Evolution of Format Preserving Encryption on IoT Devices: FF1+

Alessandro N. Baccarini
Fordham Center for Cybersecurity
Fordham University, New York, NY
abaccarini@fordham.edu

Thaier Hayajneh
Fordham Center for Cybersecurity
Fordham University, New York, NY
thayajneh@fordhan.edu

Abstract

The Internet of Things (IoT) is a network of interconnected low-power sensing devices designed to interact and communicate with each other. To avoid compromising user privacy, it is necessary to encrypt these channels. We introduce Format Preserving Encryption (FPE), a modern cryptosystem that allows full customization of the ciphertext, while offering comparable security to AES. To gauge the performance of FPE, we compare the NIST-approved FF1 algorithm against several symmetric and asymmetric encryption schemes on a Raspberry Pi 3. While suitable for small plaintexts, FF1 breaks down for longer character strings. We propose a modified algorithm, FF1+*, that implements dynamic round selection and key scheduling. Significant performance improvements are observed in our results, thus demonstrating FF1+ as a viable cryptosystem for IoT devices.*

1. Introduction

The Internet of Things (IoT) is a hybrid network that consists of interconnected objects, services, humans, and devices that can communicate, share data, and information to reach a common goal in various areas and applications [1]. IoT is rapidly growing, reaching 23.14 billion devices as of 2018, and is projected to reach 75 billion uniquely identifiable devices by 2025 [2].

As the popularity of IoT has grown, determining ways of providing security and resilience to these networks has been a subject of a great deal of research (see [3, 4, 5, 6, 7, 8, 9, 10]). To ensure user privacy and prevent the types of attacks outlined in [8, 4], it is necessary to secure the channel of communication between IoT devices. Traditionally, this is accomplished by simply choosing an encryption scheme and implementing it on the devices. Existing methods of securing IoT include using classic algorithms that have been deemed secure, including AES-256, RSA, and elliptic curve cryptography (ECC) [5, 7, 11].

As simple as it would seem to use the most secure encryption algorithm available, the size and extraordinarily limited processing power of IoT devices are of primary concern. Symmetric algorithms (AES, Triple-DES (3DES), RC4, etc.) do not require significant computational power and offer adequate security [12, 13, 14], but require complete decryption before the data can be analyzed. Asymmetric algorithms (RSA, ECC, etc.) are the most secure means of encryption [15, 16], but are too computationally demanding to be considered viable for our purposes (as demonstrated in our results in Sec. 7), despite providing partial homomorphic capabilities.

We propose the use of Format Preserving Encryption (FPE), a technique that takes advantage of the lightness of symmetric algorithms, while providing unique features that allow for complete end-user customization. FPE allows us to tweak the ciphertext produced in a manner that can be used for analysis without needing to be decrypted, a feature not currently possible with classic symmetric algorithms.

We will be testing the NIST-recommended [17] FPE algorithm FF1 on a Raspberry Pi 3 to directly simulate encryption on an IoT device. We compare FF1 with several symmetric (AES, 3DES, RC4) and asymmetric (RSA with 2000- and 3000-bit keys) algorithms. Our programs are designed to record the time required to encrypt and decrypt an arbitrary plaintext ranging from 10 to 1000 characters. We demonstrate that, for small plaintext lengths, FF1 offers comparable performance to classic algorithms; however, it suffers severe breakdown for longer plaintexts (greater than 100 characters).

To resolve the aforementioned breakdown, we propose our modified FPE algorithm *FF1+*, an evolution of the FF1 algorithm designed specifically for IoT devices. It takes advantage of dynamic round selection according to plaintext size. Furthermore, the algorithm determines the necessary key size required to maximize performance while maintaining optimal security.

In Sec. 2, we briefly discuss the basics of IoT and current security principles. Sec. 3 contains a review

URI: https://hdl.handle.net/10125/59603
ISBN: 978-0-9981331-2-6

of cryptographic algorithms used to secure the IoT. We introduce FPE in Sec. 4 and the FF1 algorithm in Sec. 5. We justify our methodology for choosing this algorithm in Sec. 6. To test the performance of FF1, we run a series of comparisons against classic algorithms and discuss our results in Sec. 7. Our modified algorithm FF1+ is outlined in 8 alongside and additional testing and analysis. We propose future areas of research for this field and conclude our analysis in Sec. 9.

2. The IoT: An Overview

It is necessary to provide a general overview of how IoT networks are structured. An IoT network's architecture is divided into three layers: perception, network, and application [7, 3, 18]. Each layer in the architecture is vulnerable to various attacks (active or passive), and must be appropriately secured to protect user privacy.

Perception - Also commonly referred to as the sensing layer, the perception layer collects data from the area surrounding the devices. Technologies such as RFID, WSN, GPS, and NFC form the primary funcitonalites of the layer. If necessary, information processing occurs here prior to network transmission. For LANs, IoT node processing will also take place here [7].

Most attacks will occur at this layer, with vulnerabilities stemming from weak wireless signals, physical features (JTAG, UART, etc.), and the inherent dynamic nature of an IoT network [7, 19]. The perception layer can be exploited by replay attacks (identity information is spoofed, altered, or replayed to one or more devices on a network), timing attacks (an adversary recovers an encryption key by passively analyzing the time required to execute algorithms), and node capture attacks (a node is compromised and all it's information is leaked) [3].

Network - The network (transport) layer performs data transmission to the application layer through wired/wireless LANs. Technologies such as cloud computing, internet gateways, switching and routing operate using FTTx, 3G/4G, Wifi, Bluetooth, Zigbee, and UMB [1]. The layer can be divided into three sub layers: the access network (provides access to the perception layer), the core network (responsible for data transmission, typically the Internet), and the local area network (group of devices that share a wireless link) [4].

