SECURITY AND COMMUNICATION NETWORKS
Security Comm. Networks 2016; **9:**1119–1131
Published online 27 December 2015 in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/sec.1404

RESEARCH ARTICLE

Tweakable parallel OFB mode of operation with delayed thread synchronization

Boris Damjanović[1]* and Dejan Simić[2]

[1] Department of Information Systems, Faculty of Organizational Sciences, University of Belgrade, Jove Ilica 154, Belgrade, Serbia
[2] Department of Information Technology, Faculty of Organizational Sciences, University of Belgrade, Jove Ilica 154, Belgrade, Serbia

ABSTRACT

Introduction of various cryptographic modes of operation is induced with noted imperfections of symmetric block algorithms. Design of some cryptographic modes of operation has already been exploited as an idea for parallelization of certain algorithms execution. To the best of our knowledge, there is no evidence in the available literature that output feedback (OFB) mode, which is used in satellite communications, has ever been parallelized. In this paper, we consider the performance of a convenient mode of operation, which performs tweakable parallel encryption using xor encrypt xor (XEX) and xor encrypt (XE) constructions in OFB like mode. We make use of an idea similar to the XTS-AES in order to create two parallel tweakable block ciphers. The first of them is designed using XEX construction, while the second is based on XE construction. Each cipher uses two threads to produce corresponding keystreams. Keystreams are first merged with each other and then used in modified tweakable parallel OFB mode of operation. As a proof of the concept, we have implemented a Java application in which these parallel solutions are applied to collect empirical data. The results obtained show that under certain conditions tweakable parallel OFB modes using XEX and XE constructions can achieve performance accelerations up to 10% and to 20%, respectively. Copyright © 2015 John Wiley & Sons, Ltd

KEYWORDS
cryptography; parallel programming; performance analysis; AES

***Correspondence**
Boris Damjanović PhD, Department of Information Systems, Faculty of Organizational Sciences, University of Belgrade, Jove Ilica 154, Belgrade, Serbia.
E-mail: damjanovic@koledzprijedor.org

1. INTRODUCTION

In this paper, we examine the performance of the proposed parallel mode of operation, which utilizes tweakable parallel encryption using XEX and XE constructions in output feedback (OFB) like mode. New design is based on the fact that in OFB mode of operation block cipher decryption operation is not needed and that OFB mode is very similar to a stream cipher [1]. In realization of its design, we use construction similar to XTS-AES tweakable block cipher [2] and combine it with OFB mode of operation. Finally, we differentiate design of two new modes of operation by using XEX and XE constructions [3]. Regarding the known literature, there is no available study that considers the acceleration of the standard OFB mode performance.

We look into the possibility of improving performance of OFB mode by utilizing two parallel threads. Each thread employs advanced encryption standard (AES) algorithm in tweakable OFB mode of operation to create keystream of bytes. Further on, these two keystreams are combined with plaintext to produce the ciphertext and vice versa. It is necessary to ensure that both threads are completed with data processing in order to merge both keystreams with plaintext (ciphertext) and to resume its execution after that. If threads are synchronized after encryption of each block, it will adversely affect their performance. However, if we postpone thread synchronizatiofn, algorithm will gain acceleration.

It is easy to implement AES algorithm in tweakable XEX and XE parallel OFB modes of operation with postponed thread synchronization in order to collect empirical data. The results of measurements show that under certain conditions tweakable parallel OFB mode using XEX construction can achieve performance accelerations of up to 10%, while XE construction can attain performance accelerations of up to 20%.

The remainder of the paper is organized as follows. In Section 2, the necessary background is introduced

including definitions of OFB mode of operation, XEX and XE constructions as well as XTS-AES mode of operation. Further on, Section 3 describes the principal idea of the algorithm, which is designed to demonstrate the concept of parallelization. Our proposed tweakable parallel OFB mode of operation using XEX and XE constructions is introduced in Section 4. The performance of the proposed mode is evaluated in Section 5, and the conclusion comes with Section 6 of the paper. Figures and tables are shown in Appendices A and B.

2. BACKGROUND AND RELATED WORK

Nowadays, there are two major mechanisms that symmetric key algorithms [1,4] use as a method for data encryption. Algorithm can be applied to blocks of data or on individual bits in a stream. Block ciphers repeatedly take input blocks and then have it encrypted with a provided key. Thanks to its great speed and efficiency; symmetric key ciphers are used in applications where large data quantity is to be encrypted.

