# Exploring a Novel Cryptographic Solution for Securing Small Satellite Communications

Samuel Jackson[1], Jeremy Straub[2], and Scott Kerlin[3]
*(Corresponding author: Jeremy Straub)*

Department of Computer Science, Oklahoma State University[1]
219 MSCS; Stillwater, Oklahoma 74078, USA
Department of Computer Science, North Dakota State University[2]
1320 Albrecht Blvd, Room 258; NDSU Dept 2740; Fargo, ND 58102, USA
(Email: jeremy.straub@ndsu.edu)
Department of Computer Science, University of North Dakota[3]
3950 Campus Road, Grand Forks, ND 58202, USA
*(Received May 1, 2017; revised and accepted Aug. 26, 2017)*

## Abstract

The need for encryption for use onboard satellites is a growing issue. While larger and modern satellites may have the hardware capabilities to use common terrestrial encryption schemes, smaller and older satellites may lack such capabilities. Small satellites are becoming increasingly important, and securing their communications a necessity. The Department of Defense, for example, is developing CubeSats, a type of small satellite, for use in operations. Despite the growing need, currently, there is no agreed upon encryption algorithm for these devices, which have limited hardware capabilities and physical space. This paper presents and characterizes a novel algorithm that uses chaos theory to encrypt data. Specifically, a proof-of-concept of this novel algorithm is presented. It is then compared against AES and SPECK in terms of speed of encryption and decryption.

*Keywords: Chaotic Cryptosystem; CubeSat; Small Spacecraft*

## 1 Introduction

Encryption for use onboard satellites is an open research problem. While larger, modern satellites may be able to employ common terrestrial encryption schemes, small satellites and older larger satellites may not have the requisite hardware capabilities. Because of the radio-based transmission medium used, spacecraft communications have no inherent physical security mechanism and thus have a particular need for other security mechanisms. The Advanced Encryption Standard (AES), a standard technique used by the NSA and others for securing data transmissions, could potentially be brought to bear on this challenge.

Muhaya [21], in particular, considered the use of multiple standard techniques. While determining that AES was a component of the solution, he demonstrated the utility of and discussed the need for enhancing the AES cryptographic algorithm with a chaotic pixel shuffling mechanism. However, this prior work failed to evaluate the computational costs of typical approaches and the proposed hybrid approach. Knowledge of this is, of course, critical to determining their suitability for use on a small or older spacecraft.

The lack of a standard, usable cryptographic algorithm to secure communications to and from spacecraft is a growing concern, as programs such as NASA's Educational Launch of Nanosatellites [25] program, the University NanoSat Program [15] and the European Space Agency's Fly Your Satellite [11] program promote the building and launching of CubeSats. While not all CubeSats require encryption (and some may be precluded from its use due to FCC restrictions on amateur licenses [27]), many future CubeSats plan to incorporate propulsion (see, *e.g.*, [20]), making their potential comprise problematic and driving a need for encryption. The growth of the use of small satellites for military applications (see, *e.g.*, [1, 29]) also drives the need for suitable cyptographic technologies for use on small satellites.

The challenge of securing communications is not limited to small satellites. In 2014, NOAA's Satellite network was hacked, shutting down the network for two days. This satellite data is vital to many different applications, ranging from providing weather forecasts to national security applications. The same lightweight approaches used for small spacecraft may also be appropriate for older satellites, which may have limited capabilities (as compared to newer models).

This paper investigates a potential solution that would

allow CubeSats, other small spacecraft, and older spacecraft with limited resources to encrypt and decrypt data within their hardware capabilities or transmission time requirements. Specifically, this paper presents a proof-of-concept for a novel cryptographic algorithm, implemented in software, which uses chaos-theory to encrypt data. The algorithm, invented by Huang, Ye, and Wong [13] for the use of encrypting images, has been modified to turn it into a block cipher that can be used to encrypt any data. The algorithm has been tested and compared against other, more common algorithms in terms of encryption and decryption speed. In line with Muhaya's prior work [21], the use of the proposed algorithm on its own as well as its use (and the use of SPECK) to augment AES are also considered.

This paper continues with a discussion on the challenges of the space environment that prompt the need for this technology. Then, prior work on block ciphers and encryption for use onboard spacecraft is reviewed. This is followed by the presentation of the proposed algorithm. The experiments used to validate the algorithms functionality, characterize its performance across multiple block sizes, and compare it to alternate techniques are then presented. Finally, this data is analyzed, before concluding.

# 2 Background

This section provides background information related to prior work in the fields of small satellites and cryptography. First, an overview of challenges to cryptography specific to operating in space is provided. This is followed by an overview of block ciphers. Finally, prior work on encryption for use in space is discussed.

## 2.1 Challenges of the Space Environment

The space environment presents a number of challenges relative to security. The first is the lack of any appreciable physical security for ground-to-space and space-to-ground communications. All of the foregoing occurs over radio frequency transmissions and, thus, can easily be intercepted and possibly jammed or even manipulated by an adversary. State actors, in particular, may have the capability of placing another craft (either aerial or in a lower orbit) in between the transmitting and receiving station, allowing them to perform a man-in-the-middle style attack. A variety of security techniques are needed to protect against this and similar scenarios (see, *e.g.*, [28]). Encryption is only a small portion of this challenge; however, it is critical in protecting sensitive commands (which might reveal the tactics or other plans of a spacecraft controller) and data.

