We are IntechOpen, the world’s leading publisher of Open Access books
Built by scientists, for scientists

4,300
Open access books available

116,000
International authors and editors

125M
Downloads

154
Countries delivered to

TOP 1%
Our authors are among the most cited scientists

12.2%
Contributors from top 500 universities

WEB OF SCIENCE™
Selection of our books indexed in the Book Citation Index
in Web of Science™ Core Collection (BKCI)

Interested in publishing with us?
Contact book.department@intechopen.com

Numbers displayed above are based on latest data collected.
For more information visit www.intechopen.com
Fast Chaotic Encryption for Hyperspectral Images

Carlos Villaseñor, Javier Gomez-Avila, Nancy Arana-Daniel, Alma Y. Alanis and Carlos Lopez-Franco

Abstract

The information collected by hyperspectral images (HI) is essential in applications of remote sensing like object detection, geological process recognition, and identifying materials. However, HI information could be sensitive, and therefore, it should be protected. In this chapter, we show a parallel encryption algorithm specifically designed for HI. The algorithm uses multiple chaotic systems to produce a crossed multidimensional chaotic map for encrypting the image; the scheme takes advantage of the multidimensional nature of HI and is highly parallelizable, which leads to a time-efficient algorithm. We also show that the algorithm gets high-entropy ciphertext and is robust to ciphertext-only attacks.

Keywords: chaotic encryption, hyperspectral images, parallel computing

1. Introduction

Hyperspectral images (HI) or image spectrometry is a spectral sensing technique in which an object is photographed using several well-defined optical bands in the broad spectral range. This technique integrates imaging and spectroscopy to attain both spatial and spectral information from an object. It was originally developed on satellite and airborne platforms for remote sensing applications utilizing satellite imaging data of the earth and planets mostly; however, it has found application in diverse fields such as military defense, medical diagnosis, and agriculture [1].

HI are characterized by their spatial and spectral resolution. The spatial resolution measures the geometric relationship of the image pixels, and the spectral resolution determines the variations within image pixels as a function of wavelength. The HI has two spatial dimensions \( (m \text{ and } n) \) and one spectral dimension \( (l) \). The hyperspectral data are represented in the form of a 3D hyperspectral data cube as it is shown in Figure 1. Each slice of the cube along the spectral dimension is called band or channel.

HI could be made up of hundreds of contiguous bands for each spatial position. Consequently, each pixel in a hyperspectral image contains a spectrum representing the light-absorbing and/or scattering properties of the spatial region represented by that pixel. The resulting spectrum acts like a signature, which can be used to classify or estimate composition of the material it represents.

However, the information in the HI could be sensitive, and thus, it should be protected. There already exist many algorithms for secure encryption [2], like advanced encryption standard (AES) [3–5] based on irreducible polynomials in
Galois fields or the Rivest-Shamir-Adleman (RSA) algorithm [6] based on large prime number factorization. These algorithms work with raw data, and they offer a secure way to protect it. They work at the binary level of the data, and then they can deal with all kinds of data structures like images, videos, and documents.

The computational run-time of AES and RSA could be very high for a high volume of data. For this reason, in the last decades, the uses of chaos-based cryptography have become very popular [7]. The chaotic systems are deterministic dynamical systems which are sensitive to their initial conditions and parameters.

In this chapter, we present an extended revision of the parallel encryption algorithm based on chaotic systems for HI previously presented in [8]. This algorithm was specifically designed for HI, and it takes advantage of the spatial and spectral distribution to get fast encryption. The algorithm gets high-entropy ciphertext, and it is robust to ciphertext-only attacks.

The chapter is organized as follow: in Section 2, we briefly review the basics of HI and their applications. In Section 3, we explain the characteristics of the chaotic systems and why they are a suitable mathematical tool for encryption. In Section 4, we present our encryption scheme with full details of implementation (we include source code in Matlab and CUDA languages). In Section 5, we present new experimentation and results, and finally, in Section 6, we present our conclusions and future work.

2. Hyperspectral images

A digital image is a rectangular array of \( n \) rows and \( m \) columns where every element of the array, also called pixel, has an intensity value. New cameras allow us to have multiple measures in the same pixel of different wavelengths, where we represent this information by adding a new dimension \( l \) called channels.

