Fault-Tolerant Encryption for Space Applications

ROOHI BANU, Student Member, IEEE
TANYA VLADIMIROVA, Member, IEEE
Surrey Space Centre

This paper is concerned with the use of commercial security algorithms like the Advanced Encryption Standard (AES) in Earth observation small satellites. The demand to protect the sensitive and valuable data transmitted from satellites to ground has increased and hence the need to use encryption on board. AES, which is a very popular choice in terrestrial communications, is slowly emerging as the preferred option in the aerospace industry including satellites.

This paper first addresses the encryption of satellite imaging data using five AES modes—ECB, CBC, CFB, OFB and CTR. A detailed analysis of the effect of single even upsets (SEUs) on imaging data during on-board encryption using different modes of AES is carried out. The impact of faults in the data occurring during transmission to ground due to noisy channels is also discussed and compared for all the five modes of AES.

In order to avoid data corruption due to SEUs, a novel fault-tolerant model of AES is presented, which is based on the Hamming error correction code. A field programmable gate array (FPGA) implementation of the proposed model is carried out and measurements of the power and throughput overhead are presented.

Manuscript received September 13, 2006; revised June 18 and July 10, 2007; released for publication April 2, 2008.

IEEE Log No. T-AES/45/1/932018.

Refereeing of this contribution was handled by M. Rice.

The research presented here is sponsored by the University of Surrey and the UK ORS scheme.

Authors' current addresses: R. Banu, 5 Fairview Close, Woking, GU22 7NT, Surrey, UK, E-mail: (roohi.banu@hotmail.co.uk); T. Vladimirova, Surrey Space Centre, Dept. of Electronic Engineering, University of Surrey, Guildford, Surrey, GU2 7XH, UK.

0018-9251/09/$25.00 © 2009 IEEE

I. INTRODUCTION

Earth observation (EO) satellites take images of the Earth with smart and sophisticated imaging sensors. Multispectral images of the Earth captured by optical on-board cameras can be used in monitoring of the environment and disasters, vegetation control, map marking, urban planning, etc. The latest trend now is towards small EO satellites, as they require smaller budgets to build and launch and also involve less maintenance costs [1]. A typical EO small satellite weighs approximately 100 kg and the orbit average power generated by solar panels is 30 to 60 W [2]. The imaging payload units of such satellites comprise imagers, mass memory, and high-rate data transceivers and consume up to 70% of the average orbit power. For example, the recently launched small EO satellite TopSat [3] weighs 110 kg with average orbit power of 55 W. The capacity of the memory data recorders of TopSat is 1 Gbytes and the downlink transmission rate through X-band is 25 Mbit/s.

Recent unauthorized intrusions into satellite data have raised the importance of using security services on board [4]. Encryption, by far the most widely adopted security service in terrestrial networks, is used to protect data from unauthorized users. Although there are many encryption products and algorithms, the use of these products and algorithms on-board satellites has been overlooked until recently. But now satellite manufacturers are realizing the importance of on-board encryption to protect valuable data, especially after cases, which have proved that intrusion into satellite data is not an impossible task [5, 6].

At present, more and more EO satellites are equipped with on-board encryption to protect the data transmitted to the ground station. However due to confidentiality and security reasons the coverage of this topic in the open literature is very limited. Examples of EO satellites that include on-board data encryption are the following: Space Technology Research Vehicle (STRV) -1d, Korea Multipurpose Satellite-II (KOMPSAT-2), MeteoSat Second Generation (MSG) Spacecraft, MetOp-A Polar Orbiting Spacecraft, Canadian satellite RADSAT-2 [7, 8, 9, 10]. The STRV, MetOp and RADSAT employ the Data Encryption Standard (DES) whereas KOMSAT uses the International Data Encryption Algorithm (IDEA). It can be seen from these examples that the encryption algorithms used in present satellite missions are proprietary or outdated algorithms like DES, rather than algorithms based on the latest encryption standards.

The Rijndael algorithm approved as the Advanced Encryption Standard (AES) by the US National Institute of Standards and Technology (NIST) is a block cipher, which encrypts one block of data at a time. To encrypt multiple blocks, modes of operation have been defined by NIST. AES is being adopted by many organizations across the world. Because of its simplicity, flexibility, easiness of implementation, and high throughput AES is used in many different applications ranging from smart cards to big servers, however at the time of the writing of this paper no use of AES on board satellites has been reported. In fact, hardware implementations of AES are well suited to resource-constrained embedded applications like satellites [11]. There are various hardware implementations of the AES algorithm on platforms like application specific integrated circuits (ASICs) and field programmable gate arrays (FPGAs) that achieve a significant throughput ranging from a few Mbit/s to Gbit/s [5, 12]. Thus the requirements of small EO satellites for high-rate data transmission are met by existing AES implementations. However, in addition to high throughput, immunity of the encryption process against faults is very important in satellites.

Satellites operate in a harsh radiation environment and consequently any electronic system used on board, including the encryption processor, is susceptible to radiation-induced faults. Most of the faults that occur in satellite on-board electronic devices are radiation-induced bit flips called single event upsets (SEUs) [13, 14]. If faulty data is transmitted to the ground station, the user's request for data retransmission has to wait until the next satellite revisit period, with revisit time varying from a couple of hours to weeks. In order to prevent faulty data transmissions, there is a need for an error-free encryption scheme on board. Satellite data can further get corrupted during transmission to ground due to noise in the transmission channel.

The impact of radiation on semiconductor devices on board depends on orbit altitude, orientation, and time. In low Earth orbit (LEO) SEU rates of the order of 10^{-6} bit^{-1} day^{-1} can be expected [15]. Static random access memory (SRAM) based FPGAs are particularly susceptible to SEUs [16]. For example, the estimated SEU rate of the XILINX Virtex FPGA device XQVR1000 placed in a satellite at an altitude of 1000 km and 60° inclination ranges from 0.4/h to 12.6/h depending on the solar flare activities [17].

Reliability is the most important issue in avionics design. SEUs must be detected and corrected on board before sending the data to ground. The triple modular redundancy (TMR) technique is one of the most widely used redundancy-based SEU mitigation techniques in satellites. A TMR design consists of three identical modules, which are connected by a majority voting circuit to determine the output [14]. However, with the TMR technique the area and power overheads triplicate in comparison with the original module.

This paper addresses reliability issues of the AES algorithm. A detailed analysis of the impact of faults during on-board encryption and during transmission for the five most commonly used AES modes is presented [18]. A novel fault-tolerant model of AES based on built-in error detection and correction (EDAC) is proposed. The implementation of the proposed model is carried out using an FPGA and the overhead caused by the incorporated EDAC function is quantified. Multispectral satellite images from Surrey Satellite Technology Ltd. (SSTL) [2] and the Internet [19] are employed to demonstrate the on-board encryption and the SEU propagation in the modes.

The paper is organized as follows. Section II gives details about the AES algorithm and discusses suitability of AES modes of operation for encryption of satellite images. In Section III an analysis of fault propagation in five popular AES modes during on-board encryption and during transmission to ground is given. Section IV introduces the proposed fault-tolerant AES model and presents a purpose-built software simulator. Section V describes and compares FPGA implementations of AES realizing different algorithmic options and incorporating the EDAC function. Section VI concludes the paper.

II. ADVANCED ENCRYPTION STANDARD—ALGORITHM AND MODES OF OPERATION

The AES is a symmetric key algorithm, in which both the sender and the receiver use a single key for encryption and decryption. AES defines the data block length to 128 bits, and the key lengths to 128, 192, or 256 bits [11]. It is an iterative algorithm and each iteration is called a round. The total number of rounds, N_r, is 10, 12, or 14 when the key length is 128, 192, or 256 bits, respectively. Each round in AES, except the final round, consists of four transformations: SubBytes, ShiftRows, MixColumns, and AddRoundKey. The final round does not have the MixColumns transformation as shown in Fig. 1. The decryption flow is simply the reverse of the encryption flow and each operation is the inverse of the corresponding one in the encryption process.

The round transformation of AES and its steps operate on some intermediate results, called state. The state can be visualized as a rectangular matrix with four rows. The number of columns in the state is denoted by Nb and is equal to the block length in bits divided by 32. For a 128 bit data block (16 bytes) the value of Nb is 4, hence the state is treated as a 4×4 matrix and each element in the matrix represents a byte. For the sake of simplicity, in the rest of the paper, both the data block and the key lengths are considered as 128 bit long. However all the discussions and the results hold true for 192 bit and 256 bit keys as well.