Spacecraft, and in particular small spacecraft, have limited mass and volume available. Wertz *et al.* [30] and Fortescue *et al.* [12] discuss these constraints and the deliberate trade-off process that is made to determine what will be included and what must be excluded from a spacecraft design. These constraints may make a dedicated encryption hardware system impractical and also limit the computational capabilities available onboard the spacecraft.

Beyond the limitations posed to what hardware can be included, additional restrictions exist. Both power and communications time are at a premium. Thus, encryption techniques must be as fast and lightweight as possible to maximize the available processing time and capabilities. The amount of data overhead imposed by the technique must also be kept to a minimum.

## 2.2 Prior Work on Block Ciphers

The Data Encryption Standard (DES) was one of the first block ciphers widely used and accepted as a standard. It was created by IBM, in cooperation with the National Security Agency (NSA), and published in 1975 [26]. This algorithm has been adapted multiple times as computers have become more powerful and avenues of attack were discovered.

In 2001, an Advanced Encryption Standard (AES) was published and superseded DES [26]. It was created by Vincent Rijmen and Joan Daemen, and is currently listed in NSA Suite B of cryptographic algorithms for use in encryption. It is a block cipher with a block size of 16 bytes, and variable key sizes ranging from 16, 24, or 32 bytes. AES is a symmetric standard, which uses the same keys to encrypt and decrypt data. Prior work has evaluated the performance of AES and other symmetric algorithms [10] and its power consumption when processing different types of data [19].

More recently, a new field of cryptographic algorithms has been investigated, namely lightweight cryptographic algorithms. While these may be less secure than other, heavier algorithms, they are created with the goal of being used on systems with limited resources, such as sensor networks or RFID tags. A family of lightweight algorithms was recently published by the NSA in 2013 [3]. Known as the SIMON and SPECK family of algorithms, SIMON was optimized for hardware and SPECK for software. Both families have been proposed as the standard for lightweight algorithms, although more testing is needed to prove their security.

## 2.3 Prior Work on Encryption for Space

Quantum Cryptography is a field of cryptographic research that is being investigated for use in satellites. Hughes *et al.* [14] investigated the technique of using Quantum cryptography to securely generate keys for use on either ground station/satellite communications or satellite/satellite communications. Their tests were successful. Rarity, Tapster, Gorman and Knight [23] also investigated the possibility of using Quantum Cryptography to create a secure key exchange technique between a ground station and a satellite, and their work suggests

that there are no technical obstacles to building such a system, within their technical specifications.

In 2011, Challa, Bhat and Mcnair [4] proposed a security solution for CubeSats called CubeSec and GndSec. Their proposed scheme involved using AES and DES encryption operating in Galois/Counter mode on hardware that supports AES and DES encryption. Specifically, they tested their algorithm on an ATXMega128 microcontroller, and achieved throughput of between 43 KBps and 256Kbps depending on the configuration. However, they did not investigate a solution based solely on software implementation.

As mentioned previously, Muhaya investigated the possibility of combining AES with a chaotic encryption scheme [21]. In his work, Muhaya looked at the possibility of using an Arnold Cat Map [22] to shuffle pixel values, and using a Chaotic Henon Map [7] to generate a random sequence of key values for the AES algorithm. However, while the results listed were promising, no analysis was performed looking at either the performance or security of AES or the chaotic algorithms operating separately.

Encryption techniques used in space are well-served by making use of ongoing development of terrestrial protocols and advancements. Attackers are, similarly, able to make use of Earth-based advances in protocol-cracking techniques, driving a need for spacecraft developers to stay current. Advances may come from areas such as sensor networks [18], the encryption of specific types of data (such as images [9, 16]) or supporting technologies (such as seed generators [2]).

# 3 Proposed Technique

This section presents the proposed novel algorithm, based on the prior work of [13]. First, a general overview is presented. Then, specific elements of the approach are discussed, including the use of the Lorenz system, key generation, diagonal and anti-diagonal permutation, and block-based diffusion.

## 3.1 Overview

The first step of the proposed algorithm, which is based on prior work [13], is to read the data into an $n \times n$ matrix. The data is then permuted along the diagonal and anti-diagonal lines. This randomizes the location of each pixel throughout the matrix. Then, block-based diffusion is performed on the matrix. As adjacent pixels normally have a high correlation with their neighbors, block-based diffusion helps remove this correlation. A general diagram of the novel algorithm is shown below.

## 3.2 Lorenz System and Key Generation

The algorithm uses the Lorenz system of equations (see [6] for a more detailed discussion of the Lorenz system) to generate several values used in both the Diagonal/Anti-diagonal permutation step and the Block-based diffusion step. The Lorenz system of equations is as shown below.

$$\begin{aligned} \dot{x} &= m(y-x)+u_1 \\ \dot{y} &= rx-y-xz+u_2 \\ \dot{z} &= xy-bz+u_3 \end{aligned} \tag{1}$$

Where $m, r$, and $b$ are constants and $x, y$, and $z$ are initial values. For values $m = 10$, $r = 28$, $b = 8/3$, and $u_1 = u_2 = u_3 = 0$, the Lorenz system is in a chaotic state, which in essence means that it will diverge from another Lorenz System with similar starting values. In other words, given a set of $\{x_1, y_1, z_1\}$ similar to $\{x, y, z\}$, the two systems will soon look completely different. Note that in [13], they use a Time-delay Lorenz System, as seen below:

$$\begin{aligned} \dot{x} &= m(y-x)+u_1 \\ \dot{y} &= rx-y-xz+u_2 \\ \dot{z} &= xy-bz(t-)+u_3 \end{aligned} \tag{2}$$

For the sake of simplicity, in this paper the algorithm uses the regular set of Lorenz Equations shown in Equation (1). Using this simpler set of equations preserves the essence of the original algorithm while making it easier for the reader to understand. It also offers prospective speed benefits, as well as making the implementation and testing of the proposed algorithm simpler.

