FlexAEAD v1.2 -A Lightweight AEAD Cipher with Integrated Authentication

Eduardo Marsola do Nascimento[1], José Antônio Moreira Xexéo[2]
[1] Petróleo Brasileiro S.A. – Petrobras
Rio de Janeiro, RJ
[2]Instituto Militar de Engenharia
Rio de Janeiro, RJ

edunasci@yahoo.com, xexeo@ime.eb.br

Abstract. *This paper[1] describes a symmetrical block cipher family – FlexAEAD v1.2. It was engineered to be lightweight, consuming less computational resources than other ciphers and to work with different block and key sizes. Other important characteristic is to integrate the authentication on its basic algorithm. This approach is helps to reduce the resource needs. The algorithm capacity to resist against linear and different cryptanalysis attacks was evaluated. The FlexAEAD also supports the authentication of the Associated Data (AD). The version 1.2 improves the cipher performance by using a multiply with carrier pseudo random generator as a counter to make each block unique.*

Keywords: authenticated encryption, lightweight, NIST LWC.

1. Introduction

On august 2018, the National Institute of Standards and Technology (NIST) published call for algorithm (NIST, 2018) describing the contest and requirements for a new lightweight authenticated encryption with associated data (AEAD) algorithm and an optional hash algorithm.

The FleaxAEAD algorithm family was inscribed in the contest and analyzed by several researchers. The cipher family is an evolution of the FlexAE algorithm presented at IEEE ICC2017 (Paris – France) and SBSEG2018 (Natal – Brazil). The first difference is the capacity to allow the validation of an associated data together with the encrypted data. The new family also resolved a reorder block attack.

During NIST contest first round, independent researchers found a weakness related to the associated data padding and an iterated differential attack. The weakness were solved and resulted on the cipher version 1.1.

The version 1.2 improves the cipher performance by using a multiply with carrier pseudo random generator as a counter to make the data blocks unique.

This specification and security claims for the cipher variations were revised and they are presented on this paper. The cipher source code is available on the URL https://github.com/edunasci/FlexAEAD.

2. Algorithm Description

The FleaxAEAD algorithm uses as a main component a key dependable permutation

[1] This is an upgrade of the specification already published at https://doi.org/10.17648/jisc.v6i1.74.

function (PF_K). On this function, the block is XORed with a key K_A at the beginning and with a key K_B at the end of the process. This function (PF_K) is invertible ($INVPF_K$), so the process can be reversed ([2]).

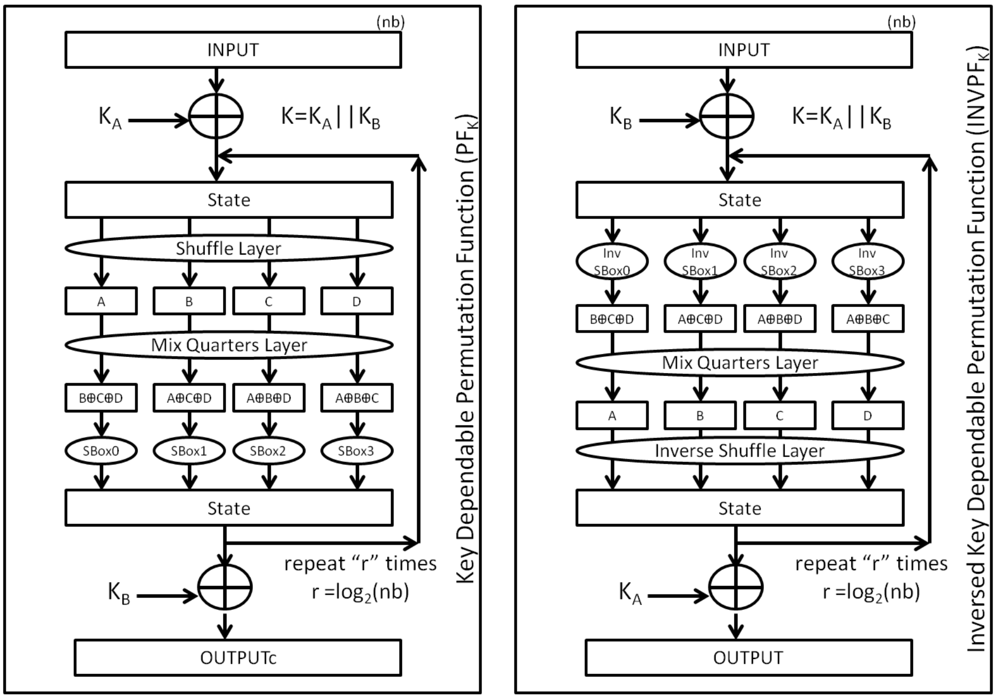

Figure 1. The permutation function PF_K and its inverse $INVPF_K$

On the (PF_K), after the XOR with K_A, the block is transformed by a shuffle layer, where a *nb* bytes input is reordered as $(\mathrm{b}[0], \mathrm{b}\left[\frac{\mathrm{nb}}{2}\right], \mathrm{b}[1], \mathrm{b}\left[\frac{\mathrm{nb}}{2}+1\right], \ldots, \mathrm{b}\left[\frac{\mathrm{nb}}{2}-1\right],$ $\mathrm{b}[\mathrm{nb}-1]$.

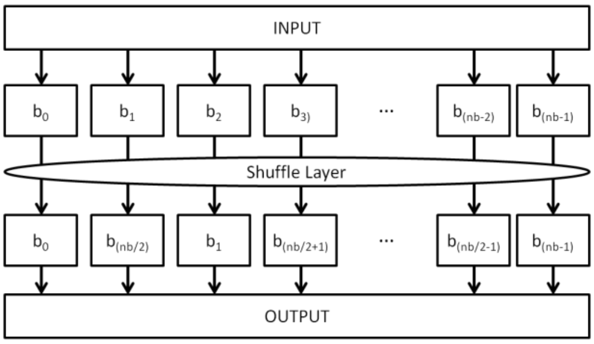

Figure 2. The Shuffle Layer

After the shuffle Layer, the input is divided in quarters and the mix quarters layers combine them together. Considering the quarters (A, B, C, D) as input, the output will be $(B \oplus C \oplus D, A \oplus C \oplus D, A \oplus B \oplus D, A \oplus B \oplus C)$. The function is its own inverse,

[2] This function were rewritten to avoid an efficient iterated truncated differential attacks proposed by Mostafizar Rahman, Dhiman Saha and Goutam Paul during the contest discussions.

if the output is submitted again to the function, it will generate the original input. A difference on one byte will generate differences in 3 bytes, in different quarters.

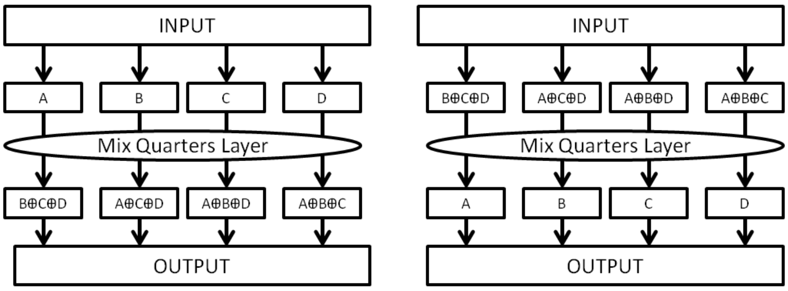

Figure 3. Mix Quarters Layer

The next is the SBox layer, where each quarter suffers a non-linear transformation using a different SBox. The first SBox is the AES SBox, the other SBoxes are generate using the process as the first (multiplicative inverse on the $GF2^8$) using different irreducible polynomial (IP), multiplicative constant (MC) and additive constants (A).

Table 1. Parameters used to create FlexAEAD SBoxes

SBox	IP	MC	AC
SBox0	$x^8 + x^4 + x^3 + x^1 + 1$ (0b100011011)	0x1F	0x63
SBox1	$x^8 + x^4 + x^3 + x^2 + 1$ (0b100011101)	0x3D	0x95
SBox2	$x^8 + x^5 + x^3 + x^1 + 1$ (0b100101011)	0x3B	0xA6
SBox3	$x^8 + x^5 + x^3 + x^2 + 1$ (0b100101101)	0x37	0xD9

The SBox Layer can be inverted using the reverse AES SBox. On the appendices the SBoxes direct and reverse tables can be found.

