{bmatrix}
\]

Figure 3: Ciphertext patterns for \( H = 3 \) and \( W = 6 \).

**Algorithm 6: Helper functions.**

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Function choose_plaintext(parity)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Function partial_attack(low, upp, ( p_t, \beta, \alpha, \gamma, \delta ) &amp; &amp; &amp; &amp; &amp;)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>for ( t \in {0, 256} ) do</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>for ( i \in {0, H} ) and ( j \in {0, W} ) do</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>( \alpha \leftarrow \max(0, i + j - (W - 1)) )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>if ( i + j \equiv \text{parity} \mod 4 ) then</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>( c_{i,j} \leftarrow t )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>else if ( i = \alpha ) then</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>( c_{i,j} \leftarrow 2 )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>else</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>( c_{i,j} \leftarrow 1 )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>Send the ciphertext ( c ) to the decryption oracle ( O_{dec} )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>Receive the plaintext ( \bar{p} ) from the decryption oracle ( O_{dec} )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>return ( p_t )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>for ( j \in {\text{low, upp}} ) and at each step increment ( j ) with 2 do</td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>( \alpha \leftarrow \max(0, j - (W - 1)); \beta \leftarrow \min(j, H - 1); )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>( \text{pos} \leftarrow 0 )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>for ( t \in {0, 256} ) do</td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>( \text{tab}<em>b p</em>{a+b-1, (\text{pos}+\gamma)} \leftarrow t )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>for ( x \in {0, 256} ) and ( y \in {0, 256} ) do</td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>( \text{ctr} \leftarrow 0 )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>( f \leftarrow (2 - \text{tab}<em>b p</em>{a+b-1, \text{pos}+\gamma}) \mod 256 )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>( g \leftarrow (1 - \text{tab}<em>b p</em>{a+b-1, \text{pos}+\gamma}) \mod 256 )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>if ( f = g ) then ( \text{ctr} \leftarrow \text{ctr} + 1 )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>if ( \text{ctr} = 256 ) then</td>
<td></td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>( k_{\text{pos}+\gamma, a+b-1, \text{pos}+\gamma} \leftarrow y )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>for ( t \in {0, 256} ) do</td>
<td></td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>( \text{tab}<em>b p</em>{a+b-1, \text{pos}+\gamma} \leftarrow (2 - \text{tab}<em>b p</em>{a+b-1, \text{pos}+\gamma}) \mod 256 )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>( \text{for} ( t \in {0, 256} ) do )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>( \text{ctr} \leftarrow \text{ctr} + 1 )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>( \text{pos} \leftarrow \text{pos} + 1 )</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Algorithm 7: CCA attack (randomly generated).

1 Function main_odd()
2 pt ← choose plaintext(0)
3 partial_attack(1, L, . . .) in this paper we showed that the actual security strength of Essaid et al.’s scheme is roughly 24 bits. To achieve an attack key odd()
4 pt ← choose plaintext(1)
5 partial_attack(2, L, . . .) in this
6 return k, ̃k, ̃k

Algorithm 8: More helper functions.

1 Function partial_attack(low, upp, pt, flag, &k, &ỹ, &ỹ)
2 for j in [low, upp] and at each step increment j with 2 do
3 α ← max(0, (j − (W − 1))); pos ← 0
4 for t ∈ [0, 256) do tab[i, α] = t;
5 tab[i, pos] = t
6 for x ∈ [0, 256] and y ∈ [0, 256) do
7 for i ∈ [0, 256) do : y ← (2 − tab[i] ⊕ x) mod 256
8 g ← (1 − tab[i]) mod 256
9 if f = g then ctr ← ctr + 1
10 if ctr = 256 then
11 for t ∈ [0, 256) do \( \delta_{p_{\text{tab}}[i]} \) ← (2 − tab[i] ⊕ x) mod 256
12 return \( \delta_{p_{\text{tab}}[i]} \)

4.2. Essaid et al.’s Generation Method for S-Boxes

In the case of Essaid et al.’s generation method it is enough to compute three s-boxes using the ideas from Examples 4.1 and 4.2. Then using similar techniques as the ones from Example 3.2 we can recover all the s-boxes, and implicitly all the key bytes.

The formal description of our chosen ciphertext attack is provided in Algorithm 9. The complexity of Algorithm 9 is the same as the complexity of Algorithm 5, namely \( O(2^{24} + 2^{8}L + HW) \). The only difference is that in the case of Algorithm 9 we need 512 oracle queries.

Similar to the case of the CPA attack, using 512 decrypted images and Algorithm 9, we managed to lower the security strength of Essaid et al.’s scheme from 128 bits to approximately 24 bits.

Algorithm 9: CCA attack (Essaid et al.’s generation method).

1 Function main_odd()
2 pt ← choose plaintext(0)
3 partial_attack(1, 4, p, 0, ̃k, ̃ỹ, ̃ỹ)
4 pt ← choose plaintext(1)
5 partial_attack(2, 3, p, 0, ̃k, ̃ỹ, ̃ỹ)
6 for i ∈ [0, 2] and j ∈ [0, 2) do
7 for i ∈ [0, 256) do \( f = 6_{\text{flag}}[i] \)
8 if f = \( \delta_{p_{\text{tab}}[1]} \) then \( \delta_{p_{\text{tab}}[2]} \) ← \( \delta_{p_{\text{tab}}[1]} \)
9 if f = \( \delta_{p_{\text{tab}}[1]} \) then \( \delta_{p_{\text{tab}}[2]} \) ← \( \delta_{p_{\text{tab}}[1]} \)
10 for pos ∈ [0, 2] do
11 for i ∈ [4, L] and t ∈ [0, 256) do \( \delta_{p_{\text{tab}}[1]}[t] \) ← \( \delta_{p_{\text{tab}}[1]}[t] \)
12 for i ∈ [4, L] and t ∈ [0, 256) do \( \delta_{p_{\text{tab}}[1]}[t] \) ← \( \delta_{p_{\text{tab}}[1]}[t] \)
13 for i ∈ [4, L] and t ∈ [0, 256) do \( \delta_{p_{\text{tab}}[1]}[t] \) ← \( \delta_{p_{\text{tab}}[1]}[t] \)
14 extract key(pos, k)
15 return k, ̃k, ̃k

5 CONCLUSIONS

In (Essaid et al., 2019), the authors described an image encryption scheme that they claimed provided a security strength of 128 bits. Unfortunately, in this paper we showed that the actual security strength of Essaid et al.’s scheme is roughly 24 bits.
our security bound, we devised a chosen plaintext attack which needs 256 queries to the encryption oracle. We also describe a chosen ciphertext attack which needs 512 queries to the decryption oracle and has a complexity of $O(2^n)$. For completeness, we also show how to attack Essaid et al.’s cryptosystem when all its s-boxes are randomly generated. In this case, using our CPA or CCA attacks, we lower the security strength to 36 bits.

REFERENCES