Denial of Service (DoS) attacks are very common in the network layer, as adversaries can flood the devices with packets or fake data. Additionally, man-in-the-middle attacks can occur at this layer by intercepting signals in between nodes [3].

Application - Lastly, the application layer functions differently depending on the use case scenario. Generally, it uses the processed data it received through the network layer [4]. Here, we can see implementations of IoT systems in smart homes, remote patient monitoring, logistics management, identity authentication, and so on [18]. As with other application technologies (e.g. web), this layer is susceptible to malicious code injection, DoS attacks, spear phishing, and sniffing [3].

3. IoT Cryptosystems

Having evaluated the current standard IoT architecture and types of attacks each layer is vulnerable to, we present an overview of the current state of encryption technologies. The primary defense against most attack vectors is implementing known secure cryptographic protocols. Different algorithms are implemented at the core of a cryptographic protocol to encrypt and decrypt plaintext and ciphertext, respectively. The two major subsets of the field are *symmetric* and *asymmetric* cryptography.

3.1. Symmetric Cryptography

A cryptosystem is said to be symmetric if the encryption and decryption algorithms use the same key. Under this category, there are two types of ciphers [13]. A *block cipher* is one in which a block of plaintext is treated as a whole and used to produce a ciphertext block of equal length. Typically, a block size of 64 or 128 bits is used.

The researchers in [20] define a *lightweight cipher* as a cryptographic algorithm specifically designed for low-resource devices that has minimal overhead, low-power consumption, and adequate security.

"Lightweight" implies small block size (32, 48, or 64 bit) compared to conventional ciphers (64 or 128 bit blocks) [21]. The key size tends to be smaller in the case of lightweight ciphers. Additionally, lightweight ciphers may simplify the key range [22] and employ elementary operations with a larger number of rounds [23].

A *stream cipher* is one that encrypts a digital data stream one bit or one byte at a time. The bit-stream generator must be implemented as an algorithmic procedure, so the cryptographic bit stream can be produced by both users. Classic examples of stream ciphers include RC4, Pike, Rabbit, and Py. The eSTREAM project of 2008 [24] produced several lightweight stream ciphers: Grain v1, MICKEY v2, and Tribium, which may be useful for IoT devices [13, 10].

3.2. Asymmetric Cryptography

Asymmetric Cryptography (or Public Key Cryptography) rely on using one key for encryption (*public*) and separate (*private*) key for decryption. Common examples of asymmetric algorithms include RSA, ECC, and Diffie-Hellman key exchange [5].

These cryptosystems provide unparalleled security compared to its symmetric counterparts. The major trade-off when using a Public Key System is the computational resources required to perform calculations [10]. The majority of IoT devices have limited available processing power. Poor algorithm performance may be infeasible for certain use cases (such as a network consisting of medical sensors) [23].

In an ideal scheme, we would use the most secure protocol available (such as RSA) in our cryptosystem. However, we must consider the extraordinarily limited processing power of IoT devices. The limiting factor of encryption on IoT devices is their extraordinarily limited processing power. Therefore, we must consider the following fundamental concerns for developing a cryptosystem for IoT devices: it is adequately secure from most common attacks, it does not require significant processing power to function properly, and it can be easily implemented in existing systems.

4. Format Preserving Encryption

Format preserving Encryption (FPE) refers to any encryption technique that takes a plaintext of a given format and produces a ciphertext in the same format. FPE allows the retrofitting of encryption technology to legacy applications; this is useful for implementing into a variety of IoT systems, such as those suggested in [25, 26].

Table 2 demonstrates an encryption of an IoT device's ID number followed by a string of data (such as temperature readings or heart rates). We see that using AES converts the plaintext into indiscernible ciphertext, while FPE produces ciphertext that can be easily used for analysis, while still being able to identify the source of the data.

Table 2. An example encryption of IoT data; the device's ID 1234 is preserved under encryption with FPE.

	Sample IoT Data
Plaintext	1234–1081–8254–9136–6918–7465
FPE	1234–3452–2341–7254–1029–8230
AES (base64)	0gV6F6/IvJrg2h/ePYo5xz0Pa lXB1HC5B4aawkLhwck=

For FPE to be considered a viable encryption technique, it must meet the following requirements:

1. The ciphertext is of the same length and format as the plaintext.
2. It should be adaptable to work with a variety of character and number types.
3. It should work with variable plaintext lengths.
4. Security strength should be comparable to that achieved with AES.
5. Security should be strong even for small plaintext lengths [27].

FPE algorithms implement *Feistel structures*, which are used to construct the block cipher. The structures are designed to convert any function (typically referred to as a F-function) into a permutation [28]. Feistel structures consists of *rounds* of reversible transformations consisting of three primary steps: splitting the data into two strings, applying a keyed function to one of the strings, and lastly reversing the roles of the strings for the subsequent round [17].

5. The FF1 Algorithm

Originally proposed to NIST under the name FFX [29], the FF1 algorithm is currently the only approved FPE algorithm. It supports the greatest range of lengths for the plaintext character string and the tweak. FF1 uses a pseudorandom function (PRF) that produces a 128-, 192-, or 256-bit output with inputted plaintext X that is a multiple of 128 bits and encryption key K [17]. PRF uses cipher block chaining with X as the plaintext input, an encryption key K, and an initial vector (IV) of all zeros [27]. FF1 uses a round function derived from AES-128 at its core with ten total rounds. Table 1 compares FF1 against several asymmetric and symmetric algorithms.