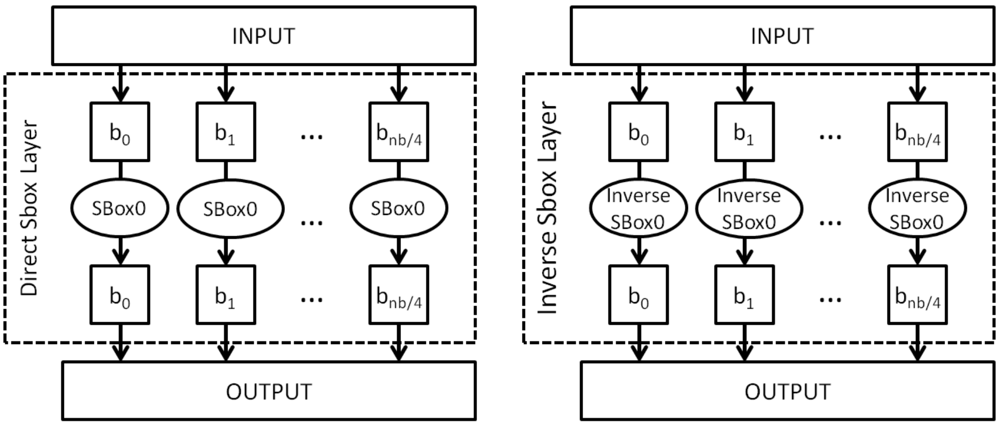

Figure 4. The SBox Layer

The number of rounds (r) on this construction is $r = \log_2 nb$, where nb=block size in bytes. This number of rounds is the minimum to assure that any bit change on the input the block will affect all bits on the output. The number of rounds grows logarithmic with the block size, keeping the number of cpu cycles needed to process small even if working with bigger block sizes.

The key dependable permutation function and its inverse can also be described on the pseudo code on the

Figure 5.

```
dirPF(INPUT[nb], K[nk])

K_A = K[1..(nk/2)]
K_B = K[(nk/2 + 1)..nk]
state = INPUT ⊕ K_A
for ( i =1 to log(nb)]
    state = ShuffleLayer(state)
    state = MixQuartersLayer(state)
    state = dirSBoxLayer(state)
end for
OUTPUT = state ⊕ K_B
```

```
invPF(INPUT[nb], K[nk])

K_A = K[1..(nk/2)]
K_B = K[(nk/2 + 1)..nk]
state = INPUT ⊕ K_B
for ( i =1 to log(nb)]
    state = MixQuartersLayer(state)
    state = invShuffleLayer(state)
    state = invSBoxLayer(state)
end for
OUTPUT = state ⊕ K_B
```

Figure 5. The key dependable permutation function and its inverse

The FlexAEAD v1.2 cipher uses three subkeys (K_0, K_1, K_2. They are created from a bit sequence generated by applying the permutation function three times using the main key K (PF_K) until have enough bits for all subkeys. The initial value is a sequence of zeros ($0^{ks/2}$). Each subkey (K_0, K_1, K_2) size is $2 \times \mathrm{nb}$, which is double the block size in bytes (or $16 \times nb$ in bits). The main key K size is $128 \times 2^{\mathrm{x}}$ bits, where $\mathrm{x} \geq 0$. The maximum size of the main key is two times the blocksize. This limit was imposed to force each subkey to be composed by a sequence that went by the process at least twice. The number of times the permutation function is applied has been chosen to have the similar resistance to linear and differential cryptanalysis attacks on the subkey generation as on encrypting a block.

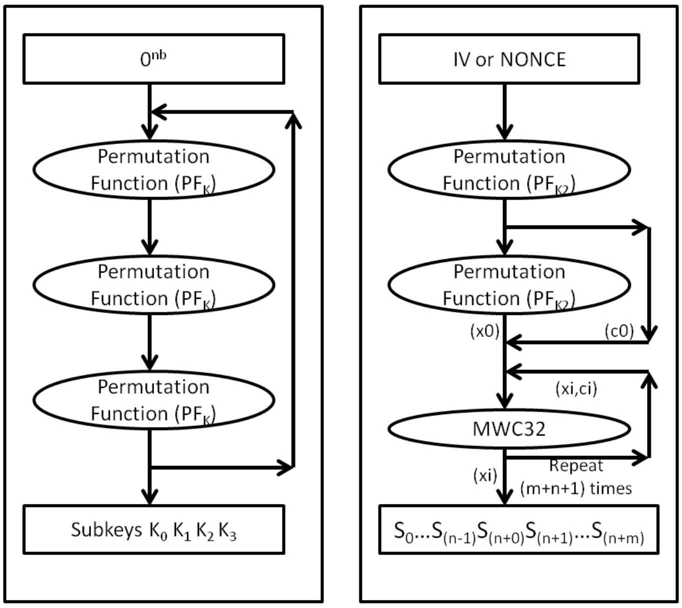

Figure 6. The K_0,K_1,K_2 and $S_0S_1\ldots S_{m+n}$ generation processes

The FlexAEAD also uses a sequence of bits ($S_0 S_1 \ldots S_{n+m}$). This sequence is the same size of the associated data plus the message to be sent. The sequence will be unique for every NONCE. It is generate by applying PF_{K2} two times over the NONCE to generate a base counter. The counter is composed by the two blocks (x,c). Each block of the counter is divided in 32 bits chunks of data. Each individual 32 bits chunk (x_0, c_0) is treated as an unsigned number (little-endian) that is applied to a 32 bits multiply with carrier (MWC32) pseudo random generator. The MWC32 function is define by $x_{i+1} = (a \times x_i + c_i) \wedge (2^{32} - 1); c_{i+1} = \left\lfloor \frac{a \times x_i + c_i}{2^{32}} \right\rfloor ; where\ a = 4294967220$. The maximum period of the counter is $\approx 2^{63}$, except if $x_0 = 0\ and\ c_0 = 0$, when the period is =1, causing the algorithm to fail. Even considering that the probability for this condition is very small ($p = 2^{-64}$), it was considered on the design. To avoid it, if $c_0 = 0$, it is replace by $c_0 = 0x1111111$.

To hash the associate data, first the associated data is divided in n blocks $(AD_0 AD_1 \ldots AD_{n-1})$. The final block is padded with 10…0 bits ([3]). Each block (AD_x) is submitted to PF_{K1}, than it is XORed with the correspondent (S_x) block, than submitted to direct SBox Layer to generate a intermediate state block (st_x). The process that each associated data block goes though is $(AD_x \rightarrow PF_{K1} \rightarrow XOR(s_x) \rightarrow dirSBox \rightarrow st_x)$. If the last block has been padded, the function PF_{K1} is applied twice: $(AD_{x(padded)} \rightarrow PF_{K1} \rightarrow PF_{K1} \rightarrow XOR(s_x) \rightarrow dirSBox \rightarrow st_x)$.

The (S_n) block is used as an intermediate state block $(S_n \rightarrow st_n)$. This operation was included[4] to avoid having the same tag, for different NONCEs, when both AD and M are empty. Another reason is to avoid having the same tag for $(N, A_{0\ldots n-1}||P_0, P_{1\ldots m-1})$ and $(N, A_{0\ldots n-1}, P_{0\ldots m-1})$.

To cipher the plain text message, it is broken into m plaintext blocks $(P_0 P_1 \ldots P_{m-1})$. The last block is padded with (10^{pb-1}), where pb is the number of padding bits to complete the block.

[3] The original cipher permitted the forgery extended length attack. The actual version solved the problem by using a resistant padding as suggested by by Alexandre Mège.

[4] Both problems where pointed by Maria Eichlseder on NIST LWC discussion forum.

