Evaluation of Format- Preserving Encryption Algorithms for Critical Infrastructure Protection

Richard Agbeyibor, Jonathan Butts, Michael Grimaila, Robert Mills

Chapter 16

EVALUATION OF FORMAT-PRESERVING ENCRYPTION ALGORITHMS FOR CRITICAL INFRASTRUCTURE PROTECTION

Richard Agbeyibor, Jonathan Butts, Michael Grimaila and Robert Mills

Abstract Legacy critical infrastructure systems lack secure communications capabilities that can protect against modern threats. In particular, operational requirements such as message format and interoperability prevent the adoption of standard encryption algorithms. Three new algorithms recommended by the National Institute of Standards and Technology (NIST) for format-preserving encryption could potentially support the encryption of legacy protocols in critical infrastructure assets. The three algorithms, FF1, FF2 and FF3, provide the ability to encrypt arbitrarily-formatted data without padding or truncation, which is a critical requirement for interoperability in legacy systems. This paper presents an evaluation of the three algorithms with respect to entropy and operational latency when implemented on a Xilinx Virtex-6 (XC6VLX240T) FPGA. While the three algorithms inherit the security characteristics of the underlying Advanced Encryption Standard (AES) cipher, they exhibit some important differences in their performance characteristics.

Keywords: Format-preserving encryption, legacy infrastructure assets

1. Introduction

Legacy industrial control systems were developed and implemented well before the threats associated with modern networking were recognized. The trend to interconnect industrial control systems, however, has introduced many security concerns [26]. The systems were designed for performance, reliability and safety using proprietary hardware, software and communications protocols. The communications protocols incorporate basic error detection and correction functionality, but lack the secure communications capabilities required by mod-

ern interconnected systems. Many legacy protocols associated with industrial control systems are incompatible with modern IP-based security such as message encryption. Arbitrarily-formatted data associated with control operations cannot be padded or truncated; this prevents the use of standard encryption that relies on fixed message data lengths (e.g., the 128 or 256 block size associated with the Advanced Encryption Standard (AES)).

The Computer Security Act of 1987 assigned the National Institute of Standards and Technology (NIST) the task of developing security standards and guidelines to secure sensitive federal information and communications systems [15]. Among the most sensitive of these federal systems are those categorized as critical infrastructure assets. According to Executive Order 13636 of February 2013, "the cyber threat to critical infrastructure continues to grow and represents one of the most serious national security challenges" [17]. Executive Order 13636 also defines critical infrastructure as "the assets, whether physical or virtual, so vital to the United States that the incapacity or destruction of such systems and assets would have debilitating impact on security, national economic security, national public health or safety, or any combination of those matters" [17].

In response, NIST started the development of a cybersecurity framework in collaboration with researchers and stakeholders from the telecommunications, energy, financial services, manufacturing, water, transportation, healthcare, and emergency services sectors [16]. The interconnected nature of systems used in these sectors requires comprehensive risk management and infrastructure assurance plans. A major concern in critical infrastructure protection is the ubiquity of systems that employ aging (legacy) technologies with limited security functionality. Many of the legacy communications protocols used in sectors such as energy and transportation are incompatible with modern IP-based security, but are too costly to replace.

An important focus of NIST is the development of cryptographic standards. Cryptography includes the algorithms used to encrypt and decrypt information, and to perform other security functions such as digital signatures, authentication and key exchange [15]. One notable success is the adoption of the Advanced Encryption Standard (AES), the gold standard for symmetric-key encryption.

In July 2013, NIST released Draft Special Publication 800-38G, which recommends methods for format-preserving encryption (FPE) [6]. FPE allows the encryption of data with non-standard formats that are not suitable for modification (e.g., information transmitted in non-IP networks or stored in legacy databases). FPE can potentially provide security to legacy critical infrastructure systems that were not designed with security in mind and that are incompatible with standard encryption technology. This paper investigates the security and performance of the three NIST-recommended FPE algorithms for use in critical infrastructure protection.

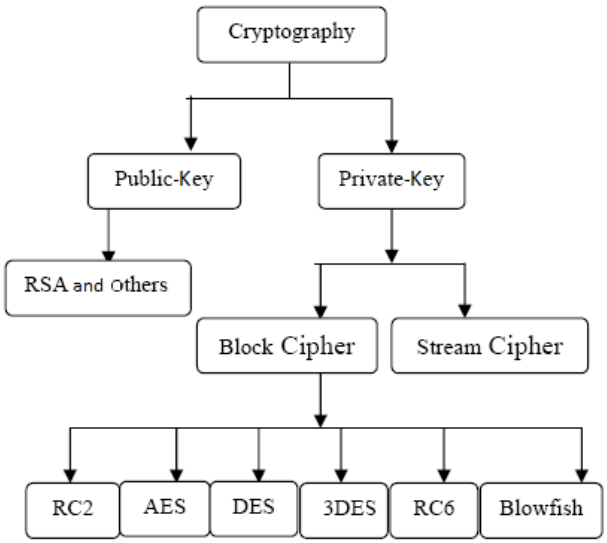

Figure 1. Modern cryptography hierarchy [1].