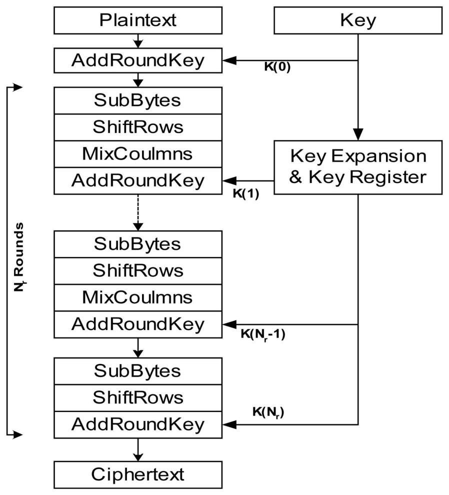


Fig. 1. Block diagram of AES algorithm.

A. Transformations of AES Algorithm

All the four transformations of the AES round work on the principles of finite fields. The number of the elements in a finite field is called the order of the field. A finite field of order p^n is generally denoted as GF (p^n), where GF stands for Galois field, p is the characteristic of the finite field, and n is the number of bits used to represent the field elements.

The basic operations of AES are defined over the elements of the field GF (2^8). The specification of AES has adopted polynomial representation of the byte elements of GF (2^8) with coefficients over the field GF (2). A polynomial representation of byte $a(x)$ with coefficients over the field GF (2) is represented as follows:

$$\begin{aligned} a(x) &= \sum_{i=0}^{7} a_i x^i \\ &= a_7x^7 + a_6x^6 + a_5x^5 + a_4x^4 + a_3x^3 + a_2x^2a_1x + a_0 \end{aligned} \tag{1}$$

where $a_i = \{0,1\}$.

The binary representation of the polynomial given by (1) is as follows:

$$a(x) \mapsto a_7a_6a_5a_4a_3a_2a_1a_0. \tag{2}$$

For example, a byte element represented in binary form as 1100 0001 is written in polynomial form as $x^7 + x^6 + 1$.

The operation of addition using byte elements is defined as addition of the corresponding polynomials. In case of polynomials over GF (2) addition is just an exclusive OR (XOR) operation. For byte multiplication the following irreducible polynomial is used in AES:

$$m(x) = x^8 + x^4 + x^3 + x + 1. \tag{3}$$

Multiplication in GF (2^8) is denoted as $\otimes$. The multiplication of two polynomials $a(x)$ and $b(x)$ is defined as the algebraic product of the polynomials modulo the irreducible polynomial $m(x)$ as below:

$$c(x) = a(x) \otimes b(x) = a(x)b(x) \bmod m(x). \tag{4}$$

The SubBytes transformation is a nonlinear byte substitution, operating on each byte of the state matrix independently. Each byte of the state is first inverted in GF (2^8) and then processed through an affine transformation. The SubBytes transformation can be calculated on the fly for each state byte. Alternatively, for the sake of computational simplicity, SubBytes can be computed in advance and stored in a look-up table (LUT) of $2^8 = 256$ elements called S-Box. ShiftRows cyclically left shifts the last three rows of the state by 1, 2, and 3 bytes, respectively. MixColumns transforms every column in the state by multiplying it with a predefined polynomial {2 3 1 1}. Finally, the AddRoundKey transformation adds the expanded round key to the state by an XOR operation [11]. The input key is expanded internally to derive round keys for each of the rounds by following a key expansion algorithm. The transformations involved in the key expansion use operations like substitution, shift, and XOR, which are similar to those involved in the AES round transformations and therefore they are not discussed in detail here.

The state of round i using the above four transformations can be expressed mathematically as follows:

$$\begin{aligned} \begin{bmatrix} d_{0,j} \\ d_{1,j} \\ d_{2,j} \\ d_{3,j} \end{bmatrix} &= \begin{bmatrix} 02 \\ 01 \\ 01 \\ 03 \end{bmatrix} \mathbf{S}_{\text{RD}}[a_{0,j+C_0}] \oplus \begin{bmatrix} 03 \\ 02 \\ 01 \\ 01 \end{bmatrix} \mathbf{S}_{\text{RD}}[a_{1,j+C_1}] \\ &\oplus \begin{bmatrix} 01 \\ 03 \\ 02 \\ 01 \end{bmatrix} \mathbf{S}_{\text{RD}}[a_{2,j+C_2}] \oplus \begin{bmatrix} 01 \\ 01 \\ 03 \\ 02 \end{bmatrix} \mathbf{S}_{\text{RD}}[a_{3,j+C_3}] \oplus K_i \\ &0 \le j < 4 \quad \text{and} \quad 0 < i \le 10 \end{aligned} \tag{5}$$

where a represents the input at the beginning of the round, j is the column number in the state matrix, $\mathbf{S}_{\text{RD}}$ is the S-Box LUT, K_i is the round key of the ith round, C_0, C_1, C_2, C_3 are 0, 1, 2, 3, respectively, to account for the amount of shifting in the ShiftRows operations and the symbol $\oplus$ denotes a bit-wise XOR operation.

B. AES Modes of Operation

The AES encryption algorithm accepts one data block and the key and produces the encrypted data block. The input and output data blocks are of identical size. The decryption algorithm accepts one encrypted data block and the key to produce the encrypted data block. Several modes of operation have been defined to apply the AES block cipher to encryption of more than one 128 bit block of data [11, 18].

The most commonly used modes with AES are: electronic code book (ECB) mode, cipher block chaining (CBC) mode, output feedback (OFB) mode, cipher feedback (CFB) mode, and counter (CTR) mode. ECB and CTR are known as nonfeedback modes whereas CBC, CFB, and OFB are known as feedback modes. In addition, ECB and CBC are referred to as block cipher modes as they require the entire data block before the start of the encryption, and OFB, CFB, and CTR are referred to as stream cipher modes as they operate in a stream-like fashion [20].

C. AES Encryption of Satellite Multispectral Images

In this section five AES modes are investigated with respect to their suitability for encryption of multispectral satellite images.

The AES encryption and decryption of the satellite multispectral images are performed with a software program written in the Java programming language. The AES software implementation consists of core modules and feedback modules. The core modules implement the ShiftRows, SubBytes, MixColumns transformations and the corresponding inverse modules for decryption. The feedback modules realise the encryption and decryption routines for the ECB, CBC, CFB, OFB, and CTR modes. The Sun's Java application programmer interface (API) for JPEG images [21] is used for image encoding and decoding during the encryption and decryption process. The software is also designed in such a way that single bit faults can be randomly injected at any round, transformation, byte, and bit level to simulate SEUs [5].

The multispectral satellite image in Fig. 2(a) [19] is employed as a test image to carry out encryption using the modes of AES. It is found that images encrypted with the ECB mode reveal patterns in the input data as shown in Fig. 2(b), which makes the ECB mode insecure. This is because in the ECB mode the same plain data input results in the same cipher data output. Fig. 2(c) shows the encrypted image using the CBC, OFB, CFB, and CTR modes, where no data patterns are revealed.

Preprocessing of the keystream is one of the main advantages of the OFB and CTR modes. This OFB/CTR feature could enable high-speed and near real-time encryption of data with less processing time and computing resources in satellites. The plain data could just be XOR-ed with the precomputed keystream to generate the encrypted data when they are to be transmitted to ground. However, the storage of the keystream requires a huge amount of memory, as the keystream length is equal to the data stream length, which makes preprocessing in OFB/CTR not so attractive for on-board use.

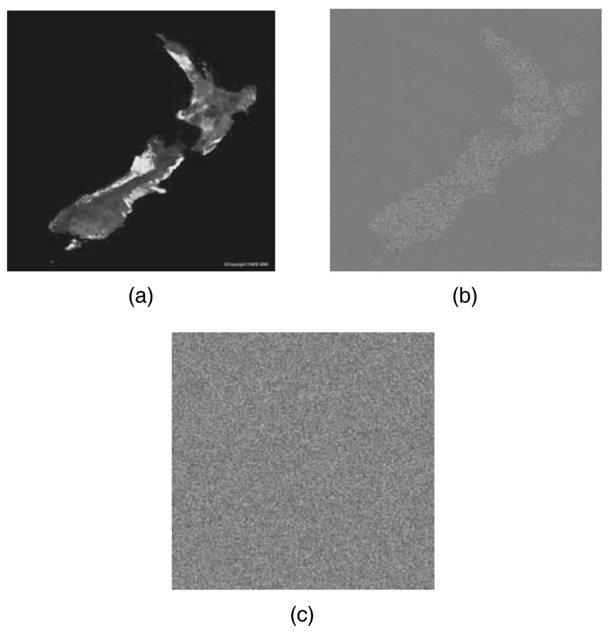


Fig. 2. (a) Plain image. (b) Encrypted image using ECB. (c) Encrypted image with CBC, OFB, CFB, and CTR.