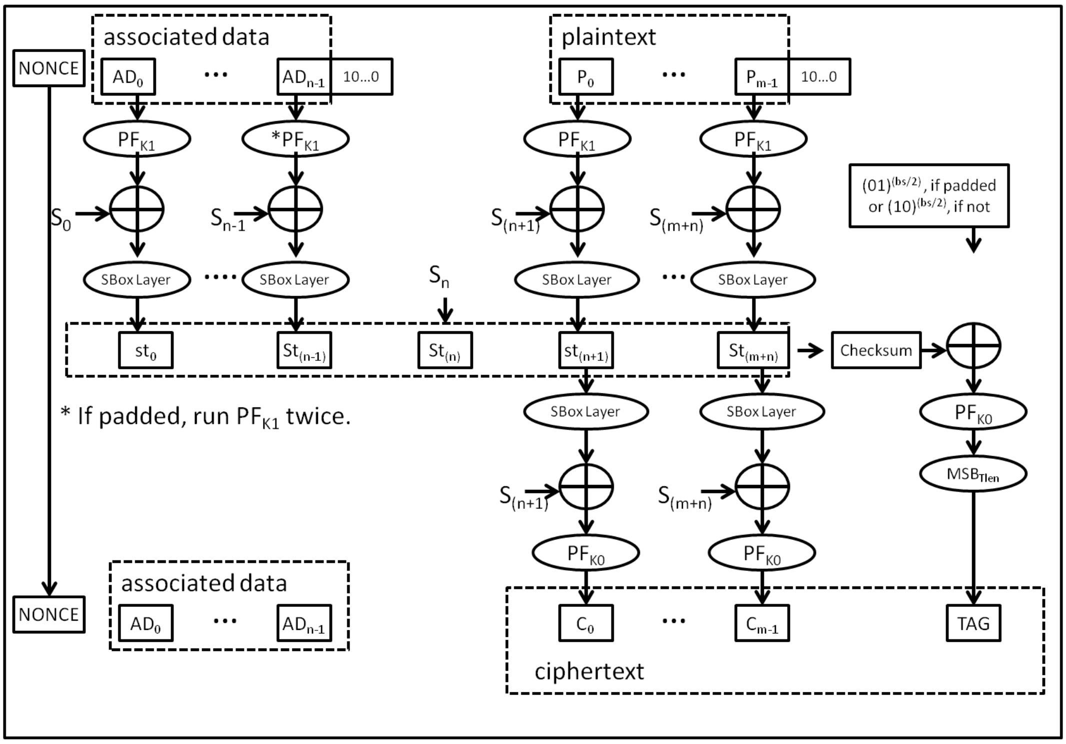

Figure 7. The FlexAEAD v1.2 encryption diagram

Each block (P_x) is submitted to PF_{K1}, it is XORed with the correspondent (S_x) block and submitted to the SBox Laye, to generate a intermediate state block (st_x). The state (st_x) is submitted to the SBox Layer XORed again with (S_x) and finally submitted to $PF_{K0}$to generate a ciphertext block (C_x). The process that each plaintext block goes though is $(P_x \rightarrow PF_{K1} \rightarrow XOR(s_x) \rightarrow dirSBox \rightarrow st_x \rightarrow dirSBox \rightarrow XOR(S_x) \rightarrow PF_{K0} \rightarrow C_x)$. It is important to observe that if the plaintext or associate data blocks are swapped in position, the generated checksum will be modified. This characteristic prevents reordering data attacks.

All intermediate state blocks are XORed together to generate a checksum. If the last message block was padded, the checksum is XORed with the bit sequence$(1010 \ldots\ 10)$. If there was no padding it is XORed with the bit sequence $(0101 \ldots 01)$. After it the result is submitted to PF_{K0} function to generate the TAG used for authentication. The TAG length $(Tlen)$ can be smaller than the block size, if it is adequate to the application. This is done by truncating the TAG on its $Tlen$ more significant bits (MSB_{Tlen}).

For decryption, first the Associated Data is submitted to the same process as in encryption $(AD_x \rightarrow PF_{K1} \rightarrow XOR(s_x) \rightarrow dirSBox \rightarrow st_x)$ or $(\ AD_{x(padded)} \rightarrow PF_{K1} \rightarrow PF_{K1} \rightarrow XOR(s_x) \rightarrow dirSBox \rightarrow st_x)$. The (S_n) block is submitted to PF_{K1} twice $(S_n \rightarrow PF_{K1} \rightarrow PF_{K1} \rightarrow (st_n)$. The Ciphertext is broken into blocks and the TAG is separated (as its size is known, the last part of the ciphertext is the TAG). The cipher text blocks are submitted to a reverse process $(C_x \rightarrow INVPF_{K0} \rightarrow XOR(S_x) \rightarrow invSBox \rightarrow st_x \rightarrow invSBox \rightarrow XOR(s_x) \rightarrow INVPF_{K1} \rightarrow P_x)$.

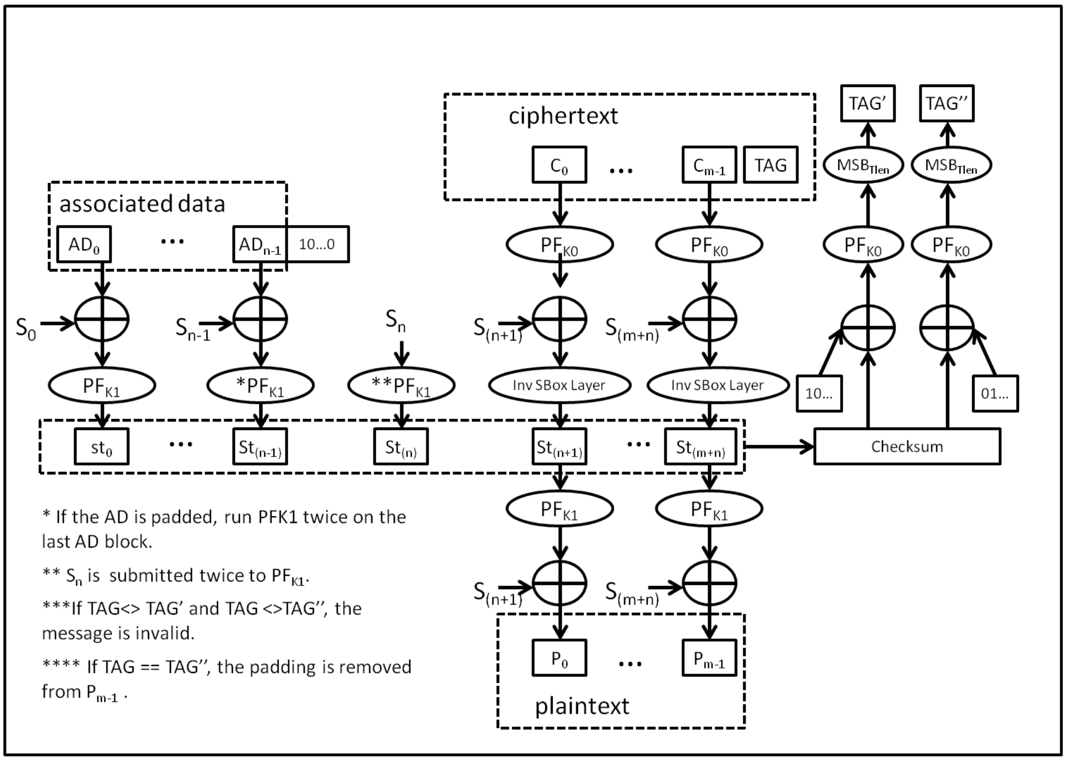

Figure 8. The FlexAEAD v1.2 decryption diagram

During the decryption process all (st_x) are XORed together. This checksum is XORed with bit sequence $(1010 \ldots 10)$ then submitted to (PF_{K0}) to generate a TAG'. If the TAG' is equal to the received TAG, the message is valid and the original plaintext was not padded. If it is different the checksum is XORed with bit sequence $(0101 \ldots 01)$ then submitted to (PF_{K0}) to generate a TAG''. If the TAG'' is equal to the received TAG, the message is valid and the original plaintext was padded. If neither calculated TAGs are equal to the received TAG, the message is invalid and it is discarded.

3. Key and Block Size Selection

Although the FlexAEAD algorithm family allows several block and key size. A few variant were selected as concrete examples for the NIST contest.

The family also allows the user to select the tag, used to validate the message, and nonce size. For this contest they will be the maximum allowed, depending on the variant. The maximum for them is the same as the block size for each variant.

The chosen variants are:

FlexAEAD128b064 – 128 bits key, 64 bits block, 64 bits nonce and 64 bits tag sizes

FlexAEAD128b128 – 128 bits key, 128 bits block, 128 bits nonce and 128 bits tag sizes

FlexAEAD256b128 – 256 bits key, 128bits block, 128 bits nonce and 128 bits tag sizes

FlexAEAD256b256 – 256 bits key, 256 bits block, 256 bits nonce and 256 bits tag sizes

These variants were implemented and the NIST test vectors were successfully generated for them.

4. Differential Cryptanalysis

The differential cryptanalysis (BIHAM and SHAMIR, 1991) technique consists on analyzing of the probabilities of the differences on the cipher SBoxes inputs and outputs.

The differential and the linear cryptanalysis of the FlexAEAD are similar to the analysis performed on the algorithm FlexAE (NASCIMENTO and XEXEO, 2018). The differences are the number of rounds and the inclusion of the function mix adjacent bytes.