The pixels are a discrete quantified measure of light; the intensity represented is an integer proportional to the number of photons detected. In a simple image, the intensity is represented with \( 2^8 \) precision, where 0 is no photons detected and 255 is the pixel saturated. Other systems use \( 2^{12}, 2^{14}, \) or \( 2^{16} \) levels of intensity [9].

Color images have three channels to represent visible light (400–800 nm of wavelength). This kind of camera uses filter for red, green, and blue light. When we have more of three channels, we called it multivariate images or multispectral images, for example, an image of 21 channels.

A common convention is to call hyperspectral images to images with more than 100 channels [9]. Every pixel of a HI has its spectral representation of that position;
classical spectrophotometers use a light source, a filter system to disperse the light into their respective wavelengths, and a detection system. The HI has commonly a spectral resolution of 10–20 nm.

HI have many applications such as oil detection [10], earth remote sensing [11], vegetation and water studies [12, 13], archaeology and art conservation [14–16], finding minerals and other materials [17], medical diagnosis [18–20], food quality and safety control [18, 21, 22], and crime scene analysis and forensic traces [23, 24].

3. Chaotic systems and chaotic encryption

Chaotic systems are dynamical systems that for a certain range of initial conditions show chaotic behavior. This chaotic behavior has the following features:

- Sensitive to initial conditions: the chaotic system is exponentially sensitive to small changes in the initial conditions. These small changes could produce a big difference in the system output.

- Bounded: the states with chaotic behavior have bounded limits.

- Deterministic: a chaotic system does not have randomness, so if we have the initial conditions and parameters, we can simulate the system.

- Aperiodic: there is no periodicity in the chaotic behavior.

The chaotic behavior is a deterministic phenomenon that seems random for a viewer that does not know the initial conditions and parameters of the system; but like any deterministic system, we can obtain the same time-series output by simulating the system with the same initial condition and parameters.

These features make the chaotic system a suitable platform for encryption. We have two principal ways of implementing chaotic encryption:

- Chaotic encryption by synchronization phenomenon: in the case of continuous signals, they could be encrypted by adding a chaotic continuous signal. For the decrypt process, we synchronize another chaotic system throughout the synchronization phenomenon and control theory to subtract the chaotic signal [25, 26].

- Simulating discrete chaotic system: it consists in generating a chaotic mask and mixing with a discrete plaintext with an involution operation. For decrypt, we generate the same mask, and by properties of the involutions, we recover the original signal. In this chapter, we concentrate in the second alternative.

Many chaotic systems have been used for different encryption problems like:

- Stream ciphers: Lorenz system has been used as a stream cipher in [27].

- Image encryption: for image encryption is common in using chaotic maps like Ikeda chaotic map [28], logistic map, tent map, quadratic map, and Bernoulli map [29], a survey is presented in [30].

- Video encryption: the chaotic maps also are used to encrypt video in [31–33].
For the proposed encryption scheme, we use the piecewise linear chaotic maps (PLM) described by Eq. (1), where $x_0 \in (0, 1)$ is the initial condition and $\mu \in (0, 0.5)$ is the parameter of the system. In Figure 2, we present a simulation of the PLM system.

$$x_{n+1} = \frac{x_n - \left\lfloor \frac{x_n}{\mu} \right\rfloor \mu}{\mu}$$ \hspace{1cm} (1)

4. Chaotic systems for parallel encryption of hyperspectral images

In this section we will review the proposed chaotic encryption scheme for HI presented in [8]. The proposed scheme is shown in Figure 3. We propose to use four chaotic systems for encrypting HI. The first three chaotic systems are used to generate a chaotic mask of the same size of the HI, and the last one is used to generate a chaotic substitution box (S-box) according to the algorithm in [34].

The key is formed with initial conditions $x_i$ and parameters $\mu_i$ of the four PLM chaotic systems. It is important to notice that this scheme can be implemented with other discrete chaotic systems. These systems are simulated in parallel to create four different chaotic signals denoted by $c_1(t)$, $c_2(t)$, $c_3(t)$, and $c_s(t)$ that are saved as the key expansion.

In Figure 4, we show the source code of the parallel implementation of the chaotic system in the CUDA programming language, where the variable ite represents the maximum number of iterations of the chaotic system, $x_{vec}$ is a vector with the initial condition parameters, $mu_{vec}$ is a vector with the parameters of the PLM systems, and $C$ is an array where the chaotic signals are stored.