2. Background

Encryption is the mathematical manipulation of data in a manner that makes it unintelligible to unauthorized parties, yet recoverable by intended recipients [27]. Figure 1 shows the modern cryptography hierarchy. Cryptographic algorithms can be categorized as symmetric or asymmetric algorithms, also known as private-key or public-key algorithms, respectively. Symmetric algorithms use the same key for encryption and decryption; the key must be distributed offline or via a secure key distribution protocol. Asymmetric algorithms use two keys: one for encryption and the other for decryption. One of the keys (private key) is kept secret by one party; the other key (public key) can be distributed openly. This resolves the problem of key distribution, but asymmetric algorithms are typically more complex and computationally intensive than symmetric algorithms.

Cryptographic algorithms operate as block ciphers or stream ciphers. Stream ciphers encipher the plaintext one character at a time and concatenate the independent encryptions to produce the ciphertext. Stream ciphers are fast, but are prone to weaknesses with regard to integrity protection and authentication [27]. On the other hand, block ciphers are slower, but their mechanisms ensure the security properties of confusion and diffusion. Confusion means that the key does not relate in a simple manner to the ciphertext; it refers to making the relationship as complex as possible using the key non-uniformly throughout the encryption process. Diffusion means that changing a single character in the plaintext causes several characters in the ciphertext to change, and vice versa [27]. Block ciphers are widely used in modern cryptography, and three in particular – AES, 3DES and Skipjack – are recommended for use by NIST [6].

AES, 3DES and Skipjack are applied to 64-bit or 128-bit blocks of data. When AES was designed, 128-bit message blocks were commonly used for cryptographic applications [22]. Messages that do not fit the prescribed block size are padded or truncated. However, many supervisory control and data acquisition (SCADA) systems used in the critical infrastructure do not permit padding. SCADA systems traditionally use low-bandwidth links and compact communications protocols such as Modbus and DNP3 [28]. Solutions have been developed to retrofit security in these systems, but they often incur significant processing and buffering overhead that cannot be tolerated in systems with strict timing constraints [28]. A preferred solution is an algorithm that can transform formatted data into a sequence of symbols such that the encrypted data has the same format and length as the original data [22].

2.1 Format-Preserving Encryption

The origins of the format-preserving encryption (FPE) problem go back 32 years. In 1981, the U.S. National Bureau of Standards (later renamed NIST) published FIPS 74 that described an approach for enciphering arbitrary strings over an arbitrary alphabet [2]. The scheme was subsequently proven to be insecure. It was not until 1997 that Brightwell and Smith [5] specifically mentioned the FPE problem and its utility, which they referred to as "datatype-preserving encryption" [5]. In 2002, Black and Rogaway [3] published a seminal paper that proposed three methods for ciphers with arbitrary finite domains: a prefix method, a cycle-walking cipher and a Feistel construction. The first two methods have strong security bounds, but are targeted for tiny-space and small-space messages. In the case of tiny-space FPE, the size of the message space $N = |X|$ is so small that it is feasible to spend $O(N)$ time or $O(N)$ space for encryption or decryption. For small-space FPE, the size of the message space $N = |X|$ is at most 2^w where w is the block size of the cipher underlying the FPE scheme. AES is most often used as the block cipher, so $w = 128$ bits and $N = 2^{128} \approx 10^{38.5}$ becomes the cutoff for "small" [12]. The third method encrypts a much wider variety of data using the Feistel construction that was first examined by Luby and Rackoff in 1988 [10]. The Feistel construction has the desirable property that ciphers built from it can be proven to reduce to the cipher that is used as a round function [19].

In 2003, Spies [24] proposed the FFSEM algorithm that employs the Feistel construction for FPE. The development of FFSEM was motivated by the desire to add security to legacy protocols and systems in the financial services sector [25]. In these systems, one of the barriers to adopting effective encryption methods was the cost of modifying databases and applications to accommodate encrypted information. Applications often expect input in specific formats. Moreover, data such as social security numbers and personal account numbers are often used as keys or indices in databases, so any randomization of these fields by a randomized or stateful algorithm can require significant schema changes [22]. In 2010, Bellare, *et al.* [2] submitted specifications for FFX, a format-preserving, Feistel-based encryption. Note that the