To analyze the differences of a specific SBox construction, a difference distribution table (DDT) is created. To create this table the input differences ($\Delta X = X' \oplus X''$) and the output differences ($\Delta Y = Y' \oplus Y''$) are calculated for every possible input pair (X', X''). The table columns are ΔY values and the lines are ΔX. Each cells contains the number of times that ΔX generates ΔY. Exemplifying, considering the AES SBox,

The difference distribution table for AES SBox shows that the maximum probability for any pair $(\Delta X \neq 0, \Delta Y \neq 0)$ is $p = \frac{4}{256} = 2^{-6}$.

To encrypt each ciphertext block the PF_K is executed at least 2 times ($P_x \rightarrow XOR(s_x) \rightarrow PF_{K1} \rightarrow st_x) \rightarrow SBOX\ Layer \rightarrow XOR(S_x) \rightarrow PF_{K0} \rightarrow C_x$). The number of rounds depends on the block size in bytes ($r = \log_2 nb$). The total of rounds for block sizes of 64, 128 and 256 bits are respectively 6, 8 and 10.

For a 64 bits block size: if the 1st round has 1 active SBox[5], the 2nd round will have 3 active SBoxes; the 3rd round will have 7 SBoxes; and from 4th round on, there will have 8 active SBoxes per round. On (r-1) or 5 rounds, there is 27 active Sboxes: $nActiveSboxes = 1 + 3 + 7 + (2 \times 8) = 27$.

[5] On the first round it is possible to control the input difference to force only one 1 active SBox.

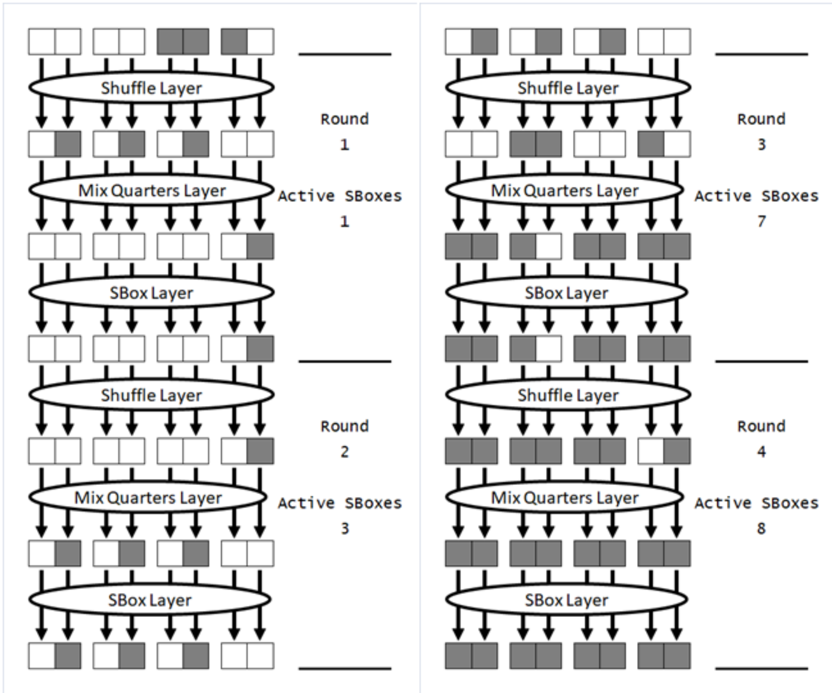

Figure 9. active sboxes after 4 rounds for 64 bits bock size

For a 128 bits block size: if the 1st round has 1 active SBox, the 2nd round has a minimum of 3 active SBoxes; the 3rd round will have 7 active SBoxes; the 4th round – 15 active SBoxes; and from 5th round on, there will be 16 active SBoxes per round. On (r-1) or 7 rounds, there is 74 active Sboxes: $nActiveSboxes = 1 + 3 + 7 + 15 + (3 \times 16) = 74$.

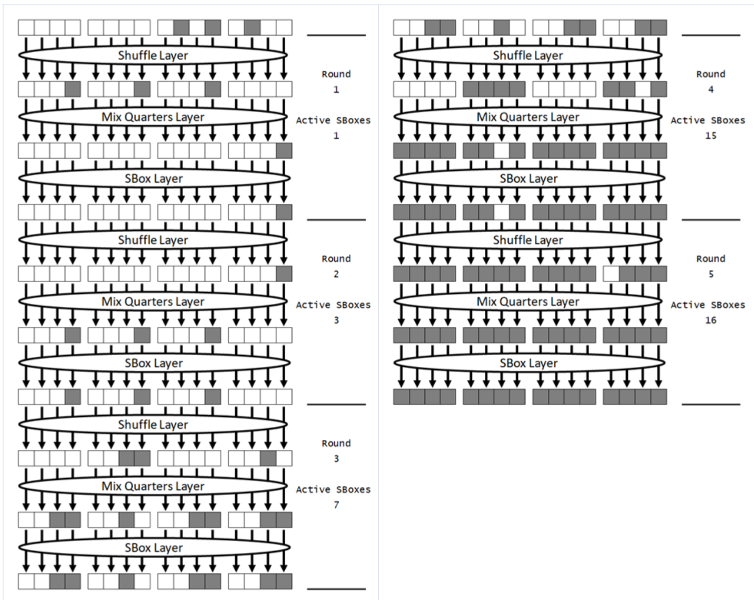

Figure 10. active sboxes after 5 rounds for 128 bits bock size

For a 256 bits block size: there is 26 active SBoxes from round 1 to 4; the 5th round - 31 active SBoxes; from the 6th round on, there is 32 active SBoxes. On (r-1) or 9 rounds, there is 185 active Sboxes: $nActiveSboxes = 26 + 31 + (4 \times 32) = 185$.

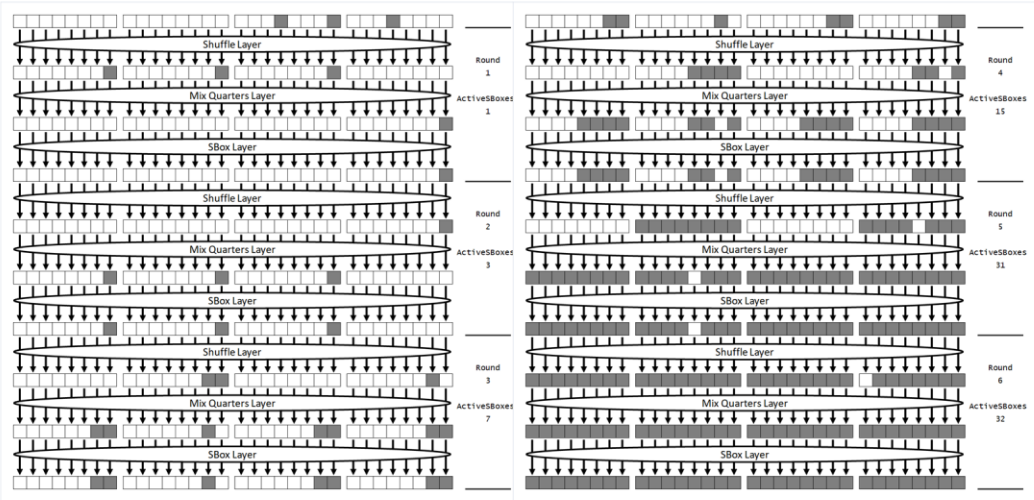

Figure 11. active sboxes after 6 rounds for 256 bits bock size

The maximum probability can be calculated by $p_D = \prod_{i=1}^{(nActiveSboxes)} 2^{-6}$ and the difficult of an attack based on differential cryptanalysis is $N_D \cong \frac{1}{P_D}$ (Heys, 2001).

Table 2. Difficult to perform a differential cryptanalysis attack

Block Size	Rounds (r-1)	Active SBoxes	p_D	N_D
64	5	27	2^{-162}	2^{162}
128	7	74	2^{-444}	2^{444}
256	9	185	2^{-1110}	2^{1110}

An attack based on a differential cryptanalysis is more difficult than a brute force attack in all cases.

5. Linear Cryptanalysis

The linear cryptanalysis (MATSUI, 1993) technique consists in evaluating the cipher using linear expressions to approximate the cipher results and calculating their biases of being true or false. The higher the bias, the easier is to uncover the key bits.

For AES SBox there are a total of 65025 possible linear expressions. The maximum bias on these expression is $\epsilon = \frac{16}{256} = 2^{-4}$.