The chaotic signal $c_s(t)$ is used to generate a chaotic S-box [34]. An S-box is a basic component for symmetric key ciphers because it is a reversible operation for confusion introduction. Confusion is a property of a secure cipher denied by Claude Shannon, and it means that each basic unit of the ciphertext should depend on several parts of the key hiding the relationship between them. The S-box is a permutation which is commonly implemented with a look-up table that maps from a byte to another byte.

In Figure 5, we present the Matlab code for the generation of the chaotic S-box and its inverse.

Using the previous codes, we can encrypt the HI by using Eq. (2), where every pixel in the direction $(i, j, k)$ is encrypted in parallel by composing a chaotic signal with the $c_1(t), c_2(t), c_3(t)$ chaotic signals. We combine them by using the XOR operation.
operation which is an involution. Afterwards we use the chaotic S-box to include confusion in the signal:

\[ I_{c}(i,j,k) = S_{\text{box}}[c_3(k) \oplus c_2(j) \oplus c_1(i) \oplus I(i,j,k)] \]  \( (2) \)

Eq. (2) is implemented in the CUDA language as it is shown in Figure 6.

To use the chaotic and encrypted function from Matlab, we compile from the terminal, in this case, using nvcc compiler with the lines in Figure 7:
In Figure 8, we show the code in Matlab for encrypting HI in parallel. Notice that the first step in Figure 8 is to load the HI in the variable I in the GPU memory (Igpu), and then we calculate the chaotic S-box in the variable S with the code in Figure 5; after that, we simulate the chaotic systems in the variable Cgpu with the code in Figure 4. Finally, we encrypt the HI with Figure 6 gathering the result in the Ic variable.

The decryption scheme of the HI is almost symmetrical. This could be achieved with Eq. (3) which is implemented in Figure 9. The code in Figure 9 is very close to the one in Figure 6, but notice that the inverse of the S-box is used and the XOR of the chaotic signal are in inverse order:

\[ I_{i, j, k} = c_1(i) \oplus c_2(j) \oplus c_3(k) \oplus S^{-1}_{box}[I_c(i, j, k)] \]  \( \text{(3)} \)

Using Eq. (2) in Eq. (3) we obtain the development in Eqs. (4)–(6) that shows how the S-box is canceled with the inverse S-box and how the chaotic signals are canceled because the XOR operation is an involution:

\[
\text{function} \ [S, \text{invS}] = \text{Chaotic\_SBox}(x, \mu)
\]

\[
S = \text{zeros}(16*16,1);
\]

\[
\text{invS} = \text{zeros}(16*16,1);
\]

\[
Q = 0:255;
\]

\[
\text{max} = 2^{32};
\]

\[
\text{for} \ i = 0:255
\]

\[
x = \text{PLM}(x, \mu);
\]

\[
d = \text{mod}(\text{floor}(\text{max} \times x), \ 256);
\]

\[
j = \text{mod}(d, 256 - i);
\]

\[
S(i+1) = Q(j + 1);
\]

\[
\text{invS}(Q(j+1) + 1) = i;
\]

\[
Q(j+1) = Q(256-i);
\]

\[
\text{end}
\]

\[
\text{end}
\]

Figure 5.
Chaotic substitution box.

\[
\text{Figure 6.}
\]
Parallel chaotic encryption (Encrypt.cu file).

\[
\text{nvcc -ptx Encrypt.cu -o cuda} \ C:\Program Files\Microsoft Visual Studio 12.0\VC\bin\%
\text{nvcc -ptx Chaos.cu -o cuda} \ C:\Program Files\Microsoft Visual Studio 12.0\VC\bin\%
\]

Figure 7.
Compilation of the CUDA files.

In Figure 8, we show the code in Matlab for encrypting HI in parallel. Notice that the first step in Figure 8 is to load the HI in the variable I in the GPU memory (Igpu), and then we calculate the chaotic S-box in the variable S with the code in Figure 5; after that, we simulate the chaotic systems in the variable Cgpu with the code in Figure 4. Finally, we encrypt the HI with Figure 6 gathering the result in the Ic variable.