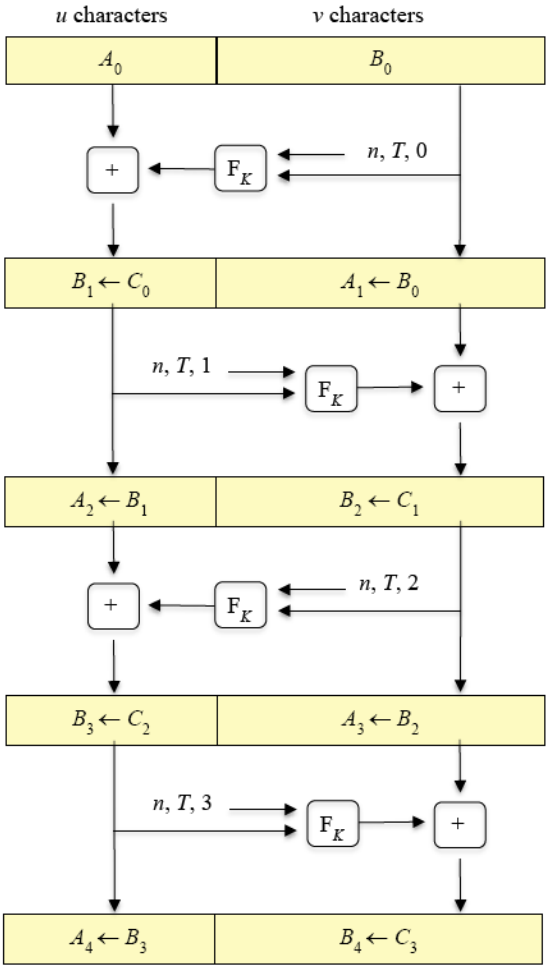

Figure 2. Feistel structure of the FF1, FF2 and FF3 algorithms [6].

"X" in FFX is a placeholder for implementations of the algorithm that are tailored to particular applications.

NIST Recommendations. The release of the FFX specification raised awareness about the FPE problem and encouraged security researchers to develop alternative algorithms. After nearly two years of deliberation, NIST released a draft of Special Publication 800-38 [6] for public comment. The publication specifies three FPE methods: FF1, FF2 and FF3. Each of these methods is a mode of operation of the AES algorithm, which is used to construct a round function within the Feistel structure for encryption as shown in Figure 2.

Algorithm 1 FF1.Encrypt(K,T,X) [4]

Prerequisites:
Approved, 128-bit block cipher, $CIPH$;
Key, K, for the block cipher;
Base, $radix$, for the character alphabet;
Range of supported message lengths, $[minlen..maxlen]$;
Maximum byte length for tweaks, $maxTlen$.

Inputs:
Character string, X, in base $radix$ of length n such that $n \in [minlen..maxlen]$;
Tweak T, a byte string of byte length t, such that $t \in [0..maxTlen]$.

Output:
Character string, Y, such that $LEN(Y) = n$.

Steps:

1: Let $u = \lfloor n/2 \rfloor$; $v = n - u$.
2: Let $A = X[1..u]$; $B = X[u+1..n]$.
3: Let $b = \lceil \lceil vLOG_2(radix) \rceil / 8 \rceil$; $d = 4\lceil b/4 \rceil + 4$.
4: Let $P = [1]^1 \| [2]^1 \| [radix]^3 \| [10]^1 \| [u \bmod 256]^1 \| [n]^4 \| [t]^4$.
5: **for** $i \leftarrow 0$ to 9 **do**
6: Let $Q = T \| [0]^{(-t-b-1) \bmod 16} \| [i]^1 \| [NUM_{radix}(B)]^b$.
7: Let $R = PRF(P \| Q)$.
8: Let S be the first d bytes of the following string of $\lceil d/16 \rceil$ blocks:
$R \| CIPH_k(R \oplus [1]^{16}) \| CIPH_k(R \oplus [2]^{16}) \| .. \| CIPH_k(R \oplus [\lceil d/16 \rceil - 1]^{16})$.
9: Let $y = NUM_2(S)$.
10: **If** i is even, let $m = u$; **Else**, let $m = v$.
11: Let $c = (NUM_{radix}(A) + y) \bmod radix^m$.
12: Let $C = STR^m_{radix}(c)$.
13: Let $A = B$.
14: Let $B = C$.
15: **end for**
16: Return $A \| B$.

Figure 3. FF1 encryption algorithm adapted from [6].

- **FF1 Algorithm:** The FF1 algorithm is derived from FFX as proposed by Bellare, *et al.* [2]. Figure 3 describes the FF1 algorithm. The NIST recommendation designates a maximally balanced Feistel structure that for an odd length message of size n divides the message into A and B halves of size $u = \lfloor n/2 \rfloor$ and $v = n - u$. The original FFX algorithm uses an alternating-Feistel structure, leaving the user to choose the size of the halves along with eight other parameters. Of the three recommendations, FF1 supports the greatest range of lengths for formatted data and the tweak.

Algorithm 2 FF2.Encrypt(K,T,X) [4]

Prerequisites:
Approved, 128-bit block cipher, $CIPH$;
Key, K, for the block cipher;
Base, $radix$, for the character alphabet;
Base, $tweakradix$, for the tweak character alphabet;
Range of supported message lengths, $[minlen..maxlen]$;
Maximum supported tweak length, $maxTlen$.

Inputs:
Numeral string, X, in base $radix$ of length n such that $n \in [minlen..maxlen]$;
Tweak numerical string, T, in base $tweakradix$ of length t such that $t \in [0..maxTlen]$.

Output:
Character string, Y, such that $LEN(Y) = n$.