We define the base, or *radix*, as the number of characters in a given alphabet. By this definition, the radix of lowercase letters in the English alphabet is equal to 26. Including more characters into the alphabet (numbers, uppercase letters, symbols, etc.) will expand the radix.

FF1 receives two inputs: a numeral string X of length n and a tweak T of length t. A *numeral string* is defined as a finite, ordered sequence of numerals (numbers) for a given base. Essentially, we would convert characters into their numerical equivalent dictated by the base. As an example, the numerical

Table 1. Comparison of FF1 with classic encryption techniques [13].

Algorithm	Key Size (bits)	Block Size	Structure	No. Rounds
FF1	128/192/256	128	Feistel	10
AES	128/192/265	128	SPN	10/12/14
3DES	56/112/168	64	Feistel	48
RC4	40-2048	N/A	Feistel	1
RSA	2048/3072	214/342	Prime Numbers	1

representation of lowercase letters in the English alphabet is

$$a \to \texttt{0}, b \to \texttt{1}, \ldots, y \to \texttt{24}, z \to \texttt{25}. \tag{1}$$

Therefore, the character string `algorithm` would be represented by the numeral string `0 11 6 14 17 8 19 7 12`.

Our choice of radix is significant, since characters outside the alphabet will be undefined, and thus unable to be encrypted. Since we are simulating the transmission of data recorded by IoT devices (that would theoretically consist of pure numbers) and for simplicity, our alphabet ALPH will be composed of base ten numerals exclusively, such that

$$\begin{gathered} \text{ALPH} = \{0,1,2,3,4,5,6,7,8,9\}, \\ radix = 10. \end{gathered} \tag{2}$$

The FF1 encryption and decryption algorithms are outlined in detail at the end of this paper in Algs. 1 and 2, respectively (adapted from [30]). The algorithms are nearly identical aside from differences in Step 6 (indices are reversed, A and B are swapped, and modular subtraction is used instead of modular addition in Step 6.f).

An example encryption/decryption is shown in Table 3 using a 128-bit AES key and ALPH consisting of $\{0 \ldots 9, a, \ldots, z\}$ ($radix = 36$):

Table 3. Example FF1 Encryption.

Plaintext (X)	`0123456789abcdefghi`
Key (K)	`2B7E151628AED2A6` `↪ABF7158809CF4F3C`
Tweak (T)	`37373737707172733737 37`
Ciphertext (Y)	`a9tv40ml9kdu509eum`

During the ten rounds of FF1, the plaintext is broken into blocks, each of which is encrypted with AES-128,-192, or -256. Naturally, this provides improved security, but increased encryption time is unavoidable and may be observed in our results. The main "encryption" portion of the algorithm occurs during Step 6. Knowing that AES is used in the PRF and CIPH functions in Steps 6.b and 6.c, respectively, we can approximate the time order $\mathcal{O}$ required to perform a full FF1 encryption compared to a single AES encryption. The number of encryptions which are processed in Step 6.c are directly proportional to the plaintext length and chosen $radix$.

Our results are presented in Table 4 for a plaintext of length n ranging from $2 \leq n \leq 597$ with $radix = 10$. The general interval is approximately 77 characters before the number of AES encryptions increases (the slight variance is caused by the algorithm's floor/ceiling functions). The PRF in Step 6.b is called for every encryption, resulting in a minimum of 10 AES encryptions to always occur. Further ranges can be experimentally determined, or trivially extrapolated.

Examining the first range $2 \leq n \leq 56$, we are conducting 10 AES encryptions for a single FF1 encryption. Therefore, we expect for FF1 to at best be approximately 10 times slower than a single AES-128 encryption. For $57 \leq n \leq 134$, the CIPH function is triggered, leading to an additional 10 encryptions per round. This increases $\mathcal{O}$ to 20 for this interval, and will add 10 for every subsequent range.

Furthermore, choosing a larger $radix$ (such as all numerals and lowercase letters, or $radix = 36$), will cause $\mathcal{O}$ to increase and the ranges to shrink, leading to nontrivial performance degradations.

Table 4. $\mathcal{O}$ for FF1 encryptions with $radix = 10$, up to $n = 597$.

Range	$\mathcal{O}$ (No. of AES Encryptions)
$2 \leq n \leq 56$	10
$57 \leq n \leq 134$	20
$135 \leq n \leq 210$	30
$211 \leq n \leq 288$	40
$289 \leq n \leq 366$	50
$367 \leq n \leq 442$	60
$443 \leq n \leq 520$	70
$521 \leq n \leq 597$	80

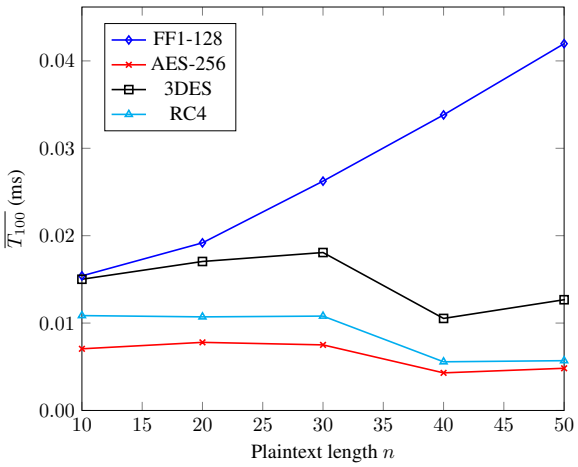

Figure 1. Preliminary testing results for $10 \leq n \leq 50$.