Given the Lorenz system, initial values $x, y$, and $z$ are chosen (as discussed in [13]). These are the secret key set. As mentioned above, given enough time, the Lorenz system will diverge from Lorenz systems with similar starting values. Hence, the system of equations is iterated some number $p$ times to preserve this quality and to give sufficient time for the system of equations to diverge. In [13], $p$ was chosen to be equal to 30. In this paper, $p$ was similarly chosen to be 30. Next, the system is iterated an additional $n$ amount of times, where $n$ is equal to the length of one side of the $n \times n$ data matrix. This generates the values used in the Diagonal/Anti-diagonal permutation step and the Block-based diffusion step.

Each $x, y$, and $z$ value set generated is stored in an array with each other $x, y$, or $z$ value. At the end of this process, the algorithm has generated three arrays of length $n$, each array storing either the $x$ values, the $y$ values, or the $z$ values. As each $x, y$, and $z$ values in a given successive sequence are generally increasing or decreasing, each $x, y$, and $z$ value are processed via the equation:

$$m = abs(m * 10^3) - floor[abs(m * 10^3)] \tag{3}$$

Where abs(a) returns the absolute value of a, and floor(b) returns the nearest integer less than or equal to m. Note that this differs from the original equation found in [6], which is:

$$m = abs(m * 10^{14}) - floor[abs(m * 10^{14})]. \tag{4}$$

At this point, as was done in [13], the array that holds the x values are copied into a second array and sorted.

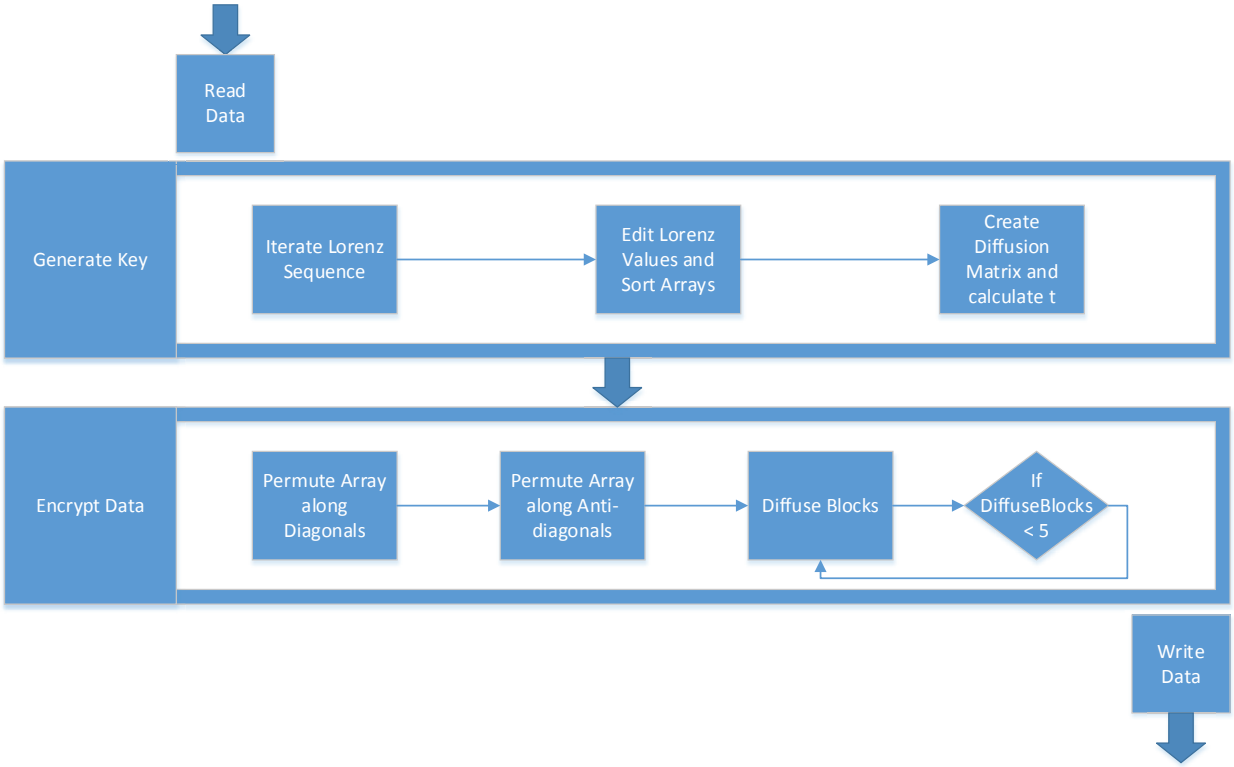


Figure 1: A general outline of the steps taken during the encrypting of data

This then produces two arrays, one with the $x$ values in their original positions, and one with the $x$ values in sorted order. Both of these arrays will be used in the diagonal/anti-diagonal permutation steps. Similarly, the y values are processed so that there are now two arrays, one which holds the y values in their original position and one which holds the y values in sorted order.