Steps:

1: Let $u = \lfloor n/2 \rfloor$; $v = n - u$.
2: Let $A = X\lfloor 1..u \rfloor$; $B = X\lfloor u + 1..n \rfloor$.
3: **If** $t > 0$, $P = [radix]^1 \parallel [t]^1 \parallel [n]^1 \parallel [NUM_{tweakradix}(T)]^{13}$;
Else $P = [radix]^1 \parallel [0]^1 \parallel [n]^1 \parallel [0]^{13}$.
4: Let $J = CIPH_K(P)$.
5: **for** $i \leftarrow 0$ to 9 **do**
6: Let $Q \leftarrow [i]^1 \parallel [NUM_{radix}(B)]^{15}$.
7: Let $Y \leftarrow CIPH_J(Q)$.
8: Let $y \leftarrow NUM_2(Y)$.
9: **If** i is even, let $m = u$; **Else**, let $m = v$.
10: Let $c = (NUM_{radix}(A) + y) \bmod radix^m$.
11: Let $C = STR^m_{radix}(c)$.
12: Let $A = B$.
13: Let $B = C$.
14: **end for**
15: Return $A \parallel B$.

Figure 4. FF2 encryption algorithm adapted from [6].

- **FF2 Algorithm:** The FF2 algorithm is derived from the VAES3 algorithm proposed by Vance [29]. Figure 4 describes the FF2 algorithm, which generates a subkey for the block cipher in the Feistel round function; this can help protect the original key from side-channel analysis [6]. FF2 differs from FF1 in that it employs a larger tweak with an independent tweak radix to allow for additional variation in the cipher.

- **FF3 Algorithm:** The FF3 algorithm is essentially equivalent to the BPS-BC component of BPS [4] instantiated with a 128-bit block and limited to tiny- and small-space messages [6]. Figure 5 describes the FF3 algorithm, which has only eight rounds, but is the least flexible in terms

Algorithm 3 FF3.Encrypt(K,T,X) [4]

Prerequisites:
Approved, 128-bit block cipher, $CIPH$;
Key, K, for the block cipher;
Base, $radix$, for the character alphabet;
Range of supported message lengths, [$minlen..maxlen$], such that $minlen \geq 2$ and $maxlen \leq 2\left\lfloor log_{radix}(2^{96})\right\rfloor$.

Inputs:
Numeral string, X, in base $radix$ of length n such that $n \in [minlen..maxlen]$;
Tweak bit string, T, such that $LEN(T) = 64$.

Output:
Character string, Y, such that $LEN(Y) = n$.

Steps:

1: Let $u = \lceil n/2 \rceil$; $v = n - u$.
2: Let $A = X[1..u]$; $B = X[u + 1..n]$.
3: Let $T_L = T[0..31]$ and $T_R = T[32..63]$;
4: **for** $i \leftarrow 0$ to 7 **do**
5: **If** is even, let $m = u$ and $W = T_R$, **Else** let $m = v$ and $W = T_L$.
6: Let $P = REV([NUM_{radix}(REV(B))]^{12}) \parallel W \oplus REV([i]^4)$.
7: Let $Y = CIPH_K(P)$.
8: Let $y = NUM_2(REV(Y))$.
9: Let $c = (NUM_{radix}(REV(A)) + y) \bmod radix^m$.
10: Let $C = REV(STR^m_{radix}(c))$.
11: Let $A = B$.
12: Let $B = C$.
13: **end for**
14: Return $A \parallel B$.

*Where $REV(X)$ reverses the order of characters in the character string X

Figure 5. FF3 encryption algorithm adapted from [6].

of the tweaks that are supported. In particular, the FF3 employs a 64-bit tweak, which is split into right and left halves that are used to add diffusion to odd and even encryption rounds, respectively.

2.2 Evaluation of the Algorithms

One of the criteria used during the evaluation of the AES candidate algorithms in 1999 was demonstrated suitability as a random number generator. Specifically, the evaluation of the output utilizing statistical tests should not provide any means to distinguish it from a truly random source. NIST used several statistical tests to evaluate the AES candidates: frequency test, block frequency test, cumulative sums test, runs test, long runs of ones test, rank

test, spectral test, non-periodic templates test, overlapping template test, universal statistical test, random excursion test, random excursion variant test, Lempel-Ziv complexity test, linear complexity test and an approximate entropy test [23]. The Rijndael algorithm performed satisfactorily in all the tests and was selected as the AES algorithm.

Since FPE algorithms are modes of operation of the underlying block cipher, FF1, FF2 and FF3 should benefit from the statistical characteristics of AES. This hypothesis is supported by theoretical results [13, 19, 20]. Our evaluation uses Shannon entropy measurements to assess the security characteristics of the three FFX algorithms. Note that entropy is a measure of unpredictability or information content; Shannon entropy quantifies the expected value of the information contained in a message and is typically measured in bits per byte [27].