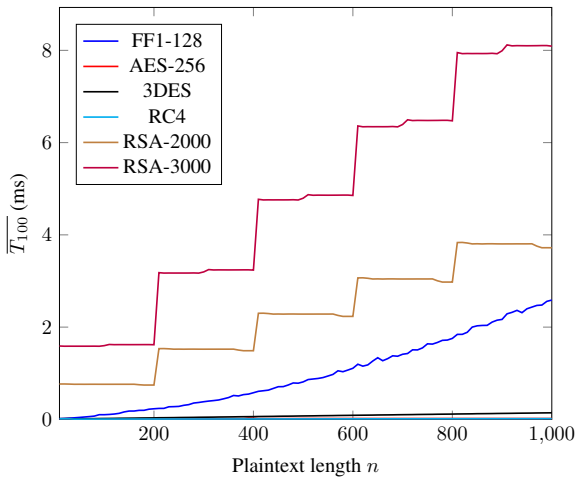

Figure 2. Preliminary testing results for $10 \leq n \leq 1000$.

6. Tweaks and Identity-based FPE

Our motivation for implementing FPE on IoT devices is as follows. The primary feature of FPE is the ability to implement tweaks into the algorithm, which allows further customization of how data is encrypted. An example tweak would preserve the first and last several digits of a plaintext which could be used for device identification, as shown in Table 2. The tweak need not be private information, which removes the need of adequately securing it on the device.

Additionally, we introduce the concept of Identity-based FPE, as proposed in [31, 32]. The scheme is used to reduce the risk of attacks due to key exposure. This is accomplished by associating the identity of the device to a derived key, which allows the device to encrypt a key K without needing to store it on the device. This additional layer of complexity drastically increases security in our cryptosystem by eliminating the possibility of compromising other IoT devices. Essentially, if a thermal sensor was attacked, only that device would be impacted; other devices in the IoT network would not be affected. The factory-set serial number can be used as the identifier for the encryption scheme.

We can sacrifice a reasonable amount of performance by using FF1 in order to gain the aforementioned features. The user has the freedom to adjust the parameters according to their specific needs. FPE reduces the amount of data transmitted (and thus the energy requirement), since the ciphertext is not limited to the cipher's block size (such as with AES).

7. Preliminary Testing

To determine if FPE is computationally feasible, we compare it with several classic symmetric encryption algorithms (AES-256, 3DES-168, and RC4-2048), as well as an asymmetric algorithms (RSA with 2000- and 3000-bit keys). We use FF1 with 128-, 192-, and 256-bit keys provided by NIST [17] (see Table 5). The programs are designed to run an encryption-decryption of a plaintext consisting purely of numbers, representative of the data transmitted from an IoT device. For our FF1 code, we used an open-source FPE implementation [33]. The remaining symmetric algorithms were written using OpenSSL. All programs were written in C.

Table 5. FF1 keys used in hex, as provided by NIST in [17].

Size (bits)	NIST Key
128	2B7E151628AED2A6ABF7158809CF4F3C
192	2B7E151628AED2A6ABF7158809CF4F3C ↪EF4359D8D580AA4F
256	2B7E151628AED2A6ABF7158809CF4F3C ↪2B7E151628AED2A6ABF7158809CF4F3C

Our preliminary test consists of running our encryption/decryption programs over a series of plaintext lengths ranging from 10 to 1000 in intervals of 10, of which each is run 100 times and averaged. This preliminary testing was performed on a Raspberry Pi 3 (CPU: Broadcom BCM2837 @ 1.2GHz; Memory: 1GB LPDDR2-900 SDRAM) board. It uses the Linux-based

operating system *Raspbain*, which means our programs will function on most IoT devices. The limited processing power of the board provides an accurate representation of how an IoT device performs when tasked with encrypting arbitrary data.

For FF1, an arbitrary tweak T alongside one of the NIST-recommended keys from Table 5 were used. An arbitrary key-IV pair (K, IV) for symmetric algorithms, as well as a public/private key pair for asymmetric algorithms. The data recorded were the plaintext length n and the average computation time for 100 iterations $\overline{T_{100}}$, measured in milliseconds. The programs encrypt and decrypt the generated plaintext, and write the length and average computation time $(n, \overline{T_{100}})$ to a file for analysis.

The results of our preliminary testing are presented in Figs. 1 ($10 \leq n \leq 50$) and 2 ($10 \leq n \leq 1000$). For the first range, we notice that FF1 performs reasonably well against the symmetric algorithms. Furthermore, it is well below RSA, which is omitted from this figure. However, our data demonstrates a quadratic regression for $n > 100$ and suffers severe breakdown. The symmetric algorithms are no longer discernible in comparison to FF1 and RSA-2000/3000.

While the symmetric algorithms maintain linear regression, FF1 begins to demonstrate quadratic regression followed by the equation

$$\overline{T_{100}} = 2 * 10^{-6} n^2 + 0.0006n + 0.012 \qquad (3)$$

with $R^2 = 0.9991$.

For RSA-2000/3000, if we were attempt to encrypt a plaintext of length n such that $210 < n < 400$, the encryption times would be doubled to approximately 1.52 ms for RSA-2000, and 3.17 ms for RSA-3000. This follows from asymmetric algorithms having constant encryption times up until their "maximum" n. Beyond the maximum, $\overline{T_{100}}$ would be doubled (until n passes two times the maximum n, then $\overline{T_{100}}$ would triple, and so on). We extrapolated the data accordingly for several ranges to demonstrate the performance of RSA for larger plaintext lengths in Fig. 2.

Fig. 3 directly compares FF1 with 128- 192-, and 256-bit keys. Aside from a few negligible time spikes, we observe no significant change in performance among the three. We will make use of this notion later.