### 3.3 Diagonal and Anti-diagonal Permutation

The algorithm, as was performed in [13], now starts the process of permuting the matrix of data along the diagonal and anti-diagonal lines. First, it is important to note that each diagonal line in the matrix has a different number of elements. For example, the diagonal which reaches from the top left corner down to the bottom right corner holds $n$ elements, while the diagonal to the right holds $n-1$ elements. This can be fixed by patching each diagonal with another diagonal so that, together, they have $n$ elements. In the above example, the diagonal with $n-1$ elements is patched with either the element in the top right or bottom left corner (which is in a diagonal of 1). Thus, together they make a diagonal with $n$ elements. By doing this to each diagonal, it is now possible to permute each diagonal symmetrically.

To permute the diagonal line, the proposed algorithm uses the two arrays of $x$ values (sorted and unsorted) produced in the key generation step. Consider a given element in the unsorted array. There is a similar element in the sorted array, however, the position is different. By looking at the difference in positions, it is possible to generate a key that will permute a given diagonal. Consider the case where each pixel on a given diagonal is mapped to an element in the unsorted array of $x$ values. For example, the first pixel is mapped to the first element of the unsorted array of $x$ values, the second pixel is mapped to the second element, and so on. With this mapping complete, it becomes possible to permute the pixels by simply matching them with the position of their assigned value of $x$ in the array of sorted $x$ values. Permutation in the anti-diagonal direction is performed similarly, except the arrays of sorted and unsorted y values are used.

During the block-based diffusion step, the original values of each pixel are modified using the diffusion matrix calculated from the values iterated from the Lorenz sequence. The goal of these steps are to harden the data against known plaintext attacks, as well as reduce correlation between pixels that were adjacent in the source image [13].

### 3.4 Block Based Diffusion

This step, based on [13], begins by breaking the $n \times n$ matrix into two matrices of size $n/2$ by $n$, which will be called matrices A and B. If $n$ is an odd number, the

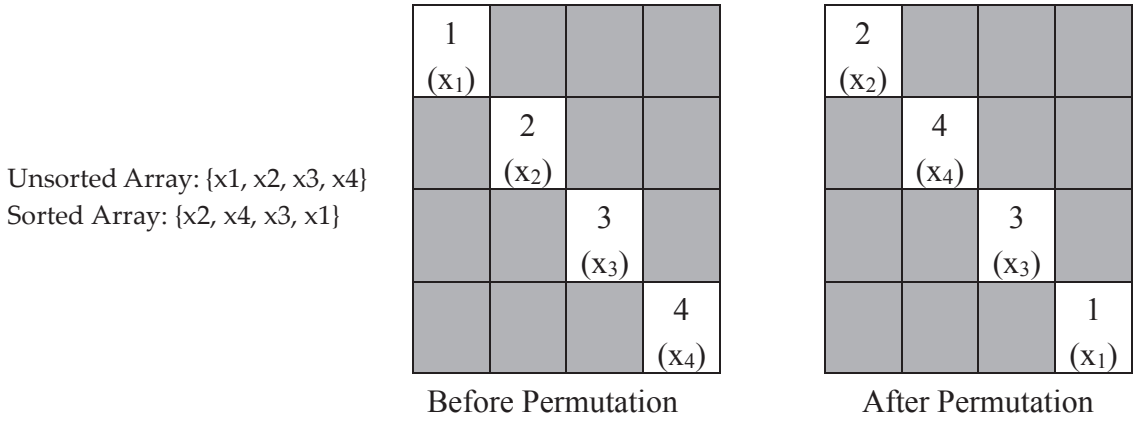


Figure 2: An example of performing permutations along a diagonal line [17]

data will need to be padded by adding one more row to make $n$ even. Next, $n^2/2$ values must be selected from the arrays of X, Y, and Z values randomly. These will form a diffusion matrix which will be applied to $a$. The diffusion matrix will be called D. It is important to note that for $n > 6$, more values must be selected than there are unique elements in X, Y, and Z. In addition to creating the diffusion matrix, a value t is calculated:

$$t = (\Sigma B) \mod (\frac{n}{2}) + (\Sigma B) \mod (n) + (\Sigma B) \mod (\frac{n^2}{2}) + 1. \quad (5)$$

where Σ B is the sum of the elements in B, and (a) mod (b) applies the modulo(b) operator to $a$. The matrix A is then diffused via the following equation, where G is the resulting matrix and $G_{i,j}$ is a position in matrix G:

$$G_{i,j} = (A_{i,j} + (t * W_{i,j})) \mod (256). \quad (6)$$

The matrix B is diffused with the following equation, where H is the resulting matrix and $H_{i,j}$ is a position in matrix H:

$$H_{i,j} = (B_{i,j} + G_{i,j}) \mod (256). \quad (7)$$

While the algorithm discussed in [13] was originally created to encrypt pictures, and hence deals with pixels and pixel values the proposed algorithm has been extended to support any type of data that can be placed in an $n \times n$ matrix. This makes it possible to use the algorithm as a general block-based cipher, and allows for easier comparison with common algorithms.

# 4 Experimental Methods and Results

The process of testing the proposed cryptographic algorithm and characterizing its performance relative to other, more common algorithms is now presented. The goal of this work is to determine what techniques would be well-suited for use in the domain of small satellites. As such, there are several constraints that limit possible solutions, which were discussed previously. It is important that algorithms not be inefficient in terms of processing time or overall data size. Thus, algorithms that double or triple the size of the ciphertext, as compared to the plaintext, would be inefficient in this environment.