TABLE I
Characteristics of AES Modes

Feature	Mode				
	ECB	CBC	OFB	CFB	CTR
Pattern Observable?	Yes	No	No	No	No
Block/Stream Cipher?	Block	Block	Stream	Stream	Stream
Feedback Mode?	No	Yes	Yes	Yes	No
Preprocessing possible?	No	No	Yes	No	Yes

The remaining AES modes, CBC and CFB, are also viable options for on-board use. These two modes have identical characteristics except that CBC is a block cipher mode whereas CFB is a stream cipher mode. Table I compares and summarizes the characteristics of all the AES modes, as discussed above.

III. FAULT PROPAGATION ANALYSIS

Due to fault propagation even a single bit error during encryption of one block of AES can result in corruption of 50% of the bits in the final encrypted data on average [22]. In addition, when using the AES feedback modes faults occurring in one block can propagate to other blocks because of the feedback. In this section propagation to subsequent blocks of single bit faults occurring both during encryption and during transmission is investigated. Five AES modes are evaluated for use on board satellites taking into account the impact of fault propagation.

A. Fault Propagation in AES Modes

1) *Electronic Code Book Mode*: If an SEU occurs during AES encryption with ECB, then the corresponding cipher data block and hence the subsequent entire plain data block will be garbled when decrypted. If a single bit of the cipher data block is corrupted due to noise in the transmission channel, then the entire corresponding plaintext block will also be corrupted. These faults are not propagated to other blocks, as there is no feedback. The faults are just confined to the concerned block only.

2) *Cipher Block Chaining Mode*: The CBC mode, illustrated in Fig. 3, is the mode in which the plain data block is XOR-ed with the cipher data of the previous block before it is encrypted. The first block is XOR-ed with an initial vector (IV), which is a random number. In Fig. 3 P1,P2...Pn represent the plain data, C1,C2...Cn represent the cipher data, and K is the key used in both encryption and decryption. The plain data blocks, the cipher data blocks, and the key are of 128 bit length each. The "E" and "D" blocks in Fig. 3 denote an encryption and decryption function using the AES algorithm, respectively.

The effect of an SEU during encryption in the CBC mode is illustrated in Fig. 3, where the SEU occurrence is marked by the star symbol * and the corrupted data blocks are represented by black boxes. If an SEU occurs while encrypting the plain block P1, the cipher block C1 will be corrupted and hence the decrypted block P1 will also be corrupted. However, this corrupted data is not propagated to the subsequent blocks despite the feedback. The reason for this is that the corrupted cipher block C1 is XOR-ed twice (with the plain block P2 before encryption and with the cipher block C2 after decryption) as shown in Fig. 3. Performing the XOR operation two times with this corrupted cipher block C1 neutralizes the fault and prevents propagation of faults to subsequent blocks as shown below:

$$\mathrm{P} \oplus X \oplus X = \mathrm{P} \tag{6}$$

where X is the faulty data and P is the plain data.

In contrast, a fault occurring in an encrypted block during transmission propagates to the next block, as shown in Fig. 4, where the transmission fault is shown by the star symbol $*$ during the transmission of the cipher block C1. The decrypted block P1 is completely garbled and the subsequent decrypted block P2 will have bit errors at the same positions as the original erroneous block C1 [20]. The decrypted blocks following the second block will not be affected by the fault. Hence, the CBC mode is self-recovering or self-synchronizing.

3) *Output Feedback Mode*: In the OFB mode the output of the encryption is fed back into the input to generate a keystream, which is then XOR-ed with the plain data to generate the cipher data, as illustrated in Fig. 5. If an SEU occurs during encryption in the OFB mode then all the subsequent blocks will be corrupted starting from the point where the fault has occurred as shown in Fig. 5. This is because the keystream required for encryption and decryption is independent of the plain and cipher data and hence the feedback propagates the faults from one block to another until the end of the encryption process.

This is demonstrated by introducing an SEU during the encryption of a plain multispectral satellite image. The satellite image in Fig. 6(a) is an SSTL multispectral image of North Sumatra taken on 4th January 2005 in the aftermath of the Tsunami disaster [2]. The image has 500×500 pixels and each pixel is

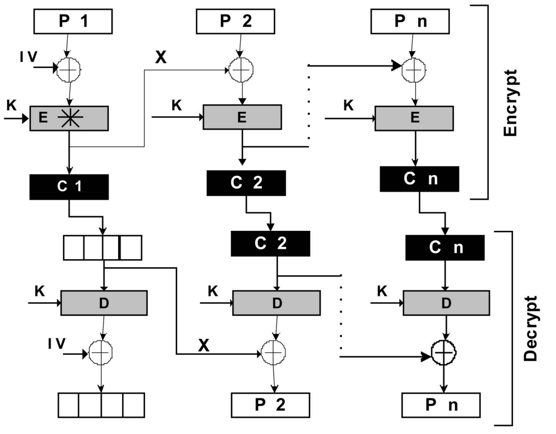


Fig. 3. Fault propagation during encryption in CBC mode.

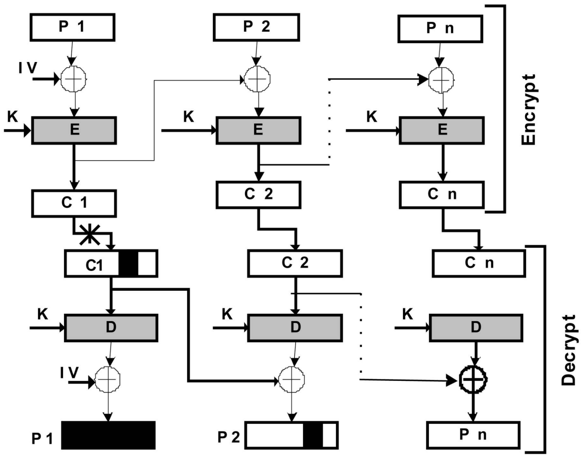


Fig. 4. Transmission fault propagation in CBC mode.

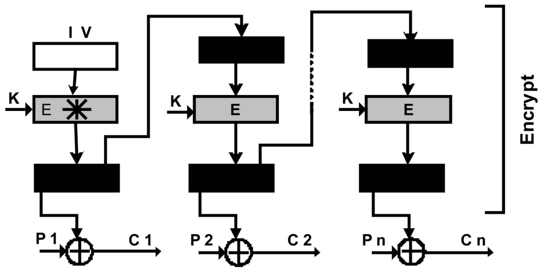


Fig. 5. SEU propagation during encryption in OFB mode.

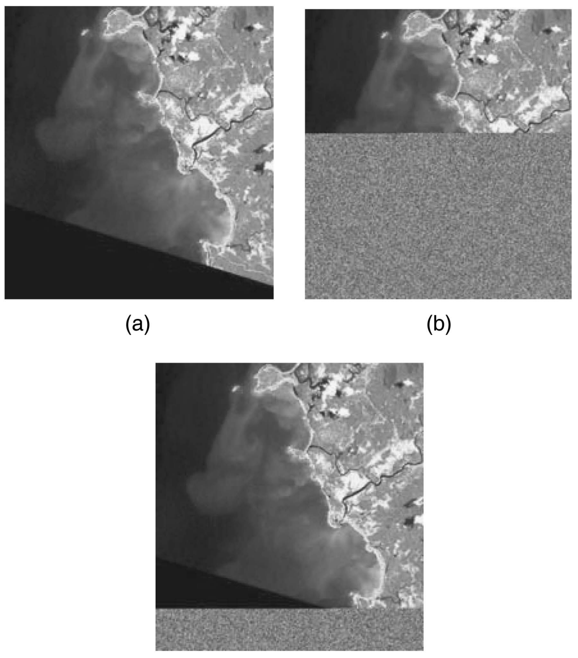


Fig. 6. (a) Plain data multispectral satellite image (courtesy of SSTL). (b) Decrypted OFB mode image with SEU occurring at 20,000th block. (c) Decrypted OFB mode image with SEU occurring at 40000th block.

of 24 bits, representing 3 spectral bands with 8 bits per band. Thus, the number of 128 bit blocks for this image is 46875. Fig. 6(b) shows the fault propagation for a single bit error, which was introduced during the encryption of the 20,000th block at the SubBytes transformation of the 4th byte in the third round. Fig. 6(c) shows the propagation of a single bit error that was introduced during the encryption of the 40,000th block at the MixColumns transformation of the 7th byte in the 6th round. In contrast, if a bit is corrupted during transmission, only a single bit in the plain data is affected and the error does not propagate to other parts of the message again for the same reason that the keystream does not depend on the plain or cipher data. So the transmission fault is not propagated. This property is very useful to applications such as satellites where the transmission channels are very noisy. Hence the OFB mode has an advantage over the CBC and CFB modes in that any bit errors that might occur inside cipher data are not propagated to affect the decryption of subsequent blocks.