In addition to security performance, the computational performance of the algorithms is an important criterion. Several metrics may be used to measure the computational performance: encryption time, processing time and total clock cycles per encryption [9]. The total clock cycle metric was used in this research to evaluate the computational speed of the FF1, FF2 and FF3 algorithms.

3. Experimental Design

In order to determine the security and performance of the FF1, FF2 and FF3 algorithms for critical infrastructure assets, a set of experiments was designed to test the hypothesis suggested by the algorithm designers and NIST [6] that the algorithms inherit the strong security characteristics of the underlying block cipher. NIST has not released details of its internal deliberations and performance assessments.

As such, statistical tests were conducted to determine the ability of the FPE algorithms to provide confusion and diffusion, and to output ciphertext that is computationally indistinguishable from a random process. A dataset containing input plaintext with varying levels of entropy was created. The FF1, FF2 and FF3 algorithms were applied to this dataset. The algorithms were implemented in C using the `offspark` AES library [18] and the entropy of the resulting ciphertext was measured.

The second objective of our research was to evaluate the computational speed of the three algorithms by measuring the operational latency of a hardware implementation. This was accomplished by implementing the algorithms in VHDL using the Xilinx ISE suite for the Virtex-6 FPGA (XC6VLX240T) [31]. A hardware-agnostic design was used to mitigate effects due to the Virtex-6 CMOS technology and Xilinx FPGA architecture. The operational latency was estimated using the number of clock cycles between the input of plaintext and the output of its ciphertext.

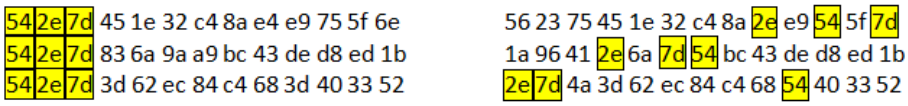

Figure 6. Plaintext strings in the 3 Front and 3 Random scenarios.

3.1 Plaintext Dataset Design

In 1999, NIST tested the ability of the AES candidate algorithms to encrypt a plaintext avalanche comprising various sequences of random and fixed plaintext bits [23]. Similarly, our research focused on the ability of the FPE algorithms to encrypt a plaintext avalanche comprising various sequences of random plaintext and fixed plaintext bytes. We studied the effect of repetitive and, thus, predictable input data on the entropy of the ciphertext. Input strings of thirteen bytes were used; this non-standard block size is used by a legacy protocol employed in the transportation sector for aircraft transponder messages [8].

Our research employed randomized experiments to enhance the reliability and validity of the statistical results. The True Random Number Generator (TRNG) service provided by Random.org [21] formed the backbone of the experimental design. Unlike pseudo-random number generators that use mathematical formulas to generate sequences of numbers that appear random, TRNG extracts randomness from physical phenomena (i.e., by measuring atmospheric noise), producing 1 MiB (2^{20} bytes) of raw random data. These true random numbers were used to create a dataset of input plaintext strings with varying levels of random and deterministic data. The experimental factor used in the study was the number of deterministic bytes in the plaintext.

The experiments involved eight scenarios in which three, six, nine and twelve bytes of the thirteen-byte plaintext string were held constant at the front or dispersed randomly throughout the string. Each of the eight scenarios was replicated 20 times in unique plaintext files, each containing 4,000 different thirteen-byte strings. In the 3 Front scenario, the first three bytes were the same in all input strings within the file. In the 3 Random scenario, the three deterministic bytes were randomly dispersed throughout the string as shown in Figure 6.

The other six scenarios followed the same design. The 4,000 thirteen-byte strings in each input file repeated the same deterministic sequence; however, each trial used a different deterministic byte sequence. The non-deterministic part of the message was composed of random data extracted from the Random.org sequence of 2013-09-17 [21].

3.2 Implementation and Entropy Measurement

All three FPE algorithms require a NIST-approved block cipher operation CIPH. The 128-bit AES algorithm was used in the implementation. The cryp-

tography community discourages the use of unverified implementations of AES because of its complexity. Therefore, we used `offspark`, a vetted open source implementation of AES used to encrypt official Dutch Government communications [18]. The `offspark` implementation is written in the C programming language, which partly motivated the use of C in the research. The research validated the `offspark` AES implementation by comparing its output with known vectors published in NIST's Known Answer Test [14].

No known answers tests exist for FF1, FF2 and FF3, nor are there any vetted implementations. Therefore, we verified our implementation via decryption. The decryption algorithms provided by NIST were applied to the ciphertext to reverse the encryption process. Satisfactorily-decrypted ciphertext provided confidence that the implementations were accurate.

Entropy is a measure of the amount of information that can be gleaned from ciphertext. The entropy $H(X)$ of a variable or distribution is defined as:

$$H(x) = -\sum p(x) log_2 p(x).$$

Comparisons of the entropy of ciphertext and the entropy of a random distribution were used to assess the security of the algorithms. The ENT tool [30] was used to measure entropy. ENT applied various statistical tests to the sequences of bytes stored in the files and reported the aggregate entropy of the 4,000 output ciphertext strings in each file.