Two other cryptographic algorithms were chosen to compare the proposed algorithm to. The first algorithm chosen was the Advanced Encryption Standard (AES). Specifically, AES operating in Electronic Codebook (ECB) mode was chosen to compare against the novel algorithm for several reasons. Firstly, being a block cipher with a block size of 16 bytes, it is easy to compare against the proposed algorithm. In addition, it is a common protocol for terrestrial computing and arguably could be a first choice for use on most platforms, including small satellites. ECB mode of operation was chosen because it appeared to operate most closely to the proposed algorithm, and thus made for balanced testing. Note that AES operating in ECB mode of operation is known to be insecure [6]. However, similar techniques to those used in the other forms of AES could prospectively be used (with similar performance and other impacts) to increase the security of the proposed algorithm.

The other algorithm that was chosen was SPECK. SPECK is a relatively new lightweight encryption algorithm that was created and proposed by the National Security Agency. While new, it appears to be secure, and may be a seen as a possible standard Lightweight Encryption approach [6]. Specifically, Speck (128/128) was chosen for testing, for several reasons. Firstly, SPECK (128/128) is also a block cipher with a block size of 16 bytes. In addition, lightweight algorithms are increasing in popularity and also have the potential to be used in the domain of small satellites. Hence, it is important to consider how the novel algorithm compares to a lightweight

algorithm.

The tests were run on a Raspberry Pi Model B. The code of the proposed algorithm, as well as the cryptographic algorithms with which the proposed algorithm was compared to, were written in the C++ programming language. AES code was implemented with the use of the Crypto++ Library version 5.6.2.

### 4.1 Experimental Setup

This section provides an overview of the configuration and design choices made in implementing the experimental design. Several areas are now discussed.

In the block-based diffusion step, a number of elements iterated from the Lorenz System must be chosen to fill a diffusion matrix. In the implementation presented in this paper, this was achieved by using a pseudo-random number generator with a seed based on the current time. To generate the seed, both the encryption and decryption algorithms have a file that stores the current seed. When the algorithm encrypts a file, it encrypts the current time as well and sends it with the ciphertext. It also stores this new value in the local file, overwriting the previous seed. When the decryption program decrypts the ciphertext, it then reads the new random number seed that was sent and overwrites the old seed. Both algorithms use the last known random seed when encrypting and decrypting, and with each transmission creates a new seed to use. It is important to note that by using this approach, the seed is overwritten with each transmission. Hence, if one transmission is not sent or received properly, it may create a scenario where all future transmissions are unrecoverable. This is an issue that will need to be addressed to facilitate the use of this algorithm in a real-world environment.

Additionally, the internal clock on a Raspberry Pi resets every time the system restarts. Even if the system is designed to never shut down, unexpected restarts may happen. This means that random number seeds may not be unique, given enough time. One potential solution to this is to set the current time from an external source, such as a GPS, or to send an updated time via the ground station during transmission, minimizing the amount of time that the Pi is running off of an old time.

### 4.2 Results

In this paper, six files were encrypted and decrypted to test the throughput of each algorithm. One text file, three JPEG pictures of varying size, and two QuickTime movie files of different size were used for testing. The picture and movie files were sources from NASA imagery and would be typical of images that could be transmitted from a spacecraft. The text file contains a short paragraph. This sort of a text file might be generated after the satellite runs a debug or system check. While it is uncommon for small satellites to transmit movies (due to bandwidth and communication window limitations), as small spacecraft capabilities advance it is not impossible for this to occur in the near future.

The file sizes of the test data are as follows. The text file was 67,655 bytes. The three picture files had sizes of 148,497 bytes (small) 409,306 bytes (medium) and 538,156 bytes (large). The small movie was 5,493,374 bytes, and the large movie was 9,689,032 bytes. Each file was encrypted and decrypted a total of 150 times, in groups of 50 operations of encryption and decryption at a time. In addition, the initial key values used in our novel algorithm testing are the same values used in [13]: x=-0.175, y=0.216, and z=-0.811. Multiple block sizes of the proposed algorithm were tested. Namely, $n = 4$, 8, 16, 32, 64, 128, and 256, where $n$ is one side of a square matrix. Block sizes for the algorithm are respectively 16, 64, 256, 1024, 4096, 16384, and 65536 bytes. Results from these experiments are presented in Tables 1 to 4.

As this testing occurred on mission-realistic hardware (and the Raspberry Pi units are single core computers, meaning that system processes could impair a given test), noise was introduced into the data set. For this reason, both the mean (which is more impacted by the interference, but would be an accurate measure of performing multiple encryptions/decryptions over time) and the median values (which provide a better view of the single file encryption cost) are presented. For sake of consistency, the median values are used as the comparison metric.

## 5 Analysis of Results

As shown in Tables 1 and 3, AES encrypted and decrypted the files faster than both SPECK and the proposed algorithm. The processing times for SPECK and the proposed algorithm are comparable, however SPECK performed better than the proposed algorithm across all scenarios. It is important to note, however, that the Raspberry Pi has built-in hardware support for AES which the Crypto++ Library has been built to take advantage of this feature [8]. Previous testing [24] has shown that (hardware aided) AES-NI is several times faster than AES encryption without the hardware optimization.

Note that times were also compared where SPECK and the novel algorithm were both added to AES encryption. While some nominal overhead is expected, these values demonstrate to how the algorithms would perform if they were combined with AES (mirroring the work done by [21]). While SPECK + AES was faster than the proposed algorithm combined with AES by a reasonable margin, it is possible that, with further optimization, the proposed algorithm combined with AES could produce similar result times. This will serve as a prospective topic for future work.