4) *Cipher FeedBack Mode*: Due to an SEU during encryption in CFB mode, the corresponding plain data block will be garbled and the faults will not propagated to subsequent blocks. This is again because of the XOR property as described by (6). The keystream used during encryption and decryption depends on the cipher data of the previous block as in CBC mode. So performing XOR two times with the corrupted data neutralises the fault and prevents propagation of faults to subsequent blocks.

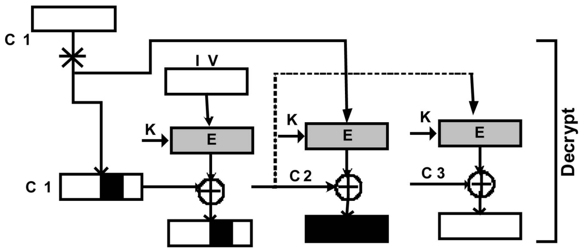


Fig. 7. Propagation of transmission fault in CFB mode.

However, in contrast to CBC, in the CFB mode a transmission fault in an encrypted data block propagates to the next block, which is corrupted completely. This is because during decryption first the XOR operation is carried out followed by encryption as shown in Fig. 7. Also the blocks following the second block will not be affected by the error. Therefore, CFB is also known as self-recovering (self-synchronising).

5) *Counter Mode*: In the CTR mode, similar to the ECB mode, an SEU fault or a transmission fault propagates to only one block as there is no feedback to propagate the faults. An SEU fault during encryption corrupts one complete data block whereas a transmission fault corrupts only the corresponding single bit in the block.

B. Discussion

SEU-inflicted single bit errors can propagate from one block to multiple blocks depending on the mode of operation. It has been observed that in case of the ECB, CBC, CFB, and CTR modes an SEU corrupts only one block of data whereas in case of the OFB mode it can propagate to the whole data block starting from the point where the SEU has occurred. However ECB is not suitable for satellite imaging applications, even though it propagates faults to just one block, as it reveals patterns in the encrypted data, as shown in Section IIC.

Faults occurring during transmission can also propagate from one block to multiple blocks depending on the mode of operation. It has been observed that in case of the ECB mode the faults propagate to one block, whereas in the CBC and CFB modes the faults can propagate to two blocks. In contrast, in the OFB and CTR modes, only a single bit in the plain data is affected and the error does not propagate to other parts of the image. Based on this analysis, we conclude that the OFB and CTR modes are more favourable for noisy channels, because unlike other modes, single bit transmission errors in the cipher data are not expanded in the received plain data. The OFB mode is also recommended in [23] as a more suitable option for satellite communications compared with other modes of DES, however, it is very sensitive to SEUs as demonstrated in Section IIC

TABLE II
Amount of Data Corrupted due to Single Bit Errors During Encryption and Transmission

Source of Error	ECB	CBC	OFB	CFB	CTR
Data corrupted due to SEU during on-board encryption	One block	One block	All data after the point of the fault occurrence	One block	One block
Data corrupted due to faults during transmission	One block	Two blocks	No fault propagation	Two blocks	No fault propagation

above. Hence the OFB mode could be used on board only if an error-free AES encryption scheme as the one proposed in Section IV is employed.

Table II summarizes the amount of data corrupted due to single bit faults during encryption and transmission depending on the AES mode used. It can be seen from Table II that the occurrence of single bit errors during on-board encryption in all AES modes results in fault propagation. This makes a very strong case for development of a self-repairing AES scheme, as it will prevent faulty data transmissions in satellites.

IV. FAULT-TOLERANT MODEL OF THE AES ALGORITHM

This section presents a novel fault-tolerant model for the AES algorithm, which is immune to radiation-induced SEUs occurring during encryption and can be used in hardware implementations on board small OE satellites [24]. The model is based on a self-repairing EDAC scheme, which is built in the AES algorithmic flow and utilizes the Hamming error correcting code [25].

The proposed Hamming code based fault-tolerant model of AES can be adapted to all the five modes of AES to correct SEUs on board. Even though the calculation of the Hamming code is carried out within the AES it does not alter any of the transformations of the algorithm and does not affect in any way the operation of AES. Also as the Hamming parity data are not sent to ground, they are not available to leak any information about the AES algorithm. Therefore the authors believe that the fault-tolerant AES model does not require a new cryptanalysis.

The EDAC function is internal to the AES algorithm in the proposed scheme. It is possible to implement an EDAC function that is external to the AES algorithm using erasure codes [26]. A disadvantage of this approach compared with the proposed method is that erasure codes cannot be adopted for the OFB mode of AES since a fault occurring in one block during encryption will propagate to all the subsequent blocks, as discussed in Section III above. Hence, the erasure code would not be able to recover the data, as the number of corrupted blocks would usually be very high. Another disadvantage is that the implementation of the erasure codes based EDAC will require an additional encoding stage to encode the plain data blocks to codeword symbols which will inevitably add an overhead to on-board resources and processing. Also the impact of SEUs on the encoder needs to be further taken care of.

A. Model Description

The proposed fault-tolerant model is based on the singe error correcting Hamming code $(12,8)$, the simplest of the available error correcting codes. The Hamming code $(12,8)$ detects and corrects a single bit fault in a byte and it is a good choice for satellite applications, as most frequently occurring faults in on-board electronics are bit flips induced by radiation. However, the AES correction model can be extended to correct multiple bit faults by using other error correcting codes such as the modified Hamming code $(16,8)$, the Read-Solomon codes, etc. [25].

The EDAC capability is based on predicting the Hamming code bits (also referred to as parity check bits) at the end of each transformation from the precalculated Hamming code. The procedure to calculate the parity check bits is discussed below.

1) *Calculation of the Hamming Code*: The parity check bits of each byte of the S-Box LUTs are precalculated. These Hamming code bits can be formally expressed as below:

$$\begin{aligned} h(\mathbf{S}_{\mathrm{RD}}[a]) &\rightarrow \mathbf{h}_{\mathrm{RD}}[a] \\ h((\mathbf{S}_{\mathrm{RD}}[a]\otimes\{02\}) &\rightarrow \mathbf{h}_{2\mathrm{RD}}[a] \\ h((\mathbf{S}_{\mathrm{RD}}[a]\otimes\{03\}) &\rightarrow \mathbf{h}_{3\mathrm{RD}}[a] \end{aligned} \tag{7}$$

where a is the state byte and h represents the calculation of the Hamming code.

As can be seen from (7), $\mathbf{h}_{\mathrm{RD}}$ is given by the parity check bits of the S-Box LUT $\mathbf{S}_{\mathrm{RD}}$, $\mathbf{h}_{2\mathrm{RD}}$ is given by the parity check bits of $(\mathbf{S}_{\mathrm{RD}}\otimes\{02\})$, and $\mathbf{h}_{3\mathrm{RD}}$ is given by the parity check bits of $(\mathbf{S}_{\mathrm{RD}}\otimes\{03\})$.

The procedure to derive the $\mathbf{h}_{\mathrm{RD}}$ parity bits is described below by taking one state byte a, represented by bits $(b_7,b_6,b_5,b_4,b_3,b_2,b_1,b_0)$ as an example. The Hamming code of the state byte a is a four-bit parity code, represented by bits (p_3,p_2,p_1,p_0), which are derived as follows:

$$\begin{aligned} p_3 &\rightarrow \text{is parity of bit group } b_7,b_6,b_4,b_3,b_1 \\ p_2 &\rightarrow \text{is parity of bit group } b_7,b_5,b_4,b_2,b_1 \\ p_1 &\rightarrow \text{is parity of bit group } b_6,b_5,b_4,b_0 \\ p_0 &\rightarrow \text{is parity of bit group } b_3,b_2,b_1,b_0. \end{aligned} \tag{8}$$

The Hamming bits of all the bytes of table $\mathbf{S}_{\mathrm{RD}}$ are precalculated and stored in the form of a memory table referred to as the $\mathbf{h}_{\mathrm{RD}}$ table.