The predictions we made in Sec. 5 were realized in our results. The time order for the encryption exceeded that predicted in Table 4. We conclude that FF1 performs well compared against symmetric and asymmetric schemes for plaintexts greater than $n = 50$. Beyond this point, FF1 struggles to compete with all symmetric algorithms. In its current state is not suitable for use on IoT devices using any protocol with relatively large maximum transmission units (MTUs). Optimization of the FF1 C code is possible, but we are still limited by the nature of FF1's internal AES calculations (as demonstrated in Sec. 5).

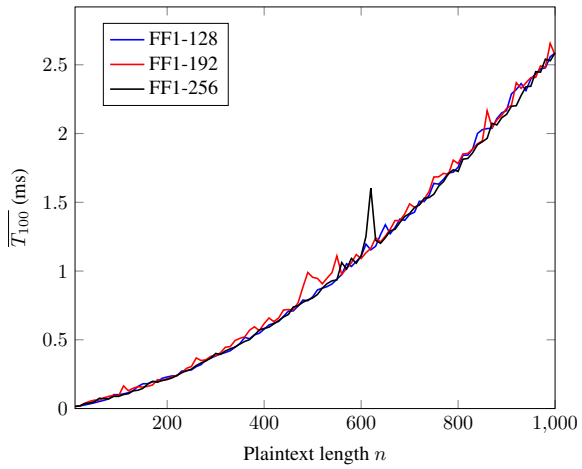

Figure 3. Plot comparing FF1 using 128-, 192-, and 256-bit keys.

8. FF1+: Dynamic Round Selection

We determined in Sec. 7 that FF1 provides comparable performance to traditional encryption methods for relatively small plaintext sizes, but suffers for $n > 50$. In order for FF1 to be considered a viable encryption scheme among existing algorithms, it must be modified. We propose *FF1+*, an evolution of the FF1 algorithm for primary use on IoT devices.

The main modification we propose is the reduction of round numbers. NIST proposes the use of 10 internal rounds (as shown in Step 6 of Alg. 1). In [29], the suggested number of rounds for FFX (which evolved into FF1) depends on the $\texttt{split}(n)$, a function that takes an allowed length n and returns a number such that $1 \leq \texttt{split}(n) \leq n/2$. They require the number of rounds $r \leq 8$ for $n = 2 \cdot \texttt{split}(n) + 1$ in order to prevent known attacks. However, this recommendation was made under the assumption that relatively small plaintexts will be used.

In one of the examples presented in Appendix B of [29], 12 rounds are specified as the minimum for a plaintext ranging from $10 \leq n \leq 36$ with $radix = 10$. Furthermore, the NIST specification of FF1 [17] concludes that ten rounds is sufficient for all n. Expanding the range of allowable plaintext lengths allows us to simultaneously reduce the number of rounds.

Table 6. Wireless IoT protocols' average data transfer dates from [30]; the third column contains the maximum transmission unit (MTU) for each protocol in bytes, which dictates their class.

Class	Protocol	Data rate	MTU (bytes)	No. FF1+ Rounds	Key Size (bits)
1	Ant	12.8 kbps	8	**10**	**128**
	Weightless-P	200 bps	10		
	Weightless-W	1 kbps	10		
	Sigfox	1000 bps	12		
	BLE	8 kbps	20		
	Weightless-N	30 kbps	20		
2	UHF RFID	40 kbps	50	**8**	**128**
	LoRaWAN 1.0	20 kbps	51		
3	6LoWPAN	250 kbps	127	**8**	**192**
	Zigbee 3.0	250 kbps	128		
	Z-Wave ZAD12837	40 kbps	128		
4	Bluetooth 4.2	1 Mbps	251	**6**	**192**
	NFC ISO	100 kbps	254		
5	Thread	250 kbps	1280	**6**	**256**
	WiFi 802.11n	200 Mbps	1500		
	NB-IoT	250 kbps	1500		

From our testing, we determined there to be no added benefit requiring $r = 10$ for $n > 100$. Therefore, concerning ourselves with large plaintexts allows us reduce the number of total rounds for the encryption and decryption functions accordingly. This will result in improved performance without sacrificing security.

To illustrate this point, Table 7 contains a sample encryption of a small plaintext and the changes to the ciphertext when adjusting the number of rounds. Each ciphertext does not have any discernible relationship to the others. Furthermore, the differences between large ciphertexts using different numbers of rounds are more drastic. This approach of dynamic round selection will allow us to improve performance and maintain security across a variety of plaintext lengths.

Table 7. Ciphertext comparison between encrypting a plaintext using 10, 8, and 6 rounds.

Plaintext	`0123456789`
Ciphertext (10)	`2433477484`
Ciphertext (8)	`8392433335`
Ciphertext (6)	`4436484765`

Table 6 contains the data rates and maximum transmission units (MTU) for several IoT protocols, which are divided into different classes according to their respective MTUs. Each class corresponds to how many rounds will be used for FF1+. Class 1 contains protocols with small MTUs, so we maintain the original recommended number of rounds and key size. All other Classes vary the number of rounds according to their respective MTUs.

As noted in Sec. 7, the key size for FF1 does not impact performance. As such, we require a 128-bit key for Classes 1 and 2, 192-bit for Classes 3 and 4, and 256-bit Class 5. This decision ensures we maintain overall security while maximizing algorithm performance.

Our results of testing FF1+ is presented in Table 8. We compare FF1 and FF1+ for each Class under the same conditions described in Sec. 7. The parameters are unchanged for Class 1, resulting in the same encryption times. For all subsequent classes, we observe performance improvements up to 39%. We conclude that dynamic round selection in FF1+ makes FPE a viable option for the IoT.