A comparison of the performance of the algorithms under the various conditions also produces data of some interest. Considering Table 2, for example, it is clear that encryption times for the Test file vary, with the 256 and 1025 byte block size encrypting in the least amount of

Table 1: Mean and media, showing comparison between encryption times of AES, SPECK, and the novel algorithm with a block size 16 bytes. Times shown are in milliseconds.

| | | Text File | Small Picture | Medium Picture | Large Picture | Small Movie | Large Movie |
|---|---|---|---|---|---|---|---|
| AES | Mean | 122.21 | 155.67 | 208.36 | 218.67 | 1896.95 | 3369.14 |
| | Median | 26.44 | 59.07 | 167.11 | 192.11 | 1820.96 | 3307.72 |
| SPECK | Mean | 133.48 | 229.60 | 563.94 | 768.55 | 7565.59 | 13180.98 |
| | Median | 92.53 | 199.89 | 545.28 | 717.17 | 7558.81 | 13131.20 |
| 16 bytes | Mean | 247.08 | 321.14 | 721.23 | 938.66 | 9376.04 | 16380.62 |
| | Median | 116.39 | 249.53 | 677.43 | 890.02 | 9310.50 | 16276.60 |
| AES + SPECK | | 118.96 | 258.96 | 712.40 | 909.27 | 9379.77 | 16438.92 |
| AES + 16 bytes | | 142.83 | 308.59 | 844.54 | 1082.13 | 11131.46 | 19584.32 |

Table 2: Mean and median times of encryption for each file, showing data from the novel algorithm for block sizes of 16, 64, 256, 1024, 4096, 16384, and 65536 bytes. Times shown are in milliseconds.

| | | Text File | Small Picture | Medium Picture | Large Picture | Small Movie | Large Movie |
|---|---|---|---|---|---|---|---|
| 16 bytes | Mean | 247.08 | 321.14 | 721.23 | 938.66 | 9376.04 | 16380.62 |
| | Median | 116.39 | 249.53 | 677.43 | 890.02 | 9310.50 | 16276.60 |
| 64 bytes | Mean | 235.02 | 280.08 | 630.58 | 820.12 | 8162.89 | 14142.78 |
| | Median | 181.70 | 214.81 | 584.57 | 768.05 | 8079.60 | 14058.80 |
| 256 bytes | Mean | 294.86 | 376.05 | 615.42 | 770.55 | 7944.10 | 13910.27 |
| | Median | 94.40 | 202.35 | 549.64 | 721.93 | 7707.64 | 13622.30 |
| 1024 bytes | Mean | 216.49 | 261.31 | 588.56 | 771.28 | 7735.40 | 13503.79 |
| | Median | 96.97 | 206.13 | 556.55 | 730.85 | 7686.17 | 13365.95 |
| 4096 bytes | Mean | 611.32 | 396.03 | 641.75 | 811.43 | 7726.50 | 13618.76 |
| | Median | 586.87 | 205.07 | 543.04 | 714.36 | 7490.01 | 13251.30 |
| 16384 bytes | Mean | 524.18 | 539.65 | 590.94 | 780.36 | 8044.89 | 14309.31 |
| | Median | 551.14 | 343.87 | 555.32 | 730.54 | 7804.75 | 14235.70 |
| 65536 bytes | Mean | 370.34 | 475.52 | 961.50 | 1217.91 | 10073.63 | 14139.14 |
| | Median | 294.81 | 306.69 | 679.57 | 865.36 | 8117.33 | 14030.40 |

Table 3: Comparison between the mean and median decryption time data results between AES, SPECK, and the novel algorithm with a block size of 16 bytes. Times shown are in milliseconds.

| | | Text File | Small Picture | Medium Picture | Large Picture | Small Movie | Large Movie |
|---|---|---|---|---|---|---|---|
| AES | Mean | 103.43 | 141.83 | 183.75 | 196.29 | 1800.80 | 3171.62 |
| | Median | 24.46 | 51.70 | 137.42 | 170.73 | 1775.65 | 3093.66 |
| SPECK | Mean | 122.53 | 225.49 | 573.42 | 748.60 | 7624.45 | 13264.09 |
| | Median | 92.25 | 201.15 | 548.90 | 722.06 | 7608.54 | 13212.20 |
| 16 bytes | Mean | 230.21 | 326.44 | 770.75 | 990.79 | 10051.66 | 17379.17 |
| | Median | 123.84 | 265.01 | 718.77 | 945.63 | 9938.16 | 17243.20 |
| AES + SPECK | | 116.70 | 252.85 | 686.32 | 892.79 | 9384.18 | 16305.86 |
| AES + 16 bytes | | 148.30 | 316.71 | 856.19 | 1116.36 | 11713.80 | 20336.86 |

Table 4: Comparison between the mean and median decryption time data results for the novel algorithm for block sizes of 16, 64, 256, 1024, 4096, 16384, and 65536 bytes. Times shown are in milliseconds.