The Hamming code $\mathbf{h}_{2\mathrm{RD}}$ is given by the parity check bits of ($\mathbf{S}_{\mathrm{RD}} \otimes \{02\}$). The Galois field multiplication of a state byte a with $\{02\}$ is defined as follows [11, 22]

$$\begin{aligned}\{02\} \otimes a &= \{02\} \cdot a(x) \bmod m(x)\\ &= x \cdot a(x) \bmod m(x)\\ &= \left(x \sum_{i=0}^{n-1} a_i x^i\right) \bmod m(x)\\ &= a_{n-1} \sum_{i=0}^{n-1} m_i x^i + \sum_{i=0}^{n-2} a_i x^{i+1}\\ &= a_{n-1} m_0 + \sum_{i=0}^{n-2} (a_{n-1} m_{i+1} + a_i) x^{i+1} \qquad (9)\end{aligned}$$

where $\{02\}$ is represented as x in polynomial form, m_i represent the coefficients of the irreducible polynomial m defined in the AES algorithm as discussed in Section IIA, and $n = 8$.

The Hamming code of the above product $\mathbf{h}_{2\mathrm{RD}}$ is calculated as follows

$$\begin{aligned}\mathbf{h}_{2\mathrm{RD}} &= h(\{02\} \otimes a)\\ &= h\left(a_{n-1} m_0 + \sum_{i=0}^{n-2} (a_{n-1} m_{i+1} + a_i)\right). \qquad (10)\end{aligned}$$

Using the irreducible polynomial given by (3) and $n = 8$, (10) can be rewritten as

$$\mathbf{h}_{2\mathrm{RD}} = h\left(a_7 + \sum_{i=0}^{6} (a_7 m_{i+1} + a_i)\right). \qquad (11)$$

Unlike the calculation of parity bit of a byte, the calculation of Hamming parity bits depends on the position of the bits in a byte and therefore it is not possible to further simplify (11). Hence the $\mathbf{h}_{2\mathrm{RD}}$ parity bits are calculated beforehand, and are stored in the form of a memory table, which is referred to as the $\mathbf{h}_{2\mathrm{RD}}$ table.

The Hamming table $\mathbf{h}_{3\mathrm{RD}}$ is given by the parity check bits of ($\mathbf{S}_{\mathrm{RD}} \otimes \{03\}$). The Galois field multiplication of a state byte a by $\{03\}$ can be described as follows.

$$\begin{aligned}\{03\} \otimes a &= (\{02\} \oplus \{01\}) \otimes a\\ &= x \cdot a(x) \bmod m(x) \oplus a(x) \bmod m(x).\end{aligned} \qquad (12)$$

Similar to the parity function, the Hamming function is also a linear operator. The Hamming code of the above product $\mathbf{h}_{3\mathrm{RD}}$ is written as follows.

$$\mathbf{h}_{3\mathrm{RD}} = h(\{03\} \otimes a) = \mathbf{h}_{2\mathrm{RD}} \oplus \mathbf{h}_{\mathrm{RD}}. \qquad (13)$$

Hence, the $\mathbf{h}_{3\mathrm{RD}}$ parity bits can be calculated from the $\mathbf{h}_{\mathrm{RD}}$ and $\mathbf{h}_{2\mathrm{RD}}$ parity bits and therefore it is not necessary to store them in the form of a parity memory table. Once we have all the parity bits, the next step is to detect and correct the faults by predicting the Hamming code bits using the precalculated Hamming code bits.

2) *Detection and Correction of Fault Using Hamming Code Bits*: The Hamming code matrix of the SubBytes transformation is predicted by referring to the $\mathbf{h}_{\mathrm{RD}}$ table. The Hamming code matrix prediction for ShiftRows involves a simple cyclic rotation of the SubBytes Hamming code bits. The Hamming code state matrix for MixColumns is predicted with the help of the $\mathbf{h}_{\mathrm{RD}}$, $\mathbf{h}_{2\mathrm{RD}}$ and $\mathbf{h}_{3\mathrm{RD}}$ parity bits and it is expressed by the equations below:

$$\begin{aligned}\mathbf{h}_{0,j} &= \mathbf{h}_{2\mathrm{RD}}[a_{0,j}] \oplus \mathbf{h}_{3\mathrm{RD}}[a_{1,j}] \oplus \mathbf{h}_{\mathrm{RD}}[a_{2,j}] \oplus \mathbf{h}_{\mathrm{RD}}[a_{3,j}]\\ \mathbf{h}_{1,j} &= \mathbf{h}_{\mathrm{RD}}[a_{0,j}] \oplus \mathbf{h}_{2\mathrm{RD}}[a_{1,j}] \oplus \mathbf{h}_{3\mathrm{RD}}[a_{2,j}] \oplus \mathbf{h}_{\mathrm{RD}}[a_{3,j}]\\ \mathbf{h}_{2,j} &= \mathbf{h}_{\mathrm{RD}}[a_{0,j}] \oplus \mathbf{h}_{\mathrm{RD}}[a_{1,j}] \oplus \mathbf{h}_{2\mathrm{RD}}[a_{2,j}] \oplus \mathbf{h}_{3\mathrm{RD}}[a_{3,j}]\\ \mathbf{h}_{3,j} &= \mathbf{h}_{3\mathrm{RD}}[a_{0,j}] \oplus \mathbf{h}_{\mathrm{RD}}[a_{1,j}] \oplus \mathbf{h}_{\mathrm{RD}}[a_{2,j}] \oplus \mathbf{h}_{2\mathrm{RD}}[a_{3,j}]\\ & \qquad 0 \le j < 4. \qquad (14)\end{aligned}$$

By substituting (13) in (14), the Hamming code matrix for MixColumns can be predicted with just two tables, $\mathbf{h}_{\mathrm{RD}}$ and $\mathbf{h}_{2\mathrm{RD}}$.

As shown in Fig. 8, for each transformation, the Hamming code is predicted using the input data state to the transformation by referring to the parity check bit tables and also the parity check bits are calculated from the output of the transformation. The predicted and calculated check bits are compared with detect and correct the fault as discussed below.

Let the predicted check bits of the transformation input be represented by (x_3, x_2, x_1, x_0) and the calculated check bits of the transformation output be represented by (y_3, y_2, y_1, y_0). The location of the faulty bit is detected by comparing the predicted and calculated Hamming check bits following the bit match patterns in Table III. Once the faulty bit position is identified, the fault correction is performed by simply flipping that bit. The encryption is then continued without any interruption to the encryption process. Here we assume that the Hamming code tables will be protected from SEUs by traditional memory protection techniques in satellite applications like memory scrubbing and refreshing [27].

B. Software Simulation

The AES fault-tolerant model was verified using a purpose-built software simulator written in the JAVA programming language. The model is tested through injecting faults randomly at different round, transformation, byte, and bit levels. A graphical user interface (GUI) is also developed to effectively

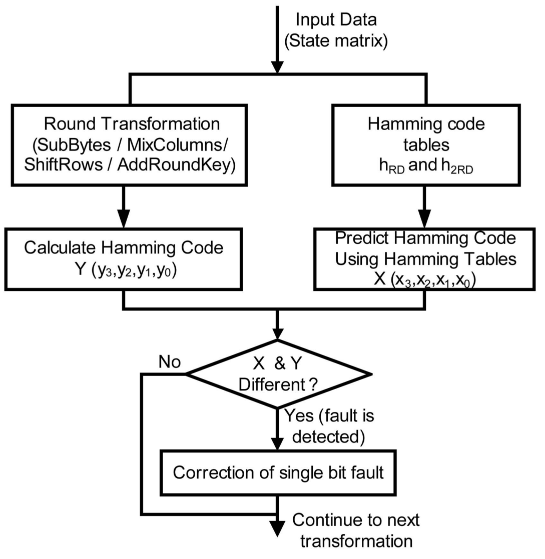

Input Data (State matrix)
Round Transformation (SubBytes / MixColumns/ ShiftRows / AddRoundKey)
Hamming code tables h_RD and h_2RD
Calculate Hamming Code Y (y3,y2,y1,y0)
Predict Hamming Code Using Hamming Tables X (x3,x2,x1,x0)
X & Y Different ?
No
Yes (fault is detected)
Correction of single bit fault
Continue to next transformation

Fig. 8. Fault detection and correction flow chart.

TABLE III
Hamming Code Bit Match Table to Locate Faulty Bit

Hamming Code Bits Comparison	Faulty Bit Position in Output
(x_3,y_3) & (x_2,y_2)	0
(x_3,y_3) & (x_1,y_1)	2
(x_3,y_3) & (x_0,y_0)	5
(x_2,y_2) & (x_1,y_1)	3
(x_2,y_2) & (x_0,y_0)	6
(x_1,y_1) & (x_0,y_0)	7
(x_1,y_1)	1
(x_0,y_0)	4

simulate fault injection and correction. The GUI, as shown in Fig. 9, has three frames. The "input" frame serves to display the input data block, encryption key, cipher block, and decryption block, etc. The "inject error" frame allows the user to simulate injection of errors at different round, transformation, byte, and bit positions. The "details" frame in the bottom half of the window shows the intermediate state of the output for every transformation and for every round in AES. The "details" subframe also shows the predicted and calculated Hamming code. Fig. 9 shows a simulator window where the error is injected at round number 4, in ShiftRows transformation, at the 6th bit position of the 5th state byte. The status bar at the lower end of the screen displays the result of the fault detection and correction technique as discussed above.