Two identical blocks of plaintext that are encrypted with the same key will produce two identical blocks of ciphertext. This imperfection of block algorithms has led to emergence of numerous cryptographic modes of operation [4–7]. Even though there are five basic [4,8] or traditional [1] modes of operation, Phillipe Rogaway [5] describes as many as 17 modes of operation. As stated in [9], stream ciphers apply rather simple encryption transformation that depends on the used keystream.

A block cipher is a function, which maps n-bit plaintext blocks to n-bit ciphertext blocks [9]. A tweakable blockcipher is a block cipher with an additional input called a tweak. The tweak is meant to provide variability and not security [10]. The notion of tweakable blockcipher was formalized by Liskov *et al*. [11]. Rogaway [3] proposed efficient ways to turn a blockcipher into a tweakable blockcipher by using XE and XEX constructions. The IEEE ratified IEEE 1619 2007 standard [2] that specifies a tweakable narrow block cipher named XTS-AES (XEX Tweakable Block Cipher with Ciphertext Stealing). In its publication 800-38E [12], NIST approved the XTS-AES mode of the AES algorithm for protecting the confidentiality of data on storage devices.

Existence of various cryptographic modes of operation has already been exploited as a concept for parallelization of certain algorithms execution. According to Lipmaa *et al*. [13], counter (CTR) mode of operation can be parallelized because blocks C1, C2,. . . , CN can be calculated in advance and encrypted simultaneously. Expedite development of GPU (Graphic Processing Unit) as well as environments such as NVidia CUDA or OpenCL has led to different attempts to have AES algorithm parallelized with CTR mode of operation. In accordance with Tran *et al*. [14], the authors initially enlarge block size in comparison with standard AES algorithm and then use coarse grain design to parallelise its execution. Di Biagio *et al*. [15] use fine grain design and internal parallelism of each round. Roch and Jacquin [16] modify already existing open source solution offered by Can Berk Güder [17] in order to expedite the execution through coarse grain implementation and CTR mode of operation. Zola and De Bona [18] use CUDA architecture and CTR mode of operation with WAES library. According to [18], WAES library is the first library that used pre-computing or pre-processing [13] for increasing the processing speed. Authors refer to this attribute of WAES library as speculative encryption [18].

Apart from transforming block algorithm into a streaming algorithm, CFB, OFB, and CTR modes of operation have another common feature; both encryption and decryption operations do not require any inverse block cipher calls [1]. That characteristic can also be exploited in the parallelisation of some cryptographic algorithm through its minor modifications.

As mentioned in [19–22], Java multithreaded application execution depends on many factors (JIT compilation, thread scheduling, garbage collection, number of running processes, and thread synchronization). Chen, Chang, and Hou [23] stress out three of the most important issues, which affect multithreaded Java application performance. At first, lock contentions could degrade the performance and limit the scalability. At next point the authors emphasize memory stall cycles, which are produced by L2 cache misses and cache-to-cache transfers. The low performance of memory systems could reduce profit that occurs because of the use of multiple cores and threads. Tarvo and Reiss [24] pointed out that synchronization operations and limited computational resources produce complex dependencies between a multithreaded program and its performance. Additionally, they stress out that the garbage collection mechanism could be influential factor of performance degradation. In their separate researches, Zhang *et al*. [25] and Gu *et al*. [26] state that thread creation impairs the performance.

2.1. Output feedback mode of operation

Output feedback mode with its ability to work like stream cipher is the basis of the proposed parallel encryption design. In this mode of operation, not a single part of open text is ever taken as input data in an underlying algorithm. Initialisation vector is consecutively encrypted in OFB mode of operation in order to obtain a continuous stream of random bytes. That is why this mode of operation is very similar to streaming algorithms. A stream cipher algorithm generates a stream of random bytes, which is then merged with open text using xor operation thus forming cipher text [1]. OFB mode of operation requires initialisation vector to be nonce, that is, unique in each execution for given key [6].