3.3 Hardware Implementation

Hardware performance was another criterion used by NIST in 1999 to evaluate the AES candidate algorithms. The Rijndael algorithm was selected partly because it proved to be one of the fastest and most efficient algorithms, and implementable on a wide range of platforms [7].

A number of different architectures can be considered when implementing an encryption algorithm in hardware or using a field programmable gate array (FPGA). Iterative looping is where only one round is designed; hence, for an n-round algorithm, n iterations of the round are used to perform an encryption. Loop unrolling involves the unrolling of multiple rounds. Pipelining is achieved by replicating the round and placing registers between each round to control the flow of data. A pipelined architecture generally provides the highest throughput [11]. Our research employed a pipelined implementation of 128-bit AES and an iterative looping architecture for the Feistel structure of FPE. Iterative looping saves hardware resources by implementing only one round of the algorithm and using control logic to manage data flow.

The NIST pseudocode description is primarily intended for software implementations. As a consequence, certain operations that depend on previous operations require carefully synchronized logic when implemented in hardware. The pseudocode of algorithm was, therefore, expanded to identify parallelizable modules and blocks that can be implemented with combinational logic. Function calls to AES within the F-block of each round require the use of loop

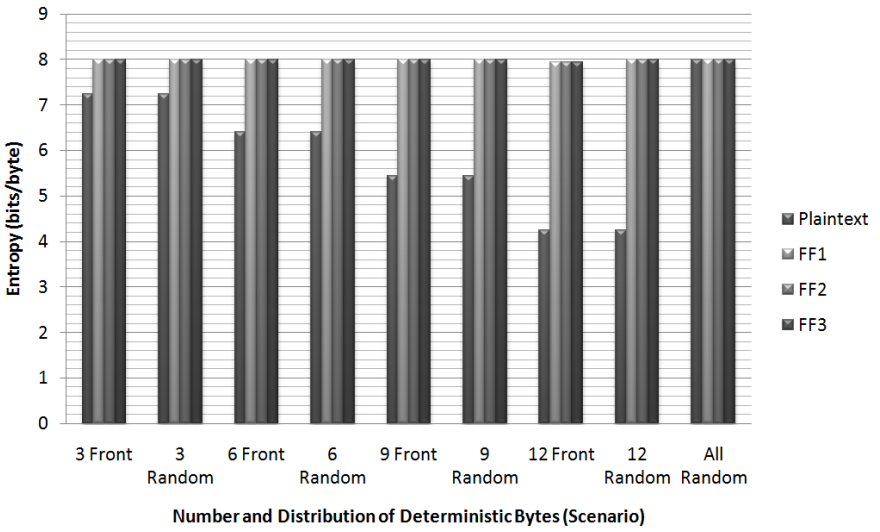

Figure 7. Mean entropy.

counters. The algorithms were coded in VHDL, simulated, placed and routed, and synthesized on a Virtex-6 (XC6VLX240T) device using the Xilinx ISE design suite. Post-PAR static timing analysis and device utilization analysis were performed on each implementation.

The throughput, latency and hardware resource requirements are usually the most critical parameters when evaluating a hardware implementation. Our research evaluated the speed of each algorithm by measuring the operational latency of an encryption cycle. To eliminate bias due to the use of a particular FPGA technology, we estimated operational latency as the number of clock cycles required for an algorithm to encrypt plaintext.

4. Results and Analysis

This section presents and analyzes the experimental results.

4.1 Security

A thirteen-byte sequence of random data obtained from Random.org served as the control in the entropy experiment. An all-random input plaintext file created with the sequence was determined to have an entropy of 7.996 bits/byte. In the following analysis, the mean entropy was calculated for 20 trials of each scenario. Note that there was no statistical significant variance between the various trials. Figure 7 and Table 1 present the security performance of each algorithm estimated in terms of the ciphertext entropy for each level of the experimental factor. As expected, the entropy decreases in the plaintext as the number of deterministic bytes increases. The input entropy ranges from

Table 1. Mean entropy.

Scenario	Plaintext	FF1	FF2	FF3
3 Front	7.240370	7.996331	7.996487	7.996388
3 Random	7.240221	7.99654	7.996503	7.996498
6 Front	6.402844	7.996482	7.996472	7.996564
6 Random	6.402425	7.996358	7.996373	7.996502
9 Front	5.448800	7.996368	7.996341	7.996511
9 Random	5.448699	7.996375	7.996554	7.996432
12 Front	4.256321	7.942249	7.939892	7.942226
12 Random	4.256166	7.996518	7.996558	7.996404
All Random	7.996332	7.996460	7.996348	7.996424

7.24 bits/byte for three deterministic bytes out of the thirteen total bytes to 4.25 bits/byte for twelve out of thirteen fixed bytes. The distribution of the deterministic bytes, whether located in the front of the string or randomly dispersed throughout the string, does not have a significant effect on the entropy of the plaintext.