The proposed model was tested extensively using the Known Answer Test (KAT) and Monte Carlo Test (MCT) vectors specified by NIST [28, 29]. The testing with the software simulator has shown that the fault-tolerant scheme using the Hamming codes (12,8) is able to detect and correct all the faults up to bit level as expected.

V. HARDWARE IMPLEMENTATION OF THE AES FAULT-TOLERANT MODEL

In EO satellites high throughput encryption processing is required to comply with high-rate data transmission bandwidth of up to a few hundred Mbit/s [2]. In order to meet the requirement for high throughput processing, hardware implementation is considered to be the preferred choice on board satellites. Advantages of FPGAs such as flexibility of design, shorter time-to-market, lower cost, remote configurability etc., make them a suitable hardware implementation approach for use in satellites.

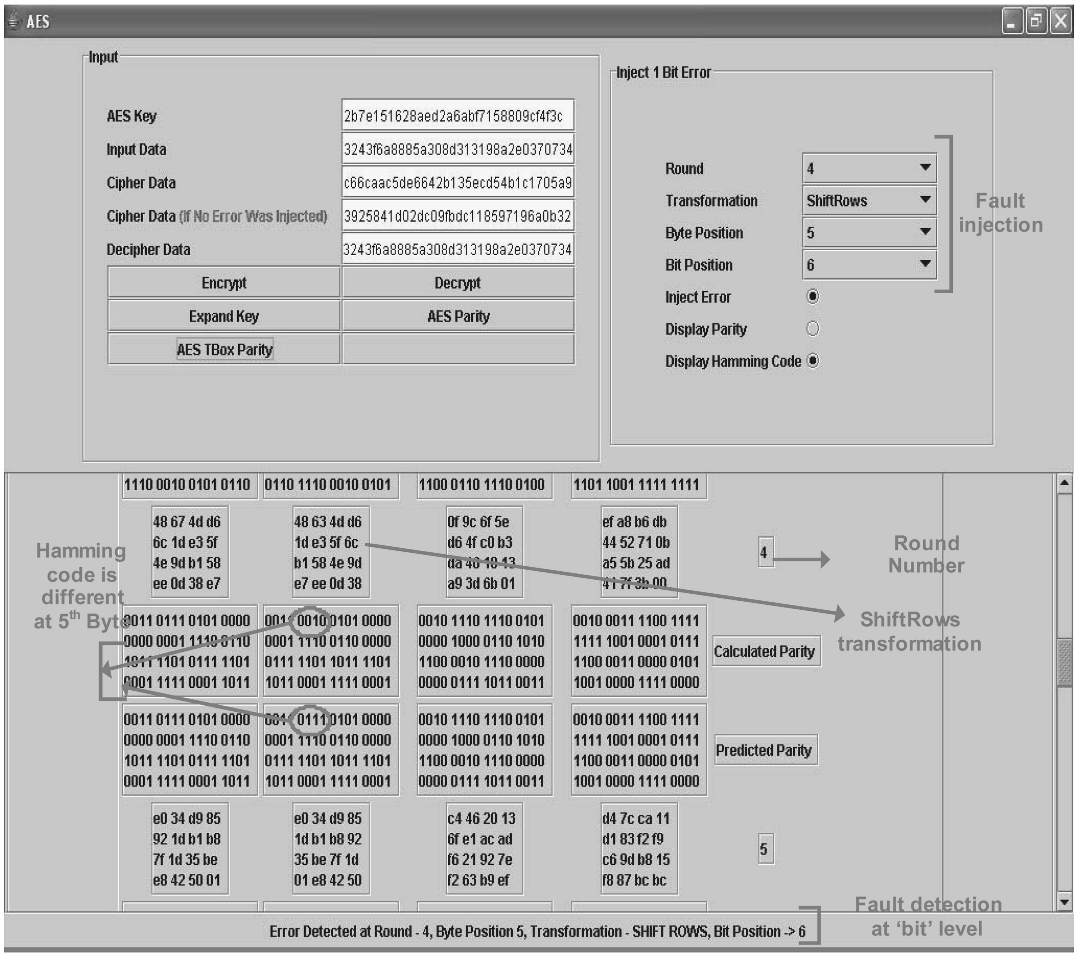
FPGA implementation of the AES algorithm and the proposed fault-tolerant model is carried out in order to explore the design space and to calculate the hardware overhead incurred by the AES EDAC. The Verilog hardware description language (HDL) is used for the coding, and modelsim is used for the functional, presynthesis, and postsynthesis simulations of the design. The HDL designs are tested extensively using the KAT and MCT vectors provided by NIST [28, 29]. Synthesis and implementation are carried out using Synplify and Xilinx ISE, respectively. The XPower tool from Xilinx is used for power estimation of the designs [30]. The target FPGA for the implementation is Xilinx Virtex-II XC2V1000.

A. FPGA Implementation of AES

Different approaches to the algorithmic realization of the SubBytes and MixColumns transformations can be adopted depending on the design goals [11]. For instance, the SubBytes transformation can be implemented using a LUT approach or a non-LUT approach. In the non-LUT approach SubBytes can be calculated on the fly for each state byte using Galois field inversion. Alternatively, the SubBytes transformation is computed in advance and the results are stored in the S-Box.

Similarly, the MixColumns transformation can be implemented using a LUT or a non-LUT approach. In the non-LUT approach every column in the state is multiplied with a predefined polynomial {2 3 1 1}. Alternatively, the two precalculated tables ($\mathbf{S}_{RD} \otimes$ {02}) and ($\mathbf{S}_{RD} \otimes$ {03}) can be used to carry out this transformation, which is called a T-Box approach. Instead of storing only the value of SubBytes in the S-box approach, the T-box approach stores the values of ($\mathbf{S}_{RD}$), ($\mathbf{S}_{RD} \otimes$ {02}) and ($\mathbf{S}_{RD} \otimes$ {03}). The other two transformations ShiftRows and AddRoundKey are implemented by just a cyclic left shift and an XOR operation, respectively.

The realisation of the SubBytes and MixColumns transformations using LUT and non-LUT approaches gives rise to three AES implementation options as detailed in Table IV. Option1 in Table IV, which


AES

Input

AES Key: 2b7e151628aed2a6abf7158809cf4f3c
Input Data: 3243f6a8885a308d313198a2e0370734
Cipher Data: c66caac5de6642b135ecd54b1c1705a9
Cipher Data (If No Error Was Injected): 3925841d02dc09fbdc118597196a0b32
Decipher Data: 3243f6a8885a308d313198a2e0370734

Inject 1 Bit Error

Round: 4
Transformation: ShiftRows
Byte Position: 5
Bit Position: 6
Inject Error: ●
Display Parity: ○
Display Hamming Code: ●

```
1110 0010 0101 0110   0110 1110 0010 0101   1100 0110 1110 0100   1101 1001 1111 1111

    48 67 4d d6           48 63 4d d6           0f 9c 6f 5e           ef a8 b6 db
    6c 1d e3 5f           1d e3 5f 6c           d6 4f c0 b3           44 52 71 0b
    4e 9d b1 58           b1 58 4e 9d           da 40 18 13           a5 5b 25 ad       4
    ee 0d 38 e7           e7 ee 0d 38           a9 3d 6b 01           41 7f 3b 00

0011 0111 0101 0000   0011 0010 0101 0000   0010 1110 1110 0101   0010 0011 1100 1111
0000 0001 1110 0110   0001 1110 0110 0000   0000 1000 0110 1010   1111 1001 0001 0111
1011 1101 0111 1101   0111 1101 1011 1101   1100 0010 1110 0000   1100 0011 0000 0101   Calculated Parity
0001 1111 0001 1011   1011 0001 1111 0001   0000 0111 1011 0011   1001 0000 1111 0000

0011 0111 0101 0000   0011 0111 0101 0000   0010 1110 1110 0101   0010 0011 1100 1111
0000 0001 1110 0110   0001 1110 0110 0000   0000 1000 0110 1010   1111 1001 0001 0111
1011 1101 0111 1101   0111 1101 1011 1101   1100 0010 1110 0000   1100 0011 0000 0101   Predicted Parity
0001 1111 0001 1011   1011 0001 1111 0001   0000 0111 1011 0011   1001 0000 1111 0000

    e0 34 d9 85           e0 34 d9 85           c4 46 20 13           d4 7c ca 11
    92 1d b1 b8           1d b1 b8 92           6f e1 ac ad           d1 83 f2 f9
    7f 1d 35 be           35 be 7f 1d           f6 21 92 7e           c6 9d b8 15       5
    e8 42 50 01           01 e8 42 50           f2 63 b9 ef           f8 87 bc bc
```

Error Detected at Round - 4, Byte Position 5, Transformation - SHIFT ROWS, Bit Position -> 6

Fig. 9. JAVA GUI to simulate fault injection and detection at 'bit' level.

is a combination of LUT and combinational logic, is the most widely adopted approach to the AES hardware implementations [11,12]. Option2 is implemented using combinational logic entirely. Option3 is purely a LUT approach, where both the SubBytes and MixColumns LUTs are merged together and implemented using the S-box and T-boxes.