The decryption scheme of the HI is almost symmetrical. This could be achieved with Eq. (3) which is implemented in Figure 9. The code in Figure 9 is very close to the one in Figure 6, but notice that the inverse of the S-box is used and the XOR of the chaotic signal are in inverse order:

\[ I_{i, j, k} = c_1(i) \oplus c_2(j) \oplus c_3(k) \oplus S^{-1}_{box}[I_c(i, j, k)] \]  \( \text{(3)} \)

Using Eq. (2) in Eq. (3) we obtain the development in Eqs. (4)–(6) that shows how the S-box is canceled with the inverse S-box and how the chaotic signals are canceled because the XOR operation is an involution:
\[ I_{i, j, k}(\theta) = c_{1}(i) \oplus c_{2}(j) \oplus c_{3}(k) \oplus S_{box}^{-1}[S_{box}[c_{3}(k) \oplus c_{2}(j) \oplus c_{1}(i) \oplus l(i, j, k)]] \] (4)

\[ I_{i, j, k} = c_{1}(i) \oplus c_{2}(j) \oplus c_{3}(k) \oplus c_{3}(k) \oplus c_{2}(j) \oplus c_{1}(i) \oplus l(i, j, k) \] (5)

5. Experiments and results

In this section, we present the result of six new experiments. In Table 1, we report the references and the size of the HI.
In the first experiment, we present the results of encrypting the “San Francisco” image. In Figure 10, we present a comparison of the ninety third channel of the original image and the cipher one. The image was loaded to the GPU in 0.4439 s and encrypted in 0.0094 s and then downloaded to the CPU memory in 0.2158 s. The original image has an entropy of 6.1057; after the encryption, we get an entropy of 7.9985. In Figure 11, we show a comparison of the hyper-histograms of the original and encrypted image.

In the second experiment, we present the results of encrypting the “Urban” image. In Figure 12, we present a comparison of the sixtieth channel of the original image and the cipher one. The image was loaded to the GPU in 0.0118 s and encrypted in 0.0023 s and then downloaded to the CPU memory in 0.0435 s. The original image has an entropy of 6.7452; after the encryption, we get an entropy of 7.9987. In Figure 13, we show a comparison of the hyper-histograms of the original and encrypted image.
In experiment 3, we present the results of encrypting the “Indian pine” image. In Figure 14, we present a comparison of the forty-fifth channel of the original image and the cipher one. The image was loaded to the GPU in 0.0035 s and encrypted in 0.0010 s and then downloaded to the CPU memory in 0.0124 s.

The original image has an entropy of 6.3325; after the encryption, we get an entropy of 7.9985. In Figure 15, we show a comparison of the hyper-histograms of the original and encrypted image.

In experiment 4, we present the results of encrypting the “Jasper Ridge” image. In Figure 16, we present a comparison of the forty seventh channel of the original image.
image and the cipher one. The image was loaded to the GPU in 0.0020 s and encrypted in 0.009 s and then downloaded to the CPU memory in 0.0055 s.

The original image has an entropy of 6.8918; after the encryption, we get an entropy of 7.9983. In Figure 17, we show a comparison of the hyper-histograms of the original and encrypted image.

In experiment 5, we present the results of encrypting the “University of Pavia” image. In Figure 18, we present a comparison of the 62nd channel of the original image and the cipher one. The image was loaded to the GPU in 0.0122 s and encrypted in 0.0026 s and then downloaded to the CPU memory in 0.0545 s.
The original image has an entropy of 6.6827; after the encryption, we get an entropy of 7.9986. In Figure 19, we show a comparison of the hyper-histograms of the original and encrypted image.

In experiment 6, we present the results of encrypting the “Samson” image. In Figure 20, we present a comparison of the hundredth channel of the original image and the cipher one. The image was loaded to the GPU in 0.0013 s and encrypted in 0.0007 s and then downloaded to the CPU memory in 0.0036 s.
The original image has an entropy of 6.5833; after the encryption, we get an entropy of 7.9981. In Figure 21, we show a comparison of the hyper-histograms of the original and encrypted image.

In Table 2, we summarize the results of the encryption. Notice that the encryption process results in high entropy close to eight (the maximum for $2^8$ representation). The entropy is a measure of uncertainty defined in Eq. (7), where $s_i$ is a symbol or codification and $P(s_i)$ is the probability of $s_i$ to appear. The number of bits for the representation, in this case eight, is depicted by b:
The proposed algorithm takes advantage of the multidimensionality nature of the HI. This leads to a high-performance algorithm that can encrypt images quickly. Notice that the load to GPU and download to CPU processes take most of the time. This could be avoided with a direct acquisition of the HI from the GPU, but this problem is beyond the scope of this chapter.

\[ H(s) = - \sum_{i=0}^{2^b-1} P(s_i) \log_2 (P(s_i)) \]  

Figure 19.
Experiment 5: hyper-histogram comparison. a) Original Hyperhistogram. b) Encrypted Hyperhistogram.