All three algorithms provide high levels of ciphertext security with no discernible differences in performance. In all but one scenario (12 Front), the ciphertext is indistinguishable from a random sequence with entropy above 7.996 bits/byte. The plaintext in the 12 Front scenario with entropy of 4.256 bits/byte causes a lower entropy in the ciphertext of 7.94 bits/byte versus the 7.996 bits/byte for the random sequence. The lowered entropy presents an upper bound on the obfuscation capabilities of FPE. Further study is necessary to clarify this performance limitation and categorize suitable plaintext.

The three FPE algorithms provide higher levels of entropy when the same number of deterministic bytes are randomly distributed throughout the string in the 12 Random scenario, These results indicate that the distribution of repeated patterns in the plaintext affects the ability of the algorithms to obfuscate the data more than the amount of repeated information.

4.2 Performance

The performance results shown in Table 2 indicate that the underlying AES core is the principal factor in the area and speed of the implementation. The AES implementation employed in the designs requires 31 clock cycles per encryption and 1,864 slices (slices are the basic building blocks in an FPGA implementation). Each slice contains a number of look up tables (LUTs) that are used to implement AND gates, OR gates and other Boolean functions. In addition to LUTs, slices also contain a number of registers that hold state and are used to implement sequential logic. In the device utilization report, any slice that is used even partially is counted towards the number of occupied slices. A design may be fitted into fewer slices if necessary, but mapping unrelated logic into the same slice may impact the ability to meet timing constraints [31].

Table 2. FPGA performance.

	AES	**FF1**	**FF2**	**FF3**
Number of Slice Registers	5,801	11,285	11,323	5,592
Number of Slice LUTs	3,452	7,426	6,825	3,587
Number of Occupied Slices	1,864	3,850	3,728	1,820
Number of 18K Block RAMs	172	343	342	170
Maximum Frequency (MHz)	336.315	279.587	284.592	283.427
Clock Cycles per Round	3	68	33	32
Clock Cycles per Encryption	31	707	374	269

The Virtex-6 provides 18 Kb and 36 Kb blocks of RAM for storing data. Our implementations did not require any 36 Kb RAM blocks.

The iterative looping architecture employed in the design minimizes the hardware resources needed for each algorithm. The FF1 implementation uses two cascaded AES blocks per round, which causes the area and number of slices required to be approximately twice those of one AES block. FF2 makes only one call to AES per round, but uses an additional AES block to generate the subkey. FF3 has the smallest footprint of the three algorithms because it relies sparingly on calls to AES.

The maximum frequency is based on the worst path delay found in the design, and it indicates the fastest frequency at which a signal may be toggled given this constraint. A simulation test bench was used to measure to operational latency of each implementation. The numbers of clock cycles required for completing one round and for completing an entire encryption cycle are reported for each algorithm (Table 2). The FF1 algorithm makes two calls to AES per round, which makes it the slowest of the three algorithms. FF2 is faster than FF1 because of its single call to AES in its F-block. FF3 is the fastest of the three algorithms because it uses only eight rounds. The overall results indicate that the FF3 algorithm requires the least hardware resources and has the lowest operational latency.

5. Conclusions

The FF1, FF2 and FF3 format-preserving encryption algorithms have important applications in critical infrastructure protection. In particular, the algorithms could be incorporated in security modules for legacy protocols and databases that are currently incompatible with standard cryptographic practices.

The experimental results demonstrate that algorithms are secure based on their ability to obfuscate repetitive input data. The algorithms successfully encipher plaintext with twelve of thirteen bytes containing a deterministic sequence. The three algorithms (as recommended by NIST) demonstrate the inherited security characteristics of the underlying AES cipher.

Because the algorithms can be implemented with AES as the underlying block cipher, they are suitable for implementation on any number of hardware platforms. The characteristics of the underlying AES implementation contribute strongly to the security and speed of the three algorithms. The FF3 algorithm requires the least hardware resources, has the lowest operational latency and has similar security performance as the other two algorithms.

The results also demonstrate that the three algorithms can obfuscate data streams with large amounts of repeated data without overhead or significant hardware costs. As such, the FF1, FF2 and FF3 algorithms provide good options for retrofitting encryption in legacy critical infrastructure systems without sacrificing interoperability or performance.

The views expressed in this paper are those of the authors and do not reflect the official policy or position of the United States Air Force, Department of Defense or U.S. Government.

References

[1] D. Abdul Elminaam, D. Abdul Kader and M. Hadhoud, Perfomance evaluation of symmetric encryption algorithms, *International Journal of Computer Science and Network Security*, vol. 8(12), pp. 280–285, 2008.

[2] M. Bellare, P. Rogaway and T. Spies, The FFX Mode of Operation for Format-Preserving Encryption, Report to NIST Describing the FFX Algorithm, National Institute of Standards and Technology, Gaithersburg, Maryland, 2010.

[3] J. Black and P. Rogaway, Ciphers with arbitrary finite domains, *Proceedings of the Cryptographer's Track at the RSA Conference*, pp. 114–130, 2002.