| | | Text File | Small Picture | Medium Picture | Large Picture | Small Movie | Large Movie |
|---|---|---|---|---|---|---|---|
| 16 bytes | Mean | 230.21 | 161.50 | 770.75 | 990.79 | 10051.66 | 17379.17 |
| | Median | 123.84 | 265.01 | 718.77 | 945.63 | 9938.16 | 17243.20 |
| 64 bytes | Mean | 228.58 | 309.22 | 676.09 | 880.58 | 8695.97 | 15322.08 |
| | Median | 107.81 | 232.37 | 628.41 | 824.79 | 8656.10 | 15104.05 |
| 256 bytes | Mean | 228.81 | 400.91 | 680.78 | 820.42 | 8457.53 | 14827.25 |
| | Median | 101.27 | 218.52 | 591.57 | 777.03 | 8119.46 | 14551.20 |
| 1024 bytes | Mean | 213.35 | 309.07 | 636.41 | 851.08 | 8328.38 | 14533.98 |
| | Median | 105.23 | 222.94 | 600.10 | 788.16 | 8269.76 | 14382.65 |
| 4096 bytes | Mean | 482.16 | 479.97 | 672.05 | 843.53 | 8427.94 | 14958.49 |
| | Median | 105.81 | 222.31 | 588.58 | 776.07 | 8064.88 | 14661.90 |
| 16384 bytes | Mean | 512.90 | 599.42 | 687.79 | 829.91 | 8575.43 | 15253.62 |
| | Median | 586.59 | 463.81 | 600.59 | 788.52 | 8162.46 | 14906.35 |
| 65536 bytes | Mean | 350.06 | 486.52 | 1034.06 | 1315.57 | 10737.79 | 15190.99 |
| | Median | 286.71 | 328.53 | 727.43 | 930.95 | 8681.52 | 15062.95 |

time. For the small picture, the 256, 1025, and 4096 block sizes are all comparable, and all slightly faster than the 16 byte block size. For larger file sizes, the larger block sizes appear to function faster than the smaller block sizes. The medium picture was best encrypted by the 256, 1024, 4096, and 16384 block sizes, with the 64 block size only marginally slower. The same trend occurs when looking at the large picture. The small picture was best encrypted by the 4096 block size, with 256, 1024, and 16384 slightly slower. It is important to note that, as we look at these large file sizes, the 16 byte encryption algorithm is performing the slowest. Finally, the large movie was best encrypted by the 4096 block size, with 256 and 1024 block sizes performing slightly slower.

Reviewing the data in Table 4 shows that the results for the text file and the small picture are similar, with block sizes of 64, 256, 1024, and 4096 all performing best and within very close limits to one another, with 16 block encryption close behind. The medium picture is also similar, with 245, 1024, and 4096 performing optimally and similarly. However, the 16384 block size performs similarly to the other three, and the 64 block size is operating slightly slower. For the large picture, again, block sizes of 256, 1024, 4096, and 16384 perform best. The small movie results are similar to the large picture. Finally, the large movie was best encrypted by block sizes of 256, 1024, and 4096, with 64, 16384 and 65536 falling not unbelievably far behind.

The foregoing demonstrates that, in addition to the selection of an algorithm for use on the spacecraft, it may be necessary to use multiple block sizes. Alternately, system developers might choose to project the types of data that will be sent in order to optimize the block size selected.

# 6 Conclusions and Future Work

The work performed has shown that, in the context of the use of a single algorithm on the Raspberry Pi hardware, AES seems to be a better choice for encrypting and decrypting data on small satellites than the proposed algorithm when hardware optimization is present. Further testing is needed to consider how the proposed algorithm compares to AES in a non-optimized environment. The potential of implementing hardware enhancement for the proposed algorithm could also be considered.

In addition, while the SPECK algorithm implementation performed better than the 16 byte block size version of the proposed algorithm, the results are similar enough that further optimization the proposed algorithm could potentially increase the speed to the point of matching or outperforming SPECK. The potential to hardware optimize both would also bear consideration.

Finally, this paper has demonstrated the importance of block size selection for optimization of the performance of the proposed algorithm. It has provided data that may aid block size selection, based on the size of the data being encrypted and decrypted.

# Acknowledgements

This research was funded by the U.S. National Science Foundation (NSF Award # 1359224) with support from the U.S. Department of Defense. This paper builds on prior work presented in [17].

# References

[1] L. R. Abramowitz, "US air force SMC/XR sense nanoSat program," in *Aiaa Space 2011 Conference*

*& Exposition*, 2011. (https://doi.org/10.2514/6.2011-7332)

[2] Y. Asimi, A. Amghar, A. Asimi and Y. Sadqi, "New random generator of a safe cryptographic salt per session," *International Journal of Network Security*, pp. 445-453, 2016.

[3] R. Beaulieu, D. Shors, J. Smith, S. Treatman-Clark, B. Weeks and L. Wingers, "The SIMON and SPECK families of lightweight block ciphers," *IACR Cryptology ePrint Archive*, pp. 404, 2013.

[4] O. Challa, G. Bhat and J. Mcnair, "CubeSec and GndSec: A lightweight security solution for CubeSat communications," in *Small Satellite Conference*, pp. 8, 2012.

[5] G. Chen, Y. Mao and C. K. Chui, "A symmetric image encryption scheme based on 3D chaotic cat maps," *Chaos, Solitons & Fractals*, pp. 749-761, 2004.

[6] J. H. Curry, "A generalized lorenz system," *Communications in Mathematical Physics*, vol. 60, no. 3, pp. 193-204, 1978.

[7] P. Cvitanovi, G. H. Gunaratne and I. Procaccia, "Topological and metric properties of hnon-type strange attractors," *Physical Review A*, pp. 1503, 1988.

[8] W. Dai, *Crypto++ Library 5.6.2*, May 25, 2018. (http://www.cryptopp.com/)

[9] N. El-Fishawy and O. M. A. Zaid, "Quality of encryption measurement of bitmap images with RC6, MRC6 and rijndael block cipher algorithms," *International Journal of Network Security*, pp. 241-251, 2007.