After calculating the bias for every SBox, the next step is to verify the cipher structure effect and determine the best linear expressions for each round. In this stage it is easier to represent the linear expressions in graphic way. The following has a graphical representation of a linear approximation for all 5 rounds of the PF_K using 64 bits block size.

The complexity of an attack is determined by the number of chosen plaintext pair (N_L) which can be calculate from the bias $N_L = \frac{1}{\epsilon^2}$ (HEYS, 2001). On the linear cryptanalysis, if the number of active SBox is known (n), the bias (ϵ) can be determined subtracting (0.5) from the probability (p) calculated using the Piling-up Lemma $\mathrm{p} = \frac{1}{2} + 2^{\mathrm{n}-1} \prod_{\mathrm{i}=1}^{\mathrm{n}} \left(\mathrm{p_i} - \frac{1}{2}\right)$ (MATSUI,1993): $\epsilon = p - 0.5$.

The number of active SBoxes on the linear cryptanalysis can be considered the same as the differential cryptanalysis per round due to the cipher its internal structure and the effect of the mix adjacent bytes function.

Table 3. Difficult to perform a linear cryptanalysis attack

Block Size	Rounds (r)	Active SBox (r rounds)	Maximum Bias	$N_L = \frac{1}{\epsilon^2}$
64	6	35	$\epsilon = 2^{-106}$	$N_L = 2^{212}$
128	8	90	$\epsilon = 2^{-271}$	$N_L = 2^{542}$
256	10	217	$\epsilon = 2^{-652}$	$N_L = 2^{1304}$

An attack based on a linear cryptanalysis is more difficult than a brute force attack, making it impractical.

6. Using the cipher to generate a pseudorandom sequence

The cipher was used to encrypted a block full of zeros again and again with the same key. The resulted were submitted to the dieharder toll. The sequence passed on all tests except on a few that it randomly presented as “WEAK”. If the NONCE or the KEY is changed or only that test is repeated, the test returned PASSED. This indicates that it is not possible to infer any pattern from the generated sequence. The test was performed on all four variants of the cipher presented on this document (FlexAEAD128b064, FlexAEAD128b128, FlexAEAD256b128 and FlexAEAD256b256). The code used to generate the sequence for the dieharder tool is on the appendices.

7. Cipher family performance

The eBAEAD - ECRYPT Benchmarking of Authenticated Ciphers from supercop framework (Bernstein, 2019) was used to compare the implementations with NIST LWC round2 candidates. A virtual machine with one VCPU (Intel(R) Xeon(R) CPU E5-2697 v4 @ 2.30GHz) running Linux Ubuntu 19.10 was used to evaluate the performance. In total, 92 implementations were compared. The measure was done twice and the median was used for comparison.

The median time for encrypt 2048 bytes message with 2048 bytes associate data for the variants FlexAEAD128b064, FlexAEAD128b128, FlexAEAD256b128 and FlexAEAD256b256 are respectively 184258, 165356, 163948 and 122412 cpu cycles. It position against the other ciphers were 29th, 25th, 24th and 20th. The FlexAEAD256b256 implementation is 4.2 times slower than the fastest implementation (ascon128av12 – 29318) but 229.8 faster than the slowest implementation compared (elephant160v1 - 28131529). The complete table with the comparison is available on the appendixes.

Considering the best implementation of each family, the FlexAEAD cipher family is the 7th position.

8. Conclusion and future works

This paper describes the FlexAEAD cipher family. This cipher was tailored to be lightweight and flexible. Its security was analyzed for three variants with concrete values against linear and differential cryptanalysis attacks. The result is summarized on Table 4. Their capacity to generate a pseudorandom sequence was also confirmed usig the dieharder tool.

Table 4. Variant parameters and cryptanalysis difficulty

Variant	Parameters sizes (in bits)				Cryptanalysis difficulty	
	Key	Block	Nonce	Tag	Linear	Differential
FlexAEAD128b064	128	64	64	64	2^{162}	2^{212}
FlexAEAD128b128	128	128	128	128	2^{444}	2^{542}
FlexAEAD256b128	256	128	128	128	2^{444}	2^{542}
FlexAEAD256b256	256	256	256	256	2^{1110}	2^{1304}

The cipher performance, was evaluated comparing its 4 variant against the 32 ciphers selected for round 2 of the NIST LWC contest. The tests show the FlexAEAD variants are faster than half of the implementations. One performance advantage is its capacity to allow parallel computing, each block can be calculated by a different thread in any order. This characteristic is an advantage when using multicore processors.

For future works, the cipher implementation should be optimized to increase the performance. The cipher should also be implemented in hardware and compared to the other ciphers.

References

BERNSTEIN, D. J.; LANGE, T. eds. eBACS: ECRYPT Benchmarking of Cryptographic Systems. URL: <https://bench.cr.yp.to> Access Date: Feb 28th 2019.

BERNSTEIN, D. J. Cryptographic competitions. URL: < https://competitions.cr.yp.to> Access Date: Feb 28th 2019.

BIHAM, E.; SHAMIR, A. Differential cryptanalysis of DES-like cryptosystems. Journal of CRYPTOLOGY, 4, n. 1, 1991. 3-72.

CRYPTOLUX RESEARCH GROUP - UNIVERSITY OF LUXEMBOURG. Lightweight Block Ciphers, 2016. URL: <https://www.cryptolux.org/index.php/Lightweight_Block_Ciphers>. Access Date: Feb 28th 2019.

DAEMEN, J.; RIJMEN, V. Specification for the advanced encryption standard (AES). Federal Information Processing Standards Publication, 2001.

DINU, D. et al. FELICS – Fair Evaluation of Lightweight Cryptographic Systems, jul. 2015. URL: <http://csrc.nist.gov/groups/ST/lwc-workshop2015/papers/session7-dinu-paper.pdf>. Access Date: Feb 28th 2019.

EICHLSEDER, M. Posting on the NIST LWC mailing list, 2019. URL:< https://groups.google.com/a/list.nist.gov/forum/#!topic/lwc-forum/SgmvFLzFQNI>. Access Date: Jul 21st 2019.

EICHLSEDER, M.; KALES, D.; SCHOFNEGGER, M. Forgery Attacks on FlexAE and FlexAEAD. IACR Cryptology ePrint Archive, Report 2019/679, 2019. URL:<https://eprint.iacr.org/2019/679>. Access Date: Jul 21st 2019.

EVEN, S.; MANSOUR, Y. A construction of a cipher from a single pseudorandom permutation. Journal of Cryptology, 10, 1997. 151-161.

JUTLA, C. S. Encryption modes with almost free message integrity. International Conference on the Theory and Applications of Cryptographic Techniques, 2001. 529-544.

MATSUI, M. Linear cryptanalysis method for DES cipher. Workshop on the Theory and Application of of Cryptographic Techniques, 1993. 386-397.

MÈGE, A.: OFFICIAL COMMENT: FlexAEAD. Posting on the NIST LWC mailing list, 2019. URL:<https://groups.google.com/a/list.nist.gov/forum/#!topic/lwc-forum/DPQVEJ5oBeU> . Access Date: Jul 21st 2019.

NASCIMENTO, E.M.; XEXÉO, J.A.M. "A flexible authenticated lightweight cipher using Even-Mansour construction". 2017 IEEE International Conference on Communications (ICC), Paris, 2017, pp. 1-6. (doi: 10.1109/ICC.2017.7996734). URL:<http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7996734&isnumber=7996317>. Access Date: Feb 28th 2019.

NASCIMENTO, E.M. "Algoritmo de Criptografia Leve com Utilização de Autenticação". 2017. 113p. Dissertação (mestrado) - Instituto Militar de Engenharia, Rio de Janeiro, 2017. URL: <http://www.comp.ime.eb.br/pos/arquivos/publicacoes/dissertacoes/2017/2017-Eduardo.pdf>. Access Date: Feb 28th 2019.

NASCIMENTO, E.M.; XEXÉO, J.A.M. A Lightweight Cipher with Integrated Authentication. In: CONCURSO DE TESES E DISSERTAÇÕES - SIMPÓSIO BRASILEIRO EM SEGURANÇA DA INFORMAÇÃO E DE SISTEMAS COMPUTACIONAIS (SBSEG), 18., 2018, 1. Anais Estendidos do XVIII Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais. Porto Alegre: Sociedade Brasileira de Computação, oct. 2018 . p. 25 - 32.