The three options in Table IV are used to implement AES. The designs are realized as soft intellectual property (IP) cores written in Verilog HDL as follows.

Option1. An existing open-source AES IP core (AES (Rijndael) core) is used. The core is provided at the OPENCORES website [31].

Option2. The core is also an open-source AES IP core (128 bit AES core), from the OPENCORES website [31].

Option3. The core is developed in-house.

To study the effect of these implementation options on design parameters, the three AES cores are implemented on the XC2V1000 device from the Xilinx Virtex-II family of FPGAs. For all the three options, throughput, area, power & energy consumption estimations are carried out based on back-annotated simulations. Table V presents throughput, and total power consumption at 25 MHz frequency of operation. The device utilization is represented as percentage of the total available 5120 slices and 40 block RAMs (BRAM).

TABLE IV
Implementation Options of AES Algorithm

Implementation Approach	Transformation	
	SubBytes	MixColumns
Option1	LUT (S-Box)	Non-LUT (Multiplication in Galois field)
Option2	Non-LUT (Composite field arithmetic)	Non-LUT (Multiplication in Galois field)
Option3	LUT (S-Box)	LUT (T-Boxes)

The throughput of the AES implementations is calculated using the equation below:

$$\text{Throughput} = \left(\frac{128}{n}\right) \cdot f \qquad (15)$$

TABLE V
AES Implementations using Xilinx Virtex-II Device XC2V1000 (On-Line Key Expansion)

Device is XC2V1000 FF 896 −6 & $f = 25$ MHz							
Option No (datapath width)	Throughput (Mbit/s)	Total Power (mW)	Execution Time* (ms)	Energy (mJ)	Device Utilization		Maximal Frequency $f_{\max}$ (MHz)
					Slices	BRAM	
Option1 (128 bit)	267	885	68	60.2	452 (8%)	20 (50%)	111
Option2 (8 bit)	25.1	548	720	394.5	994 (19%)	0 (0%)	72
Option3 (128 bit)	267	1130	68	76.8	1226 (23%)	40 (100%)	90

Note: *Execution time is estimated as follows: the product of the number of blocks in the test image and the number of clock cycles to encrypt one block is divided by the frequency.

where n is the number of clock cycles required to encrypt a single block of 128 bits and f is the frequency of operation.

Table V also presents the estimated execution time and energy required to encrypt the test image which is shown in Fig. 2(a). The image has 871 (W) $\times$ 868 (H) pixels and each pixel is of 24 bits, representing three spectral bands with 8 bits per band. Thus, the number of 128 bit blocks for this image is 141755.

From Table V, we can observe that the Option1 and Option3 IP cores deliver a similar throughput of 267 Mbit/s whereas the Option2 IP core delivers a much lower throughput of 25 Mbit/s. This is because Option2 is implemented using 8 bit datapath architecture and operates on a single state byte at a time whereas Option1 and Option3 use a 128 bit architecture and operate on the whole state matrix (16 bytes) at once. Correspondingly, Option1 and Option3 take 12 clock cycles whereas Option2 takes 127 clock cycles for encryption of one block of data. It is also observed that even though Option1 and Option3 achieve identical throughput, Option3 consumes more power, energy and area (slices and BRAMs) compared with Option1. This is because the Option3 implementation uses LUTs (T-box & S-box).

The Option2 IP core consumes less power compared with Option1. This is mainly because Option2 architecture employs an 8 bit datapath and hence uses less FPGA resources compared with Option1. However, power consumption of Option2, even though low, is comparable to Option1. This is because, in Option2 SubBytes is implemented using combinational logic and hence the switching activity will be higher compared with Option1, which uses an S-Box LUT. Combinational logic is mapped onto the configurable logic block (CLB) slices of the FPGA whereas LUTs are mapped onto the embedded BRAMs. In Xilinx FPGAs, the design mapped onto the embedded resources (BRAM) consumes less power compared with the design mapped onto the CLB slices [32]. However as it is seen from Table V, Option2 consumes much more energy than Option1, due to taking more clock cycles, i.e., longer time for encryption.

From the above analysis, it can be concluded that the Option1 IP core consumes less energy and occupies less area than the other options and hence it is the best choice among the three.

B. FPGA Implementation of the Fault-Tolerant AES Model

In order to implement and calculate the overhead of the fault-tolerant AES model, the Hamming code based EDAC is incorporated into the Option1 AES IP core, which is identified as the optimal implementation in terms of power, throughput, and area in Section VA above. Two additional Hamming tables $\mathbf{h}_{\text{RD}}$ and $\mathbf{h}_{2\text{RD}}$ are calculated using expression (8) and $\mathbf{h}_{3\text{RD}}$ using (13). For comparison purposes the EDAC function is also incorporated in the Option3 AES IP core which uses LUTs for both SubBytes and MixColumns.

The Hamming code prediction path is implemented as follows. For the SubBytes transformation the Hamming code is predicted using the $\mathbf{h}_{\text{RD}}$ table. For ShiftRows it is just a cyclic left shift operation of the Hamming code and it is realized as a shift-by-wire. The Hamming code prediction for the MixColumns transformation is carried out using (14) with the help of the Hamming code tables $\mathbf{h}_{\text{RD}}$ and $\mathbf{h}_{2\text{RD}}$. An XOR gate is used for the prediction of the parity check bits for the AddRoundKey transformation. At each transformation the parity check bits are calculated and compared with the predicted one. Comparators are included at each step to compare the predicted and the calculated Hamming codes. If they differ an error flag is set and the error correction logic is used to identify the corrupted bit as described in Table III.

The FPGA implementations of the AES fault-tolerant model for Option1 and Option3 are quantified in terms of area, power overhead, and maximum frequency of operation as shown in Table VI and Table VII, respectively. For both

TABLE VI
FPGA Implementation of Fault-Tolerant Model with Option1 AES

XC2V1000 & f = 25 MHz			
Option1 IP Core	FPGA Utilization	Power (mW)	$f_{\max}$
Option1 AES	452 (8%) 20 (50%)	885	111
Option1 AES+EDAC	675 (13%) 39 (97%)	2234	91.43
Overhead	49%	152%	−17%

TABLE VII
FPGA Implementation of Fault-Tolerant Model with Option3 AES

XC2V1000, f = 25 MHz			
Option3 IP Core	FPGA Utilization	Power (mW)	$f_{\max}$
Option3 AES	1226 (23%) 40 (100%)	1130	90
Option3 AES+EDAC	1695 (33%) 40 (100%)	2412	75.4
Overhead	38%	131%	−16%

fault-tolerant AES implementations, Option1+EDAC and Option3+EDAC, the throughput remains 267 Mbit/s at 25 MHz. However, as can be seen from Table VI and Table VII the maximum frequency of operation is lower, which is due to the addition of Hamming code comparators in the datapath. It is clear from Table VI and Table VII that the Option3+EDAC implementation consumes more power compared with the Option1+EDAC. It can also be observed from Table VI and Table VII that the FPGA device utilization overhead for Option3+EDAC is lower by 11% and the power consumption overhead is lower by 21%. Even though the overhead of the Option3+EDAC implementation is smaller compared with Option1+EDAC, it has higher area and power consumption. Hence the Option1 based fault-tolerant implementation is a more favorable option for on-board use. Compared with the TMR technique for SEU mitigation [14], the proposed approach provides better results in terms of area and power.

VI. CONCLUSIONS

This paper investigates the reliability of the AES algorithm for use on board EO small satellites. Popular modes of AES such as ECB, CBC, OFB, CFB, and CTR are discussed in detail and their advantages and disadvantages for encryption of satellite multispectral images are compared. The impact of the propagation of SEU faults occurring during on-board encryption is analyzed. In addition, an analysis of the propagation of faults that occur during transmission due to noise is carried out.