[10] D. S. A. Elminaam, H. M. A. Kader and M. M. Hadhound, "Evaluating the performance of symmetric encryption algorithms," *International Journal of Network Security*, pp. 213-219, 2010.

[11] European Space Ahgency, *CubeSats - Fly Your Satellite!* May 25, 2018. (https://www.esa.int/Education/CubeSats_-_Fly_Your_Satellite)

[12] P. Fortescue, G. Swinerd and J. Stark, "Spacecraft systems engineering, 4th edition," *Wiley*, pp. 724, 2011. ISBN: 978-0-470-75012-4.

[13] X. Huang, G. Ye and K. Wong, "Chaotic image encryption algorithm based on circulant operation," *Presented at Abstract and Applied Analysis*, vol. 2013, pp. 8, 2013.

[14] R. J. Hughes, W. T. Buttler, P. G. Kwiat, S. K. Lamoreuax, G. L. Morgan, J. E. Nordholt and C. G. Peterson, "Quantum cryptography for secure satellite communications," in *Proceedings of the IEEE Aerospace Conference*, 2000. (DOI: 10.1109/AERO.2000.879387)

[15] G. Hunyadi, J. Ganley, A. Peffer and M. Kumashiro, "The university nanosat program: An adaptable, responsive and realistic capability demonstration vehicle," in *Proceedings of the IEEE Aerospace Conference*, vol. 5, 2004.

[16] S. P. Indrakanti and P. S. Avadhani, "Permutation based image encryption technique," *International Journal of Computer Applications*, pp. 45-47, 2011.

[17] S. Jackson, S. Kerlin and J. Straub, "Implementing and testing a novel chaotic cryptosystem for use in small satellites," in *Proceedings of the ACM Conference on Computer and Communications Security*, pp. 1638-1640, 2015.

[18] C. H. Ling, C. C. Lee, C. C. Yanh and M. S. Hwang, "A secure and efficient one-time password authentication scheme for WSN," *International Journal of Network Security*, 2017. (DOI: 10.6633/IJNS.201703.19(2).02)

[19] D. S. A. Minaam, H. M. Abdul-Kader and M. M. Hadhound, "Evaluating the effects of symmetric cryptography algorithms on power consumption for different data types," *International Journal of Network Security*, pp. 78-87, 2010.

[20] J. Mueller, R. Hofer and J. Ziemer, "Survey of propulsion technologies applicable to cubeSats," in *NASA Techical Reports Server*, 2010. hdl:2014/41627.

[21] F. T. B. Muhaya, "Chaotic and AES cryptosystem for satellite imagery," *Telecommunication Systems*, pp. 573-581, 2013.

[22] G. Peterson, "Arnold cat map," *Math45-Linear Algebra*, 1997. (http://pages.physics.cornell.edu/~sethna/teaching/562_S03/HW/pset02_dir/catmap.pdf)

[23] J. Rarity, P. Tapster, P. Gorman and P. Knight, "Ground to satellite secure key exchange using quantum cryptography," *New Journal of Physics*, pp. 82, 2002.

[24] P. Schmid and A. Roos, "AES-NI performance analyzed; Limited To 32nm core i5 CPUs," *Conclusion*, 2010. (http://www.tomshardware.com/reviews/clarkdale-aes-ni-encryption,2538.html.)

[25] G. Skrobot and R. Coelho, "ELaNaeducational launch of nanosatellite: Providing routine RideShare opportunities," in *Presented at Proceeding SmallSat Conference*, 2012. (https://ntrs.nasa.gov/search.jsp?R=20120015542)

[26] D. Stinson, "Cryptography: Theory and Practice," *Cryptography & Coding Theory*, pp. 616, 2006.

[27] J. Straub and J. Vacek, "Escaping earths orbit but not earthly regulations: A discussion of the implications of ITAR, EAR, FCC regulations and title VII on interplanetary CubeSats and CubeSat programs," *Interplanetary CubeSat Workshop*, 2013. (https://icubesat.files.wordpress.com/2013/06/icubesat-org_2013-b-3-3-regulations_straub_201305291815.pdf)

[28] J. Straub, "CyberSecurity for aerospace autonomous systems," in *Presented at SPIE Defense Security*, 2015. (DOI:10.1117/12.2177959)

[29] D. Weeks, A. B. Marley and J. London III, "SMDC-ONE: An army nanosatellite technol-

ogy demonstration," in *Small Satellite Conference*, 2009. (https://digitalcommons.usu.edu/smallsat/2009/all2009/62/)

[30] J. R. Wertz, D. F. Everett and J. J. Puschell, "Space mission engineering: The new smad," *Amazon*, 2011. (https://www.amazon.co.uk/Space-Mission-Engineering-New-Smad/dp/1881883159)

## Biography

**Samuel Jackson** is an undergraduate student studying Computer Science at the Oklahoma State University in Stillwater, OK. In the summer of 2015, he participated in the small satellite research experience for undergraduates program at the University of North Dakota.

**Jeremy Straub** is an assistant professor in the Department of Computer Science at the North Dakota State University. He holds a PhD in Scientific Computing, an MS and an MBA and two BS degrees. He has published over 40 journal articles and over 120 full conference papers, in addition to making numerous other conference presentations. His research spans the gauntlet between technology, commercialization and technology policy. He serves as the associate director of the NDSU Institute for Cyber Security Education and Research.

**Scott Kerlin** graduated with an MS in Computer Science in 2010. He is currently Undergraduate Director for the University of North Dakota Computer Science Department. He specializes in programming, Computer Science education, cyber security and information assurance.