NASCIMENTO, E.M.; XEXÉO, J.A.M. FlexAEAD - A Lightweight Cipher with Integrated Authentication. Round 1 submission to NIST lightweight cryptography Standardization process, 2019. URL: <https://csrc.nist.gov/CSRC/media/Projects/Lightweight-Cryptography/documents/round-1/spec-doc/FlexAEAD-spec.pdf>. Access Date: Jul 21st 2019.

NASCIMENTO, E.M.; XEXÉO, J.A.M. FlexAEAD v1.1 -A Lightweight AEAD Cipher with Integrated Authentication. Journal of Information Security and Cryptography (Enigma), 6(1), 15-24. 2019. doi:https://doi.org/10.17648/jisc.v6i1.74.

NIST - NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY. Submission Requirements and Evaluation Criteria for the Lightweight Cryptography Standardization Process, 2018. URL:< https://csrc.nist.gov/CSRC/media/Projects/Lightweight-Cryptography/documents/final-lwc-submission-requirements-august2018.pdf>. Access Date: Oct 21st 2019.

RAHMAN, M.; Saha, D.; Paul, G. Posting on the NIST LWC mailing list, 2019. URL:< https://groups.google.com/a/list.nist.gov/forum/#!topic/lwc-forum/VLWtGnJStew> . Access Date: Jul 21st 2019.

RAHMAN, M.; Saha, D.; Paul, G. Iterated Truncated Differential for Internal Keyed Permutation of FlexAEAD. IACR Cryptology ePrint Archive, Report 2019/539, 2019. URL:<https://eprint.iacr.org/2019/539> . Access Date: Jul 21st 2019.

APPENDICE A – Direct and Inverse SBoxes

Table 5. Direct SBox0 (AES SBox)

*	-	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	-	63	7C	77	7B	F2	6B	6F	C5	30	01	67	2B	FE	D7	AB	76
1	-	CA	82	C9	7D	FA	59	47	F0	AD	D4	A2	AF	9C	A4	72	C0
2	-	B7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	D8	31	15
3	-	04	C7	23	C3	18	96	05	9A	07	12	80	E2	EB	27	B2	75
4	-	09	83	2C	1A	1B	6E	5A	A0	52	3B	D6	B3	29	E3	2F	84
5	-	53	D1	00	ED	20	FC	B1	5B	6A	CB	BE	39	4A	4C	58	CF
6	-	D0	EF	AA	FB	43	4D	33	85	45	F9	02	7F	50	3C	9F	A8
7	-	51	A3	40	8F	92	9D	38	F5	BC	B6	DA	21	10	FF	F3	D2
8	-	CD	0C	13	EC	5F	97	44	17	C4	A7	7E	3D	64	5D	19	73
9	-	60	81	4F	DC	22	2A	90	88	46	EE	B8	14	DE	5E	0B	DB
A	-	E0	32	3A	0A	49	06	24	5C	C2	D3	AC	62	91	95	E4	79
B	-	E7	C8	37	6D	8D	D5	4E	A9	6C	56	F4	EA	65	7A	AE	08
C	-	BA	78	25	2E	1C	A6	B4	C6	E8	DD	74	1F	4B	BD	8B	8A
D	-	70	3E	B5	66	48	03	F6	0E	61	35	57	B9	86	C1	1D	9E
E	-	E1	F8	98	11	69	D9	8E	94	9B	1E	87	E9	CE	55	28	DF
F	-	8C	A1	89	0D	BF	E6	42	68	41	99	2D	0F	B0	54	BB	16

Table 6. Inverse SBox0 (AES SBox)

*	-	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	-	52	09	6A	D5	30	36	A5	38	BF	40	A3	9E	81	F3	D7	FB
1	-	7C	E3	39	82	9B	2F	FF	87	34	8E	43	44	C4	DE	E9	CB
2	-	54	7B	94	32	A6	C2	23	3D	EE	4C	95	0B	42	FA	C3	4E
3	-	08	2E	A1	66	28	D9	24	B2	76	5B	A2	49	6D	8B	D1	25
4	-	72	F8	F6	64	86	68	98	16	D4	A4	5C	CC	5D	65	B6	92
5	-	6C	70	48	50	FD	ED	B9	DA	5E	15	46	57	A7	8D	9D	84
6	-	90	D8	AB	00	8C	BC	D3	0A	F7	E4	58	05	B8	B3	45	06
7	-	D0	2C	1E	8F	CA	3F	0F	02	C1	AF	BD	03	01	13	8A	6B
8	-	3A	91	11	41	4F	67	DC	EA	97	F2	CF	CE	F0	B4	E6	73
9	-	96	AC	74	22	E7	AD	35	85	E2	F9	37	E8	1C	75	DF	6E
A	-	47	F1	1A	71	1D	29	C5	89	6F	B7	62	0E	AA	18	BE	1B
B	-	FC	56	3E	4B	C6	D2	79	20	9A	DB	C0	FE	78	CD	5A	F4
C	-	1F	DD	A8	33	88	07	C7	31	B1	12	10	59	27	80	EC	5F
D	-	60	51	7F	A9	19	B5	4A	0D	2D	E5	7A	9F	93	C9	9C	EF
E	-	A0	E0	3B	4D	AE	2A	F5	B0	C8	EB	BB	3C	83	53	99	61
F	-	17	2B	04	7E	BA	77	D6	26	E1	69	14	63	55	21	0C	7D

Table 7. Direct SBox1

*	-	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	-	95	A8	6C	C4	69	1F	3D	EC	8C	F8	B7	31	C1	3F	29	56
1	-	7E	D4	44	E0	E3	86	C7	F3	D8	F0	C0	0B	AC	4C	74	A1
2	-	60	C3	35	34	7D	87	2F	98	AE	97	1C	49	BC	A5	A6	1A
3	-	33	DF	27	55	58	03	DA	6E	09	48	1E	78	02	88	8F	DE
4	-	6F	53	D9	5E	A2	BD	22	61	E1	E2	9C	21	C8	CE	13	9F
5	-	08	75	94	16	36	D5	FB	40	01	79	EA	3A	6B	F2	52	E7
6	-	C6	BA	D7	A7	AB	B0	F5	FA	73	2B	B9	38	32	FE	68	9B
7	-	DB	AA	7B	43	37	9E	04	7A	39	1D	1B	D1	FF	64	57	2D
8	-	E8	FD	91	66	B3	59	17	7F	0E	DC	81	12	4E	A9	EF	F9
9	-	AF	CD	2E	80	76	62	CF	14	3B	8A	5F	2C	B1	41	F7	D6
A	-	5B	71	82	CA	15	3E	54	5C	23	4F	B5	FC	C5	7C	18	CC
B	-	B8	2A	84	D3	4D	4A	25	F6	8D	89	26	00	11	4B	CB	F1
C	-	3C	DD	65	28	B4	96	EB	BF	ED	83	07	9A	C2	8E	45	72
D	-	E6	93	AD	BE	E4	9D	24	19	46	E9	20	47	0C	06	92	E5
E	-	B2	BB	6D	30	85	42	99	0D	A3	5A	77	8B	5D	0F	05	EE
F	-	A4	50	B6	70	D2	51	D0	90	A0	63	0A	67	F4	6A	C9	10

Table 8. Inverse SBox1

*	-	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	-	BB	58	3C	35	76	EE	DD	CA	50	38	FA	1B	DC	E7	88	ED
1	-	FF	BC	8B	4E	97	A4	53	86	AE	D7	2F	7A	2A	79	3A	05
2	-	DA	4B	46	A8	D6	B6	BA	32	C3	0E	B1	69	9B	7F	92	26
3	-	E3	0B	6C	30	23	22	54	74	6B	78	5B	98	C0	06	A5	0D
4	-	57	9D	E5	73	12	CE	D8	DB	39	2B	B5	BD	1D	B4	8C	A9
5	-	F1	F5	5E	41	A6	33	0F	7E	34	85	E9	A0	A7	EC	43	9A
6	-	20	47	95	F9	7D	C2	83	FB	6E	04	FD	5C	02	E2	37	40
7	-	F3	A1	CF	68	1E	51	94	EA	3B	59	77	72	AD	24	10	87
8	-	93	8A	A2	C9	B2	E4	15	25	3D	B9	99	EB	08	B8	CD	3E
9	-	F7	82	DE	D1	52	00	C5	29	27	E6	CB	6F	4A	D5	75	4F
A	-	F8	1F	44	E8	F0	2D	2E	63	01	8D	71	64	1C	D2	28	90
B	-	65	9C	E0	84	C4	AA	F2	0A	B0	6A	61	E1	2C	45	D3	C7
C	-	1A	0C	CC	21	03	AC	60	16	4C	FE	A3	BE	AF	91	4D	96
D	-	F6	7B	F4	B3	11	55	9F	62	18	42	36	70	89	C1	3F	31
E	-	13	48	49	14	D4	DF	D0	5F	80	D9	5A	C6	07	C8	EF	8E
F	-	19	BF	5D	17	FC	66	B7	9E	09	8F	67	56	AB	81	6D	7C