Table 8. Our results comparing FF1 and FF1+ encryption times, and the respective performance increase percentages.

Class	FF1 (ms)	FF1+ (ms)	% Increase
1	0.01907	0.01907	0
2	0.04197	0.03257	22
3	0.13065	0.09525	27
4	0.29432	0.17753	39
5	4.04496	2.54526	37

Our comparison of FF1+ against classic encryption techniques is presented in Fig. 4. The marked points at $n = \{50, 250\}$ for FF1+ indicate when the algorithm switches the number of rounds, such that for $n = 50$, the rounds are reduced from 10 to 8, and for $n = 250$, they

are reduced from 8 to 6. Similar to FF1, the plaintext intervals of FF1+ obey quadratic regression. While still not efficient as efficient as classical symmetric algorithms for larger plaintext lengths, we do observe a noticeable improvement with FF1+ over FF1, as well as maintaining better performance than asymmetric algorithms. We conclude that FF1+ will consistently perform better than FF1 for $n > 50$, but will still not be as efficient as classic symmetric algorithms for large plaintexts.

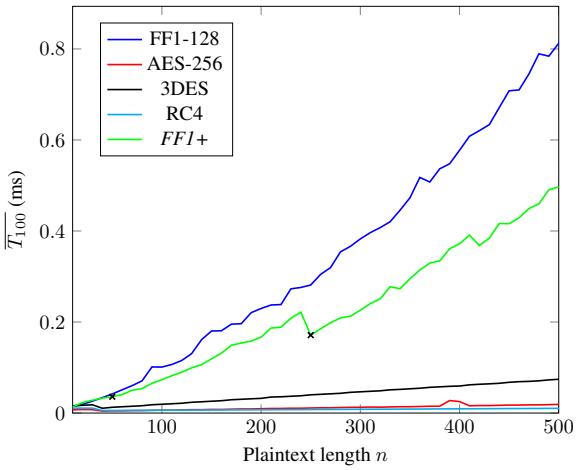

Figure 4. FF1+ testing for $10 \leq n \leq 500$. Asymmetric algorithms are neglected as they maintain significantly slower performance.

9. Conclusion

We introduced the basics of FPE and its inherent value to IoT security. We demonstrated that, in its current form, FF1 suffers suffers in performance for semi-large plaintext lengths for IoT devices. Formal implementation and optimization of the algorithm may improve performance, but FF1 will always limited by its core encryption structure. Our modified algorithm FF1+ successfully resolves this issue by implementing dynamic round selection according to plaintext length. Protocols are divided into classes according to their MTU and are assigned a number of rounds and required key size.

An additional modification we plan to implement is swapping AES for HIGHT as the internal cipher function. HIGHT is a lightweight block cipher developed for low-resource devices [34] . It uses a 64-bit block length and 128-bit key length, and has been shown in [35] to perform more efficiently and faster than AES. The comparison between FF1 and our planned modifications to FF1+ are shown in Table 9. This can further improve overall algorithm performance, while simultaneously reducing computational requirements.

Table 9. Future modifications to FF1+ compared to FF1.

Algorithm	FF1	FF1+
Block size (bits)	128	64
Number of rounds	10	6-10
Key size	128	128/192/256
CIPH function	AES-128	HIGHT

To confirm that FF1+ maintains security, we will conduct various forms of cryptanalysis (known plaintext attacks, ciphertext-only attacks, differential cryptanalysis, integral cryptanalysis, side-channel attacks meet-in-the-middle attacks, and dictionary attacks [36]) against our algorithm, as well as examine the strong and weak avalanche effects [37]. Testing will also be conducted on a field-programmable gate array (FPGA) board. Limited research of FF1 on a FPGA exists [38], and we will expand upon this by testing FF1+ and compare it with other classic encryption techniques.

Since FPE is designed to maintain the format of the plaintext under encryption, we may be able to resolve homomorphic properties from FF1. At the time of writing, there is no existing research in this area. This would provide yet another valuable feature for IoT devices that currently exists only for asymmetric algorithms such as RSA, ElGamal, and Pallier (all partial homomorphic) [39]; this category of algorithms all require significantly more processing power than FF1.

Algorithm 1 FF1 Encryption

Prerequisites:

- A designated cipher function CIPH of an approved block cipher (128-, 192-, or 256-bit),
- A Key K for the block cipher,
- The chosen base (alphabet), $radix$,
- The range of supported message lengths (n) from $2 \leq n \leq 2^{32}$,
- The maximum byte length for tweaks $maxTlen$.

Functions:

- $\text{PRF}(X)$ - the output of a pseudorandom function applied to block X.
- $\text{STR}^m_{radix}(x)$ - conversion of a nonnegative integer x into a string of m numerals in base $radix$.
- $\text{NUM}(X)$ - the integer that a bit string X.
- $\text{NUM}_{radix}(X)$ - the number that a numeral string X represents in base $radix$.

Inputs:

- A numeral string X of length n, where $X \in radix$,
- The tweak T as a byte string of length t, where $0 \leq t \leq maxTlen$.

Output:

- A numeral string Y of length n.