Output feedback mode is a stream cipher. Thus, a 1-bit error in the ciphertext will produce a 1-bit error in the plaintext. This characteristic makes the OFB mode extremely valuable for satellite communications, where

DOI: 10.1002/sec

transmission channel is noisy [27–29]. To our best knowledge, there are no published papers analyzing possibilities of OFB mode parallelization. As pointed out in [6], OFB mode of operation is recommended for use with any approved block cipher, such as the AES algorithm.

In NIST Special Publication 800-38A, the output feedback mode of operation is defined as follows [6]:

Listing 1. ***Pseudo code for output feedback ciphering and deciphering***

OFB encryption

$I_1 = IV$
$I_j = O_{j-1}$ *For* $j = 2, \ldots, n$;
$O_j = Ciph_K(I_j)$ *For* $j = 1, \ldots, n$;
$C_j = P_j \oplus O_j$ *For* $j = 1, \ldots, n-1$;
$C_n^* = P_n^* \oplus MSB_U(O_n)$

OFB decryption

$I_1 = IV$
$I_j = O_{j-1}$ *For* $j = 2, \ldots, n$;
$O_j = Ciph_K(I_j)$ *For* $j = 1, \ldots, n$;
$P_j = C_j \oplus O_j$ *For* $j = 1, \ldots, n-1$;
$P_n^* = C_n^* \oplus MSB_U(O_n)$

Whereby:

- *IV: initialization vector*
- I_j: j^{th} *input block,*
- O_j: j^{th} *output block,*
- P_j: j^{th} *plain text block,*
- C_j: j^{th} *cipher text block,*
- P_n^*: *last plain text block; can be part of a block too,*
- C_n^*: *last cipher text block, can be part of a block too,*
- $MSB_M(X)$: *most significant M bits of bitstream X,*
- *b: block lenght in bits.*

In OFB mode of operation, the stream of output blocks O_j is combined using xor operation with consecutive plaintext blocks (P_j) in order to produce blocks of ciphertext (C_j). Output blocks are also fed back partially or as the whole to be used as input for underlying algorithm. If the last block is a partial u-bit long block, most significant u-bits of that block are used in xor operation while the rest of (b-u) bits are disregarded [6].

In course of OFB decryption, the initialisation vector is consecutively transformed through forward cipher function $Ciph_K$ of underlying algorithm in order to generate a stream of output blocks (O_j). The output blocks are then xored with blocks of ciphertext to restore plaintext.

Output feedback mode of operation requires unique initialisation vector for each message that has ever been ciphered with a single key. If the same IV were used while encrypting more than one message, confidentiality of these messages would be compromised.

International Organization for Standardization (ISO) standard specification [30] defines a version of output feedback mode that extends FIPS 81 [31] by an additional parameter j. Only j left side bits of each block Yi are used to mask plain text P in order to produce cipher text C.

Output feedback mode of operation as defined in [6] can be shown as in Figure A1.

In both encryption and decryption with OFB mode, forward cipher function of each block but the very first one depends on the results of the previous round. It should be noticed that forward cipher transformation is sufficient for function of OFB mode of operation.

2.2. Xor encrypt xor, XE constructions, and XTS-AES mode of operation

An important role in the proposed parallel encryption design has XEX and XE constructions [3,11] as well as the XTS-AES mode of operation [2].

Liskov, Rivest, and Wagner [11] defined the notion of a tweakable blockcipher. Rogaway [3] formally defined tweakable blockcipher as a map:

$$\tilde{E} : KxTx\{0,1\}^n \rightarrow \{0,1\}^n \tag{1}$$

where each

$$\tilde{E}_T^K(\cdot) = \tilde{E}(K, T, \cdot) \tag{2}$$

is a permutation and T is the set of tweaks.

Xor encrypt xor mode is proposed by Rogaway [3] and instantiated by IEEE [2] and NIST [12] through XTS–AES mode. XTS–AES mode uses a pair of keys for encryption. The first key is used to encrypt the sector address and to generate tweak values, and the second key is used to actually encrypt the data.

The XEX construction [3] is defined by

$$\tilde{E}_K^{N_{i_1}\ddot{i}_k}(M) = E(M \oplus \Delta) \oplus \Delta \tag{3}$$

where

$$\Delta = \alpha^{i_1}\alpha^{i_2}, \ldots, \alpha^{i_k}\mathbf{N}$$

and

$$\mathbf{N} = E_K(N)$$

According to [2], the XEX construction first computes a mask value T using Equation (6):

$$T = Enc(K2, s) \otimes \alpha^t \tag{4}$$

where the multiplication is in GF(2^n), (with n being blocksize of underlying algorithm), α is a primitive element of GF(2^n), and s is the value of the 128-bit tweak.