Table 9. Direct SBox2

*	-	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	-	A6	9D	5F	08	3E	7B	F1	B0	8E	EC	2C	0C	69	B6	AD	ED
1	-	B2	60	E7	F8	E3	39	97	11	41	DB	AE	27	23	3F	67	51
2	-	C8	B3	A1	4B	62	A9	89	2E	04	20	0D	72	5A	26	19	7C
3	-	55	36	18	1B	C6	D4	66	0A	00	34	0E	74	22	B9	5D	D3
4	-	F5	CD	48	84	25	73	50	14	C4	43	45	6F	31	E8	86	E9
5	-	F7	7A	E5	D6	17	32	CC	E0	D8	C2	E6	35	79	29	AF	77
6	-	3B	90	EE	12	F9	02	1C	BA	96	DE	FB	A4	A2	CB	94	A3
7	-	91	57	8B	3C	F2	2F	CF	61	80	E4	4D	9C	5B	15	78	B1
8	-	0F	AB	13	A7	B5	44	B7	70	03	83	4C	98	DD	4F	FF	8A
9	-	F3	FA	30	4E	33	D0	42	D5	6D	5C	81	95	D2	2B	01	99
A	-	6A	56	AC	B4	07	CA	9E	EF	1A	EA	88	C1	93	8D	E1	7D
B	-	FD	A5	F0	3A	E2	B8	0B	C5	49	6E	05	71	46	1F	2A	8F
C	-	68	F6	D9	38	82	47	FC	7E	09	37	F4	1D	9F	A0	A8	52
D	-	DA	24	FE	75	6C	BC	C3	63	C0	9B	10	BD	BF	1E	40	4A
E	-	59	16	5E	BB	54	C7	EB	64	8C	9A	06	3D	76	28	21	BE
F	-	D1	85	87	AA	53	CE	DF	65	58	DC	7F	D7	C9	6B	2D	92

Table 10. Inverse SBox2

*	-	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	-	38	9E	65	88	28	BA	EA	A4	03	C8	37	B6	0B	2A	3A	80
1	-	DA	17	63	82	47	7D	E1	54	32	2E	A8	33	66	CB	DD	BD
2	-	29	EE	3C	1C	D1	44	2D	1B	ED	5D	BE	9D	0A	FE	27	75
3	-	92	4C	55	94	39	5B	31	C9	C3	15	B3	60	73	EB	04	1D
4	-	DE	18	96	49	85	4A	BC	C5	42	B8	DF	23	8A	7A	93	8D
5	-	46	1F	CF	F4	E4	30	A1	71	F8	E0	2C	7C	99	3E	E2	02
6	-	11	77	24	D7	E7	F7	36	1E	C0	0C	A0	FD	D4	98	B9	4B
7	-	87	BB	2B	45	3B	D3	EC	5F	7E	5C	51	05	2F	AF	C7	FA
8	-	78	9A	C4	89	43	F1	4E	F2	AA	26	8F	72	E8	AD	08	BF
9	-	61	70	FF	AC	6E	9B	68	16	8B	9F	E9	D9	7B	01	A6	CC
A	-	CD	22	6C	6F	6B	B1	00	83	CE	25	F3	81	A2	0E	1A	5E
B	-	07	7F	10	21	A3	84	0D	86	B5	3D	67	E3	D5	DB	EF	DC
C	-	D8	AB	59	D6	48	B7	34	E5	20	FC	A5	6D	56	41	F5	76
D	-	95	F0	9C	3F	35	97	53	FB	58	C2	D0	19	F9	8C	69	F6
E	-	57	AE	B4	14	79	52	5A	12	4D	4F	A9	E6	09	0F	62	A7
F	-	B2	06	74	90	CA	40	C1	50	13	64	91	6A	C6	B0	D2	8E

Table 11. Direct SBox3

*	-	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	-	D9	EE	83	B5	F4	02	EF	64	8E	4D	34	48	C2	29	C6	90
1	-	B3	9F	52	22	2F	E7	D0	76	95	8D	A1	2B	56	D7	7D	1C
2	-	2D	9A	3B	12	DD	00	24	A2	63	11	07	94	5D	F6	0E	7F
3	-	FF	5E	F3	65	E5	F1	A0	93	1E	BC	DE	A9	8B	F5	FA	B2
4	-	62	7E	B9	57	69	4C	FD	43	1A	08	35	05	E6	88	A5	44
5	-	45	01	BD	5B	B6	CC	BE	D3	9B	9E	8F	40	32	C3	8A	3E
6	-	0B	58	DB	99	0D	E1	87	B8	06	0F	0C	66	A4	FE	3D	10
7	-	FB	BB	6B	53	5A	C1	20	42	31	7C	CF	E0	89	E2	6C	09
8	-	04	17	CB	C0	E9	AC	5F	4E	81	8C	13	BA	0A	CE	55	23
9	-	38	4B	F0	79	6E	21	B7	82	46	D1	71	BF	26	86	D6	2E
A	-	97	C9	74	A6	2A	98	59	DA	AF	78	92	28	6A	6D	1D	4F
B	-	F8	61	7A	60	F2	6F	15	C4	ED	16	D4	EA	70	CD	EB	DC
C	-	B0	77	19	3A	D8	5C	F9	27	72	50	C5	3C	37	E3	A8	AA
D	-	F7	2C	73	1F	33	75	C7	68	67	36	4A	96	AB	EC	FC	1B
E	-	C8	7B	E8	A3	80	B4	9C	AE	18	41	D5	E4	25	51	14	49
F	-	AD	3F	CA	91	D2	A7	84	9D	30	DF	85	47	03	39	B1	54

Table 12. Inverse SBox3

*	-	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	-	25	51	05	FC	80	4B	68	2A	49	7F	8C	60	6A	64	2E	69
1	-	6F	29	23	8A	EE	B6	B9	81	E8	C2	48	DF	1F	AE	38	D3
2	-	76	95	13	8F	26	EC	9C	C7	AB	0D	A4	1B	D1	20	9F	14
3	-	F8	78	5C	D4	0A	4A	D9	CC	90	FD	C3	22	CB	6E	5F	F1
4	-	5B	E9	77	47	4F	50	98	FB	0B	EF	DA	91	45	09	87	AF
5	-	C9	ED	12	73	FF	8E	1C	43	61	A6	74	53	C5	2C	31	86
6	-	B3	B1	40	28	07	33	6B	D8	D7	44	AC	72	7E	AD	94	B5
7	-	BC	9A	C8	D2	A2	D5	17	C1	A9	93	B2	E1	79	1E	41	2F
8	-	E4	88	97	02	F6	FA	9D	66	4D	7C	5E	3C	89	19	08	5A
9	-	0F	F3	AA	37	2B	18	DB	A0	A5	63	21	58	E6	F7	59	11
A	-	36	1A	27	E3	6C	4E	A3	F5	CE	3B	CF	DC	85	F0	E7	A8
B	-	C0	FE	3F	10	E5	03	54	96	67	42	8B	71	39	52	56	9B
C	-	83	75	0C	5D	B7	CA	0E	D6	E0	A1	F2	82	55	BD	8D	7A
D	-	16	99	F4	57	BA	EA	9E	1D	C4	00	A7	62	BF	24	3A	F9
E	-	7B	65	7D	CD	EB	34	4C	15	E2	84	BB	BE	DD	B8	01	06
F	-	92	35	B4	32	04	3D	2D	D0	B0	C6	3E	70	DE	46	6D	30

APPENDICE B – encrypt-dieharder.c code to generate pseudorandom sequence

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "FlexAEADv1.2.c"

int main (  ) {

   unsigned char *npub;
   unsigned char *k;
   unsigned char *state;

   struct FlexAEADv1 flexaeadv1;

   k = malloc(KEYSIZE);
   memset( k, 0x00, KEYSIZE);

   npub = malloc(BLOCKSIZE);
   memset( npub, 0x00, BLOCKSIZE);

   FlexAEADv1_init( &flexaeadv1, k );

   fprintf(stderr, "FlexAEADv1 ZERO %d %d\n", BLOCKSIZE*8, KEYSIZE*8 );