Steps:

1: Let $u \equiv \lfloor \frac{n}{2} \rfloor$ and $v \equiv n - u$.
2: Set the first and second halves of X to numeral strings A and B, such that $A = X[1..u], B = X[u+1..n]$.
3: Let $b = \lceil \lceil v \log_2(radix) \rceil / 8 \rceil$.
4: Let $d = 4\lceil b/4 \rceil + 4$
5: Let $P = [1]^1 \| [2]^1 \| [1]^1 \| [radix]^3 \| [10]^1 \| [u \bmod 256]^1 \| [n]^4 \| [t]^4$.
6: **for** $i = 0$ to 9
  a: Let $Q = T \| [0]^{(-t-b-1) \bmod 16} \| [i]^1 \| [\text{NUM}_{radix}(B)]^b$.
  b: Let $R = \text{PRF}(P \| Q)$.
  c: Set S to the first d bytes of the string of $\lceil d/16 \rceil$: $R \| \text{CIPH}_K(R \oplus [1]^{16}) \| \ldots \| \text{CIPH}_K(R \oplus [\lceil d/16 \rceil - 1]^{16})$.
  d: Let $y = \text{NUM}(S)$
  e: **if** i is even, let $m = u$; **else**, let $m = v$.
  f: Let $c = (\text{NUM}_{radix}(A) + y) \bmod (radix^m)$.
  g: Let $C = \text{STR}^m_{radix}(c)$
  h: Set $A = B$, then $B = C$.
7: Return $Y = A \| B$.

Algorithm 2 FF1 Decryption

Note: The *Prerequisites*, *Functions*, *Inputs*, and *Output* are the same for the Decryption algorithm.

Steps:

1: Let $u \equiv \lfloor \frac{n}{2} \rfloor$ and $v \equiv n - u$.
2: Set the first and second halves of X to numeral strings A and B, such that $A = X[1..u], B = X[u+1..n]$.
3: Let $b = \lceil \lceil v \log_2(radix) \rceil / 8 \rceil$.
4: Let $d = 4\lceil b/4 \rceil + 4$
5: Let $P = [1]^1 \| [2]^1 \| [1]^1 \| [radix]^3 \| [10]^1 \| [u \bmod 256]^1 \| [n]^4 \| [t]^4$.
6: **for** $i = 9$ to 0
  a: Let $Q = T \| [0]^{(-t-b-1) \bmod 16} \| [i]^1 \| [\text{NUM}_{radix}(A)]^b$.
  b: Let $R = \text{PRF}(P \| Q)$.
  c: Set S to the first d bytes of the string of $\lceil d/16 \rceil$: $R \| \text{CIPH}_K(R \oplus [1]^{16}) \| \ldots \| \text{CIPH}_K(R \oplus [\lceil d/16 \rceil - 1]^{16})$.
  d: Let $y = \text{NUM}(S)$
  e: **if** i is even, let $m = u$; **else**, let $m = v$.
  f: Let $c = (\text{NUM}_{radix}(B) - y) \bmod (radix^m)$.
  g: Let $C = \text{STR}^m_{radix}(c)$
  h: Set $B = A$, then $A = C$.
7: Return $Y = A \| B$.

References

[1] R. Mahmoud, T. Yousuf, F. Aloul, and I. Zualkernan, "Internet of things (iot) security: Current status, challenges and prospective measures," in *Internet Technology and Secured Transactions (ICITST), 2015 10th International Conference for*, pp. 336–341, IEEE, 2015.

[2] statista, "Internet of things (iot) connected devices installed base worldwide from 2015 to 2025 (in billions)," may 2018.

[3] M. U. Farooq, M. Waseem, A. Khairi, and S. Mazhar, "A critical analysis on the security concerns of internet of things (iot)," *International Journal of Computer Applications*, vol. 111, no. 7, 2015.

[4] Q. Jing, A. V. Vasilakos, J. Wan, J. Lu, and D. Qiu, "Security of the internet of things: perspectives and challenges," *Wireless Networks*, vol. 20, no. 8, pp. 2481–2501, 2014.

[5] U. Kumar, T. Borgohain, and S. Sanyal, "Comparative analysis of cryptography library in iot," *arXiv preprint arXiv:1504.04306*, 2015.

[6] T. Xu, J. B. Wendt, and M. Potkonjak, "Security of iot systems: Design challenges and opportunities," in *Proceedings of the 2014 IEEE/ACM International Conference on Computer-Aided Design*, pp. 417–423, IEEE Press, 2014.

[7] H. Suo, J. Wan, C. Zou, and J. Liu, "Security in the internet of things: a review," in *Computer Science and Electronics Engineering (ICCSEE), 2012 international conference on*, vol. 3, pp. 648–651, IEEE, 2012.

[8] K. T. Nguyen, M. Laurent, and N. Oualha, "Survey on secure communication protocols for the internet of things," *Ad Hoc Networks*, vol. 32, pp. 17–31, 2015.

[9] J. Wurm, K. Hoang, O. Arias, A.-R. Sadeghi, and Y. Jin, "Security analysis on consumer and industrial iot devices," in *Design Automation Conference (ASP-DAC), 2016 21st Asia and South Pacific*, pp. 519–524, IEEE, 2016.

[10] M. Katagi and S. Moriai, "Lightweight cryptography for the internet of things," *Sony Corporation*, pp. 7–10, 2008.

[11] N. Sklavos and I. D. Zaharakis, "Cryptography and security in internet of things (iots): Models, schemes, and implementations," in *New Technologies, Mobility and Security (NTMS), 2016 8th IFIP International Conference on*, pp. 1–2, IEEE, 2016.

[12] A. K. Lenstra, "Unbelievable security matching aes security using public key systems," in *International Conference on the Theory and Application of Cryptology and Information Security*, pp. 67–86, Springer, 2001.

[13] B. J. Mohd, T. Hayajneh, and A. V. Vasilakos, "A survey on lightweight block ciphers for low-resource devices: Comparative study and open issues," *Journal of Network and Computer Applications*, vol. 58, pp. 73–93, 2015.