Given plaintext P and ciphertext C, encryption and decryption functions are produced by the following formulas [2]:

$$\begin{aligned} C &= Enc(K1, P \oplus T) \oplus T \\ P &= Dec(K1, C \oplus T) \oplus T \end{aligned} \tag{5}$$

In case of XTS-AES, logical sector address is used as tweak, which is modifying encryption process. Specification for the XTS-AES tweakable block cipher [2] introduces multiplication, which is defined by the following procedure:

Input: j is the power of α
byte array $a_0[k], k = 0, 1, \ldots, 15$;
Output: byte array $a_j[k], k = 0, 1, \ldots, 15$.

The output array is defined recursively by the following formulas where i is iterated from 0 to j:

$$a_{i+1}[0] \leftarrow (2(a_i[0] mod 128)) \oplus (135\lfloor a_i[15]/128 \rfloor)$$
$$a_{i+1}[k] \leftarrow (2(a_i[k] mod 128)) \oplus (\lfloor a_i[k-1]/128 \rfloor), \quad (6)$$
$$k = 1, \ldots, 15.$$

3. THE BASIC IDEA

As implied in [1] and [6], for functioning of the CFB, OFB, and CTR modes of operation, it is sufficient to implement forward cipher function. According to [9], mentioned modes of operation create a stream cipher from any block cipher. Generated keystream is xored with plaintext during encryption to create ciphertext and vice versa [32]. The fact that the described modes of operation need no decryption function implies that a keystream generator or even a cryptographic algorithm without decryption function could be used as an underlying algorithm.

Advanced encryption standard algorithm can use keylengths of 128, 192, and 256 bits. In case of the 256-bit protection an initial 256-bit long key is extended to required 240 bytes. Plaintext or an initialization vector is copied into a 4*x*4 byte matrix called State [33,34]. Then, State matrix is mixed during the initial round with bytes of the extended key. Thereafter, the buffer State is modified by SubBytes, ShiftRows, MixColumns, and AddRoundKey transformations during 14 rounds [35]. In case of a 128-bit encryption, mentioned transformations are used during 10 rounds.

Thanks to AES algorithms characteristics; as illustration, it is possible to create a new algorithm that implements only encryption function. Inputs for such an algorithm are plain text buffers 16 bytes long and a 256-bit key. This initial key is then divided into two parts 128-bit long. After that, plain text block is copied in two State buffers, which are encrypted with 128-bit encryption by the corresponding key parts. The resulting cipher text blocks are then merged using xor operation. If P1, P2,…,Pn are plaintext blocks, the following pseudo code represents ciphering of the j^{th} block:

Listing 2. ***Pseudo code of algorithm adjusted for parallel encryption***

$K1_{128}, K2_{128} = Split(K_{256})$
$O1_j = Ciph(P_j, K1_{j,128})$
$O2_j = Ciph(P_j, K2_{j,128})$
$C_j = O1_j \oplus O2_j$

whereby P_j is the j^{th} block of plaintext, Split is the function that splits initial key, $K1_{128}$, $K2_{128}$are the keys obtained by splitting initial 256-bit key, $O1_j$, $O2_j$ are the j^{th} output blocks obtained by ciphering with K1 and K2 keys, and C_j is the j^{th} block of ciphertext.

This algorithm designed for demonstration requires a 256-bit key for its operation. The initial key is then divided into two parts that are supplied to two encryption routines. Each plain text block is encrypted two times with 128-bit encryption. The obtained intermediate results are merged with xor operation to form eventual result. Such an algorithm that operates as key stream is well adjusted for parallelization as its procedure can be implemented with two parallel threads.

However, it is not possible to decrypt plaintext already encrypted using this algorithm. That is why it can be used as a key stream generator only. As mentioned previously, any block algorithm operates as a key stream when used with CFB, OFB, and CTR modes of operation. Block algorithms with CFB, OFB, and CTR modes of operation do not involve inverse block cipher function as this function is provided through the mode construction itself [1].