Figure 20.
Experiment 6: image comparison. a) Original image. b) Encrypted image. c) Original histogram. d) Encrypted histogram.
6. Conclusions and future work

In this chapter, we have reviewed the proposed method in [8] with a full review of the source code. The proposed algorithm is a parallel encryption scheme specific for hyperspectral images. This scheme takes advantage of the multidimensionality nature of the HI, where every pixel of every channel is encrypted in parallel. These features lead to a high-performance algorithm capable of encrypting HI in a short time.

The algorithm uses four chaotic systems, the first three for generating a chaotic cube mask of the same size of the HI and the fourth one to generate a chaotic S-box to aggregate confusion in the ciphertext. As the experimentation has shown, the algorithm can generate high-entropy ciphertext. This algorithm is recommended for fast encryption in ciphertext-only attacks.

As future work, most of the computational time is wasted in the load to the memory of the HI image. A system to load directly to the GPU memory is desirable. Also, we recommend the readers to explore new ways of combining different chaotic signals for high-dimensional masks. Furthermore, an extension of this algorithm for working with a compatible compression algorithm is desired.

<table>
<thead>
<tr>
<th>Number</th>
<th>Name</th>
<th>Load (s)</th>
<th>Encrypt (s)</th>
<th>Download (s)</th>
<th>Total (s)</th>
<th>Original entropy</th>
<th>Encrypted entropy</th>
<th>Decrypt (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>San Francisco</td>
<td>0.4439</td>
<td>0.0099</td>
<td>0.2158</td>
<td>0.6696</td>
<td>6.1057</td>
<td>7.9985</td>
<td>0.0095</td>
</tr>
<tr>
<td>2</td>
<td>Urban</td>
<td>0.0118</td>
<td>0.0023</td>
<td>0.0435</td>
<td>0.0576</td>
<td>6.7452</td>
<td>7.9987</td>
<td>0.0031</td>
</tr>
<tr>
<td>3</td>
<td>Indian pine</td>
<td>0.0035</td>
<td>0.0010</td>
<td>0.0124</td>
<td>0.0169</td>
<td>6.3325</td>
<td>7.9985</td>
<td>0.0012</td>
</tr>
<tr>
<td>4</td>
<td>Jasper Ridge</td>
<td>0.0020</td>
<td>0.0009</td>
<td>0.0055</td>
<td>0.0084</td>
<td>6.8918</td>
<td>7.9983</td>
<td>0.0009</td>
</tr>
<tr>
<td>5</td>
<td>University of Pavia</td>
<td>0.0122</td>
<td>0.0026</td>
<td>0.0545</td>
<td>0.0693</td>
<td>6.6827</td>
<td>7.9986</td>
<td>0.0023</td>
</tr>
<tr>
<td>6</td>
<td>Samson</td>
<td>0.0013</td>
<td>0.0007</td>
<td>0.0036</td>
<td>0.0056</td>
<td>6.5833</td>
<td>7.9981</td>
<td>0.0008</td>
</tr>
</tbody>
</table>

Table 2. Encryption results.

Figure 21. Experiment 6: hyper-histogram comparison. a) Original Hyperhistogram. b) Encrypted Hyperhistogram.
Fast Chaotic Encryption for Hyperspectral Images
DOI: http://dx.doi.org/10.5772/intechopen.88980

Author details
Carlos Villaseñor, Javier Gomez-Avila, Nancy Arana-Daniel*, Alma Y. Alanis
and Carlos Lopez-Franco
Centro Universitario de Ciencias Exactas e Ingenierías, Universidad de Guadalajara,
Guadalajara, Mexico

*Address all correspondence to: nancyaranad@gmail.com

© 2019 The Author(s). Licensee IntechOpen. This chapter is distributed under the terms
of the Creative Commons Attribution License (http://creativecommons.org/licenses/by/3.0),
which permits unrestricted use, distribution, and reproduction in any medium,
provided the original work is properly cited.
References


[18] Carrasco O, Gomez RB, Chainani A, Roper WE. Hyperspectral imaging
applied to medical diagnoses and food safety. In: Geo-Spatial and Temporal Image and Data Exploitation III. 2003