[14] G. Singh, "A study of encryption algorithms (rsa, des, 3des and aes) for information security," *International Journal of Computer Applications*, vol. 67, no. 19, 2013.

[15] A. S. Wander, N. Gura, H. Eberle, V. Gupta, and S. C. Shantz, "Energy analysis of public-key cryptography for wireless sensor networks," in *Pervasive Computing and Communications, 2005. PerCom 2005. Third IEEE International Conference on*, pp. 324–328, IEEE, 2005.

[16] G. Gaubatz, J.-P. Kaps, and B. Sunar, "Public key cryptography in sensor networks—revisited," in *European Workshop on Security in Ad-Hoc and Sensor Networks*, pp. 2–18, Springer, 2004.

[17] M. Dworkin, "Recommendation for block cipher modes of operation: methods for formatpreserving encryption," *NIST Special Publication*, vol. 800, p. 38G, 2016.

[18] S. Li, L. Da Xu, and S. Zhao, "The internet of things: a survey," *Information Systems Frontiers*, vol. 17, no. 2, pp. 243–259, 2015.

[19] M. R. Abdmeziem, D. Tandjaoui, and I. Romdhani, "Architecting the internet of things: state of the art," in *Robots and Sensor Clouds*, pp. 55–75, Springer, 2016.

[20] X. Fan, K. Mandal, and G. Gong, "Wg-8: A lightweight stream cipher for resource-constrained smart devices," in *International Conference on Heterogeneous Networking for Quality, Reliability, Security and Robustness*, pp. 617–632, Springer, 2013.

[21] A. Bogdanov, L. R. Knudsen, G. Leander, C. Paar, A. Poschmann, M. J. Robshaw, Y. Seurin, and C. Vikkelsoe, "Present: An ultra-lightweight block cipher," in *International Workshop on Cryptographic Hardware and Embedded Systems*, pp. 450–466, Springer, 2007.

[22] M. Cazorla, K. Marquet, and M. Minier, "Survey and benchmark of lightweight block ciphers for wireless sensor networks," in *Security and Cryptography (SECRYPT), 2013 International Conference on*, pp. 1–6, IEEE, 2013.

[23] N. Alassaf, B. Alkazemi, and A. Gutub, "Applicable light-weight cryptography to secure medical data in iot systems," *Arabia*, 2003.

[24] S. Babbage, C. Canniere, A. Canteaut, C. Cid, H. Gilbert, T. Johansson, M. Parker, B. Preneel, V. Rijmen, and M. Robshaw, "The estream portfolio," *eSTREAM, ECRYPT Stream Cipher Project*, 2008.

[25] A. G. Logan, W. J. McIsaac, A. Tisler, M. J. Irvine, A. Saunders, A. Dunai, C. A. Rizo, D. S. Feig, M. Hamill, M. Trudel, *et al.*, "Mobile phone–based remote patient monitoring system for management of hypertension in diabetic patients," *American journal of hypertension*, vol. 20, no. 9, pp. 942–948, 2007.

[26] H. Fernandez-Lopez, J. A. Afonso, J. H. Correia, and R. Simoes, "Remote patient monitoring based on zigbee: lessons from a real-world deployment," *Telemedicine and e-Health*, vol. 20, no. 1, pp. 47–54, 2014.

[27] S. William, *Cryptography and network security: principles and practices*, vol. 232. Pearson Education, 2006.

[28] B. Schneier and J. Kelsey, "Unbalanced feistel networks and block cipher design," in *International Workshop on Fast Software Encryption*, pp. 121–144, Springer, 1996.

[29] M. Bellare, P. Rogaway, and T. Spies, "The ffx mode of operation for format-preserving encryption," *NIST submission*, vol. 20, 2010.

[30] R. Components, "11 internet of things (iot) protocols you need to know about," aug 2015.

[31] M. Bellare and V. T. Hoang, "Identity-based format-preserving encryption," *CCS 2017*, 2017.

[32] B. Latré, B. Braem, I. Moerman, C. Blondia, and P. Demeester, "A survey on wireless body area networks," *Wireless Networks*, vol. 17, no. 1, pp. 1–18, 2011.

[33] 0NG, "Fpe - format preserving encryption implementation in c." https://github.com/0NG/Format-Preserving-Encryption, 2018.

[34] D. Hong, J. Sung, S. Hong, J. Lim, S. Lee, B.-S. Koo, C. Lee, D. Chang, J. Lee, K. Jeong, *et al.*, "Hight: A new block cipher suitable for low-resource device," in *International Workshop on Cryptographic Hardware and Embedded Systems*, pp. 46–59, Springer, 2006.

[35] B. J. Mohd, T. Hayajneh, Z. A. Khalaf, A. Yousef, and K. Mustafa, "Modeling and optimization of the lightweight hight block cipher design with fpga implementation," *Security and Communication Networks*, vol. 9, no. 13, pp. 2200–2216, 2016.

[36] C. Swenson, *Modern cryptanalysis: techniques for advanced code breaking*. John Wiley & Sons, 2008.

[37] S. Ramanujam and M. Karuppiah, "Designing an algorithm with high avalanche effect," *IJCSNS International Journal of Computer Science and Network Security*, vol. 11, no. 1, pp. 106–111, 2011.

[38] R. Agbeyibor, J. Butts, M. Grimaila, and R. Mills, "Evaluation of format-preserving encryption algorithms for critical infrastructure protection," in *International Conference on Critical Infrastructure Protection*, pp. 245–261, Springer, 2014.

[39] L. Morris, "Analysis of partially and fully homomorphic encryption," *Rochester Institute of Technology*, pp. 1–5, 2013.