[4] E. Brier, T. Peyrin and J. Stern, BPS: A Format-Preserving Encryption Proposal, National Institute of Standards and Technology, Gaithersburg, Maryland, 2010.

[5] M. Brightwell and H. Smith, Using datatype-preserving encryption to enhance data warehouse security, *Proceedings of the Twentieth National Information Systems Security Conference*, 1997.

[6] M. Dworkin, Recommendation for Block Cipher Modes of Operation: Methods for Format-Preserving Encryption, Draft NIST Special Publication 800-38G, National Institute of Standards and Technology, Gaithersburg, Maryland, 2013.

[7] A. Elbirt, W. Yip, B. Chetwynd and C. Paar, An FPGA-based performance evaluation of the AES block cipher candidate algorithm finalists, *IEEE Transactions on Very Large Scale Integration Systems*, vol. 9(4), pp. 545–557, 2001.

[8] C. Finke, J. Butts and R. Mills, ADS-B encryption: Confidentiality in the friendly skies, *Proceedings of the Eighth Annual Cyber Security and Information Intelligence Research Workshop*, pp. 9–13, 2013.

[9] T. Good and M. Benaissa, AES on FPGA from the fastest to the smallest, *Proceedings of the Seventh International Workshop on Cryptographic Hardware and Embedded Systems*, pp. 427–440, 2005.

[10] M. Luby and C. Rackoff, How to construct pseudorandom permutations from pseudorandom functions, *SIAM Journal on Computing*, vol. 17(2), pp. 373–386, 1988.

[11] M. McLoone and J. McCanny, High performance single-chip FPGA Rijndael algorithm implementations, *Proceedings of the Third International Workshop on Cryptographic Hardware and Embedded Systems*, pp. 65–76, 2001.

[12] B. Morris, P. Rogaway and T. Stegers, How to encipher messages on a small domain, *Proceedings of the Twenty-Ninth Annual International Conference on Advances in Cryptology*, pp. 286–302, 2009.

[13] M. Naor and O. Reingold, On the construction of pseudorandom permutations: Luby-Rackoff revisited, *Journal of Cryptology*, vol. 12(1), pp. 29–66, 1999.

[14] National Institute of Standards and Technology, Advanced Encryption Standard (AES), Federal Information Processing Standards Publication 197, Gaithersburg, Maryland, 2001.

[15] National Institute of Standards and Technology, Critical Infrastructure Protection, Gaithersburg, Maryland, 2002.

[16] National Institute of Standards and Technology, Cybersecurity Framework, Gaithersburg, Maryland, 2013.

[17] B. Obama, Improving critical infrastructure cybersecurity: Executive Order 13636, *Federal Register*, vol. 78(33), pp. 11739–11744, 2013.

[18] Offspark, `offspark`: Straightforward Security Communication, Rijswijk, The Netherlands, 2014.

[19] J. Patarin, Luby-Rackoff: Seven rounds are enough for $2^{n(1-\varepsilon)}$ security, *Proceedings of the Twenty-Third Annual International Conference on Advances in Cryptology*, pp. 513–529, 2003.

[20] J. Patarin, Security of random Feistel schemes with five or more rounds, *Proceedings of the Twenty-Fourth Annual International Conference on Advances in Cryptology*, pp. 106–122, 2004.

[21] Random.org, Random Binary File 2013-09-17, Dublin, Ireland (`www.random.org/files`), 2013.

[22] P. Rogaway, A Synopsis of Format-Preserving Encryption, Voltage Security, Cupertino, California, 2013.

[23] J. Soto, Randomness Testing of the AES Candidate Algorithms, National Institute of Standards and Technology, Gaithersburg, Maryland, 1999.

[24] T. Spies, Feistel Finite Set Encryption Mode, National Institute of Standards and Technology, Gaithersburg, Maryland, 2008.

[25] T. Spies, Format Preserving Encryption, Voltage Security, Cupertino, California, 2008.

[26] K. Stouffer, J. Falco and K. Scarfone, Guide to Industrial Control Systems (ICS) Security, NIST Special Publication 800-82, National Institute of Standards and Technology, Gaithersburg, Maryland, 2011.

[27] W. Trappe and L. Washington, *Introduction to Cryptography with Coding Theory*, Pearson Prentice Hall, Upper Saddle River, New Jersey, 2005.

[28] P. Tsang and S. Smith, Yasir: A low-latency, high-integrity security retrofit for legacy SCADA systems, *Proceedings of the IFIP TC-11 Twenty-Third International Information Security Conference*, pp. 445–459, 2008.

[29] J. Vance, VAES3 Scheme for FFX: An Addendum to the FFX Mode of Operation for Format Preserving Encryption, National Institute of Standards and Technology, Gaithersburg, Maryland, 2011.

[30] J. Walker, Ent: A Pseudorandom Number Sequence Test Program, Fourmilab, Lignieres, Switzerland (`www.fourmilab.ch/random`), 2008.

[31] Xilinx, Virtex-6 Family Overview, Xilinx Data Sheet, San Jose, California, 2011.