   // ### reset the counter and checksum
   memcpy( flexaeadv1.counter, npub, NONCESIZE);
   dirPFK(     flexaeadv1.counter,     flexaeadv1.nBytes,     (flexaeadv1.subkeys     +
   (4*flexaeadv1.nBytes)),  flexaeadv1.nRounds, flexaeadv1.state );
   memcpy( flexaeadv1.add, flexaeadv1.counter, NONCESIZE);
   dirPFK(     flexaeadv1.counter,     flexaeadv1.nBytes,     (flexaeadv1.subkeys     +
   (4*flexaeadv1.nBytes)),  flexaeadv1.nRounds, flexaeadv1.state );

   state = malloc(BLOCKSIZE);
   while(1)
   {
       memset( state, 0x00, BLOCKSIZE );
       mwc32( flexaeadv1.counter, flexaeadv1.add, flexaeadv1.nBytes );
       encryptBlock( &flexaeadv1, state);
       fwrite(state, 1, flexaeadv1.nBytes, stdout);
   }

   free(state);
}

// execution example: ./encrypt-dieharder | dieharder -a -g 200
```

APPENDICE C – FlexAEAD plus NIST LWC round 2 family comparison using supercop metrics.

Order	Implementation	Family	Running Time (CPU Cycles)
1	ascon128av12	ascon	29318
2	xoodyakv1	xoody	59900
3	gimli24v1	gimli	60210
4	saeaes128a120t64v1	saeas	72165
5	knot128v2	knot	74529
6	schwaemm256128v1	sparkle	97388
7	flexaead256b256v12	flexaead	122412
8	comet128aesv1	comet	128220
9	tinyjambu128	TinyJambu	178573
10	saturninctrcascadev2	saturnin	185784
11	isapa128av20	isap	227008
12	grain128aead	grain	249013
13	drygascon128	drygascon	290076
14	sestatetweaes128v1	estate	342294
15	elephant200v1	elephant	376517
16	spook128mu384v1	spook	397602
17	mixfeed	mixfeed	490278
18	subterraneanv1	subterrean	621702
19	romulusn3v12	romulus	1304337
20	giftcofb128v1	gift-cofb	1314584
21	pyjamask128aeadv1	pyjamask	1442589
22	skinnyaeadtk296128v1	skinny	1466825
23	sundaegift0v1	Sunday-gift	1747733
24	spoc128sliscplight256v1	spoc	2034062
25	spix128v1	spix	2066541
26	paefforkskinnyb128t192n48v1	forkae	2101380
27	wageae128v1	wage	2387185
28	hyenav1	hyena	4994081
29	aceae128v1	ace	5238965
30	photonbeetleaead128rate128v1	photonbeetle	5923798
31	twegift64locusaeadv1	lotus-aead	8280978
32	orangezestv1	orange	8435062
33	oribatida192v12	oribatida	11647161

APPENDICE D – FlexAEAD plus NIST LWC round 2 implementations comparison using supercop metrics.

Order	Implementation	Running Time (CPU Cycles)	Ratio to the fastest	Ratio to the slowest
1	ascon128av12	29318	1,00	959,53
2	ascon128v12	43504	1,48	646,64
3	ascon80pqv12	43845	1,50	641,61
4	xoodyakv1	59900	2,04	469,64
5	gimli24v1	60210	2,05	467,22
6	saeaes128a120t64v1	72165	2,46	389,82
7	saeaes128a120t128v1	72257	2,46	389,33
8	knot128v2	74529	2,54	377,46
9	saeaes192a120t128v1	83798	2,86	335,71
10	saeaes128a64t64v1	92152	3,14	305,27
11	saeaes128a64t128v1	92234	3,15	305,00
12	saeaes256a120t128v1	95846	3,27	293,51
13	knot128v1	96333	3,29	292,02
14	schwaemm256128v1	97388	3,32	288,86
15	saeaes192a64t128v1	107388	3,66	261,96
16	saeaes192a64t64v1	108514	3,70	259,24
17	schwaemm256256v1	120749	4,12	232,98
18	saeaes256a64t128v1	122100	4,16	230,40
19	saeaes256a64t64v1	122178	4,17	230,25
20	**flexaead256b256v12**	**122412**	**4,18**	**229,81**
21	schwaemm192192v1	125570	4,28	224,03
22	comet128aesv1	128220	4,37	219,40
23	schwaemm128128v1	133025	4,54	211,48
24	**flexaead256b128v12**	**163948**	**5,59**	**171,59**
25	**flexaead128b128v12**	**165356**	**5,64**	**170,13**
26	tinyjambu128	178573	6,09	157,54
27	tinyjambu192	181889	6,20	154,66
28	saturninctrcascadev2	185784	6,34	151,42
29	**flexaead128b064v12**	**184258**	**6,28**	**152,67**
30	knot256	193097	6,59	145,69
31	knot192	212229	7,24	132,55
32	tinyjambu256	219892	7,50	127,93
33	isapa128av20	227008	7,74	123,92
34	grain128aead	249013	8,49	112,97
35	drygascon128	290076	9,89	96,98
36	isapa128v20	291570	9,95	96,48
37	drygascon256	341380	11,64	82,41
38	sestatetweaes128v1	342294	11,68	82,19
39	elephant200v1	376517	12,84	74,72
40	spook128mu384v1	397602	13,56	70,75
41	spook128mu512v1	400485	13,66	70,24
42	spook128su512v1	424144	14,47	66,33
43	spook128su384v1	424165	14,47	66,32
44	estatetweaes128v1	448988	15,31	62,66
45	mixfeed	490278	16,72	57,38
46	comet64speckv1	523365	17,85	53,75
47	comet64chamv1	533802	18,21	52,70
48	isapk128av20	549780	18,75	51,17
49	comet128chamv1	580788	19,81	48,44
50	subterraneanv1	621702	21,21	45,25
51	isapk128v20	949978	32,40	29,61
52	romulusn3v12	1304337	44,49	21,57
53	giftcofb128v1	1314584	44,84	21,40
54	pyjamask128aeadv1	1442589	49,20	19,50
55	skinnyaeadtk296128v1	1466825	50,03	19,18
56	skinnyaeadtk29664v1	1477820	50,41	19,04
57	pyjamask96aeadv1	1553833	53,00	18,10
58	sundaegift0v1	1747733	59,61	16,10
59	sundaegift64v1	1753566	59,81	16,04
60	sundaegift96v1	1771096	60,41	15,88
61	sundaegift128v1	1785456	60,90	15,76
62	romulusm3v12	1797693	61,32	15,65
63	romulusn1v12	1973774	67,32	14,25
64	spoc128sliscplight256v1	2034062	69,38	13,83
65	spix128v1	2066541	70,49	13,61
66	romulusn2v12	2087824	71,21	13,47
67	paefforkskinnyb128t192n48v1	2101380	71,68	13,39
68	saefforkskinnyb128t192n56v1	2122993	72,41	13,25
69	paefforkskinnyb128t256n112v1	2147240	73,24	13,10
70	saefforkskinnyb128t256n120v1	2165208	73,85	12,99
71	skinnyaeadtk39664v1	2191586	74,75	12,84
72	skinnyaeadtk3128128v1	2192532	74,78	12,83
73	skinnyaeadtk396128v1	2193596	74,82	12,82
74	skinnyaeadtk312864v1	2228684	76,02	12,62
75	wageae128v1	2387185	81,42	11,78
76	romulusm1v12	2661749	90,79	10,57
77	spoc64sliscplight192v1	2676209	91,28	10,51
78	romulusm2v12	2806694	95,73	10,02
79	paefforkskinnyb128t288n104v1	3444336	117,48	8,17
80	hyenav1	4994081	170,34	5,63
81	aceae128v1	5238965	178,69	5,37
82	paefforkskinnyb64t192n48v1	5493556	187,38	5,12
83	photonbeetleaead128rate128v1	5923798	202,05	4,75
84	estatetwegift128v1	7608370	259,51	3,70
85	twegift64locusaeadv1	8280978	282,45	3,40
86	twegift64lotusaeadv1	8287825	282,69	3,39
87	orangezestv1	8435062	287,71	3,34
88	oribatida192v12	11647161	397,27	2,42
89	oribatida256v12	12775173	435,75	2,20
90	elephant176v1	22550562	769,17	1,25
91	photonbeetleaead128rate32v1	24505944	835,87	1,15
92	elephant160v1	28131529	959,53	1,00