In order to avoid data corruption due to SEUs, a fault detection and correction model of AES is proposed based on the Hamming code (12,8). The model provides an SEU self-recovering capability, which is built in the AES datapath. Also FPGA implementation is carried out to calculate the area and power consumption overhead of the proposed fault correction model. The fault-tolerant model of the AES provides adequate processing speed of 267 Mbit/s on a Xilinx Virtex-II FPGA, which is in excess of the typical data rate in small EO satellites of 25 Mbit/s. Also it consumes a very small portion of the power available to the payload unit. The estimated hardware overhead of the optimal fault-tolerant AES IP core is 49% in terms of area and 152% in terms of power. The model can be extended for detection and correction of multiple bit faults by using other more sophisticated error-correcting codes such as modified Hamming code, Reed-Solomon codes, etc.

The proposed fault detection and correction AES model targets the satellite application domain, however it can also be used in other applications aimed at hostile environments such as nuclear reactors, interplanetary exploration, unmanned aerial vehicles, etc. Terrestrial applications, which require a high level of reliability, such as bank servers, telecommunication servers, etc. can benefit from the use of AES fault-tolerant techniques too.

ACKNOWLEDGMENTS

Provision of satellite images for this research by SSTL is gratefully acknowledged.

REFERENCES

[1] Sun, W., Stephens, P., and Sweeting, M. N.
Micro-minisatellites for affordable EO constellations—RapidEye and DMC.
In *Proceedings of the IAA Symposium on Small Satellites for Earth Observation*, Berlin, Germany, Apr. 2001, IAA-B3-0603.

[2] Surrey Satellite Technology Ltd.
www.sstl.co.uk (last accessed on 18th June 2007).

[3] Directory of Earth Observation Resources.
http://directory.eoportal.org/pres_TopSat.html (last accessed 18th June 2007).

[4] Consultative Committee for Space Data Systems
Security threats against space missions.
Informational Report CCSDS 350.1-G-1, Green Book, NASA, Washington, D.C., Oct. 2006.

[5] Vladimirova, T., Banu, R., and Sweeting, M. N.
On-board encryption in satellites.
In *Proceedings of the 8th Military and Aerospace Applications of Programmable Logic Devices and Technologies International Conference* (MAPLD'2005), F-184, NASA, Washington, D.C., Sept. 2005.

[6] Sweet, K.
The increasing threat to satellite communications.
Online Journal of Space Communication, 6 (Nov. 2003).

[7] Weiss, H., and Stanier, J.
Space mission communications security.
In 5th Ground System Architecture Workshop (GSAW) 2001, http://sunset.usc.edu/events/GSAW/gsaw2001/SESSION9/Shave.pdf (last accessed 18th June 2007).

[8] Guttlich, J., Sinander, N., and Schaffner, E.
MeteoSat second generation (MSG) ground segment—LRIT/HRIT mission specific implementation.
Document EUM/MSG/SPE/057, 4.0, Sept. 21, 1999.

[9] Michalik, H., Hinsenkamp, L., and Schonenberg, A.
Secure space links—Impacts on on-board link data processing.
Data Systems In Aerospace (DASIA 2006), Berlin, Germany, May 22–25, 2006.

[10] Consultative Committee for Space Data Systems
The application of CCSDS protocols to secure systems.
Informational Report CCSDS 350.0-G-2, Green Book, NASA, Washington, D.C., Jan. 2006.

[11] Daemen, J., and Rijmen, R.
The Design of Rijndael: AES—The Advanced Encryption Standard.
New Yrok: Spriger-Verlag, 2002.

[12] Zhang, X., and Parhi, K. K.
High-speed VLSI architecture for the AES algorithm.
IEEE Transaction on VLSI Systems, **12**, 9 (Sept. 2004), 957–967.

[13] Gussenhoven, M. S., and Mullen, E. G.
Space radiation effects program: An overview.
IEEE Transactions on Nuclear Science, **40**, 2 (Apr. 1993), 221–227.

[14] Kastensmidt, F. L., Carro, L., and Reis, R.
Fault-Tolerance Techniques for SRAM-Based FPGAs.
New York: Springer, 2006.

[15] Underwood, C. I.
The
single-event effect behaviour of commercial-off-the-shelf memory devices—A decade in low Earth orbit.
IEEE Transactions on Nuclear Science, **45**, 3, Pt. 3 (June 1998), 1450–1457.

[16] Leon, A. F.
Field programmable gate arrays in space.
IEEE Instrumentation and Measurements Magazine, **6**, 4 (Dec. 2003), 42–48.

[17] Fuller, E., Caffrey, M., Salazar, A., Carmichael, C., and Fabula, J.
Radiation testing update, SEU mitigation, and availability analysis of the virtex FPGA for space reconfigurable computing.
In *Proceedings of Military and Aerospace Applications of Programmable Logic Devices and Technologies International Conference* (MAPLD 2000), Sept. 26–28, 2000.

[18] Banu, R., and Vladimirova, T.
Investigation of fault propagation in encryption of satellite images using the AES algorithm.
In *Proceedings of 25th IEEE Military Communications Conference* (MILCOM 2006), Washington, D.C., Oct. 23–25, 2006, 1–6.

[19] Vegetation images samples.
http://www.vgt.vito.be/AShtml/c24_08022000_newzealand.htm (last accessed on 10th June 2007).

[20] Burr, W. E.
Selecting the advanced encryption standard.
Security & Privacy Magazine, **1**, 2 (Mar.–Apr. 2003), 43–52.

[21] Efford, N.
Digital Image Processing: A Practical Introduction Using Java.
Reading, MA: Harlow: Addison-Wesley, 2000.

[22] Bertoni, G., Breveglieri, L., Koren, I., Maistri, P., and Piuri, V.
Error analysis and detection procedures for a hardware implementation of the AES.
IEEE Transactions on Computers, **52**, 4 (Apr. 2003), 493–505.

[23] Kim, Y.-C., Kim, K.-O., and Lee, T.-W.
VLSI implementation of an OFB processor for encryption of real-time data.
In *Proceedings of the 2nd IEEE Asia Pasific Conference on ASICs*, Aug. 28–30, 2000, 179–182.

[24] Banu, R., and Vladimirova, T.
On-board encryption in Earth observation small satellites.
In *Proceedings of 40th IEEE International Carnahan Conference on Security Technology* (ICCST 2006), Oct. 16–19, 2006, 203–208.

[25] Wicker, S. B.
Error-correction coding for digital communication and storage.
Upper Saddle River, NJ: Prentice-Hall, Jan. 1995.

[26] Luby, M. G., Mitzenmacher, M., Shokrollahi, M. A., and Spielman, D. A.
Efficient erasure correcting codes.
IEEE Transactions on Information Theory, **47**, 2 (Feb. 2001), 569–583.

[27] Mariani, R., and Boschi, G.
Scrubbing and partitioning for protection of memory systems.
In *Proceedings of the 11th IEEE International Symposium on On-Line Testing*, July 6–8, 2005, 195–196.

[28] The Rijndael Home Page
http://www.esat.kuleuven.ac.be/~rijmen/rijndael (last accessed on 18th June 2007).

[29] NIST
AES validation list, Sept. 2004.
http://csrc.nist.gov/cryptval/aes/aesval.html (last accessed 18th June 2007).

[30] XPower User Guide
www.xilinx.com/XPower (last accessed 18th June 2007).

[31] OPENCORES website
www.opencores.org (last accessed 18th June 2007).

[32] Tiwari, A.
Low power FPGA design techniques for embedded systems.
Ph.D. dissertation, Computer Science Engineering, University of Cincinnati, Cincinnati, OH, 2005.

Roohi Banu (SM'07) received the B.Tech from S.V. University, India, and the M.Tech, IIT, Chennai, India and the Ph.D. from Surrey Space Centre, University of Surrey, UK.

Prior to joining her Ph.D. she worked as a design engineer at Motorola where she was involved in System-on-Chip (SOC) ASIC design. Her research interests include SOC, FPGA, and ASIC design, low power and high-speed design techniques, cryptographic algorithms, security architectures, and fault-tolerant computing.

Tanya Vladimirova (M'96), was awarded an M.Sc. in applied mathematics by the Technical University of Sofia, Bulgaria, an M.Eng. in computer systems engineering and a Ph.D. in VLSI design by the St. Petersburg Electro-Technical University (LETI), Russia.

She is reader at the Department of Electronic Engineering of the University of Surrey, UK, and leads the Microelectronics Design and Embedded Systems research group at the Surrey Space Centre. Her research interests are in the areas of system-on-a-chip, FPGA design, image processing, intelligent embedded systems, and wireless sensor networks.