4. PARALLEL TWEAKABLE OFB MODE OF OPERATION USING XEX AND XE CONSTRUCTIONS

As pointed out, OFB mode of operation does not require implementation of inverse transformation (InvCipher) of an underlying block cipher algorithm. Thanks to this feature; it is possible to create a new algorithm that implements only encryption function. The number of rounds in 256 and 128 bit versions of AES algorithm is 14 and 10, respectively. As shown, it is possible to create a new algorithm by using two parallel encryption threads with 10 rounds each. In such design, 256-bit key is divided into two 128-bit parts. To improve its resistance, tweak similar to one that is shown in [2,3] is added to algorithm. The created key streams can be used for parallelization of its execution. The Listing (3) of pseudo code illustrates it as follows:

Listing 3. ***Illustration of tweakable parallel OFB mode with XEX construction***

OFB parallel encryption
$K1, K2 = Split(K_{256})$
First thread
$EIV' = Ciph(IV, K2)$
$F'_1 = EIV'$
$T'_1 = \alpha^1 \otimes EIV'$
$I'_1 = T'_1 \oplus F'_1$
$O'_1 = Ciph(I'_1, K1)$
$C'_1 = T'_1 \oplus O'_1$

DOI: 10.1002/sec

$T'_i = \alpha^i \otimes T'_{i-1}$ *For* $i = 2, \ldots, n;$
$F'_i = C'_{i-1}$ *For* $i = 2, \ldots, n;$
$I'_i = T'_i \oplus F'_i$ *For* $i = 2, \ldots, n;$
$O'_i = Ciph(I'_i, K1)$ *For* $i = 2, \ldots, n;$
$C'_i = T'_i \oplus O'_i$ *For* $i = 2, \ldots, n;$

Second thread
$EIV'' = Ciph(IV, K1)$
$F''_1 = EIV''$
$T''_1 = \alpha^1 \otimes EIV''$
$I''_1 = T''_1 \oplus F''_1$
$O''_1 = Ciph(I''_1, K2)$
$C''_1 = T''_1 \oplus O''_1$

$T''_i = \alpha^i \otimes T''_{i-1}$ *For* $i = 2, \ldots, n;$
$F''_i = C''_{i-1}$ *For* $i = 2, \ldots, n;$
$I''_i = T''_i \oplus F''_i$ *For* $i = 2, \ldots, n;$
$O''_i = Ciph(I''_i, K2)$ *For* $i = 2, \ldots, n;$
$C''_i = T''_i \oplus O''_i$ *For* $i = 2, \ldots, n;$

$C_i = P_i \oplus C'_i \oplus C''_i$ *For* $i = 1, \ldots, n-1;$

Final block
$C^*_N = P^*_N \oplus MSB_U(C^{*\prime}_N \oplus C^{*\prime\prime}_N)$

OFB parallel decryption
K1, K2 = Split(K_{256})
First thread
$EIV' = Ciph(IV, K2)$
$F'_1 = EIV'$
$T'_1 = \alpha^1 \otimes EIV'$
$I'_1 = T'_1 \oplus F'_1$
$O'_1 = Ciph(I'_1, K1)$
$C'_1 = T'_1 \oplus O'_1$

$T'_i = \alpha^i \otimes T'_{i-1}$ *For* $i = 2, \ldots, n;$
$F'_i = C'_{i-1}$ *For* $i = 2, \ldots, n;$
$I'_i = T'_i \oplus F'_i$ *For* $i = 2, \ldots, n;$
$O'_i = Ciph(I'_i, K1)$ *For* $i = 2, \ldots, n;$
$C'_i = T'_i \oplus O'_i$ *For* $i = 2, \ldots, n;$

Second thread
$EIV'' = Ciph(IV, K1)$
$F''_1 = EIV''$
$T''_1 = \alpha^1 \otimes EIV''$
$I''_1 = T''_1 \oplus F''_1$
$O''_1 = Ciph(I''_1, K2)$
$C''_1 = T''_1 \oplus O''_1$

$T''_i = \alpha^i \otimes T''_{i-1}$ *For* $i = 2, \ldots, n;$
$F''_i = C''_{i-1}$ *For* $i = 2, \ldots, n;$
$I''_i = T''_i \oplus F''_i$ *For* $i = 2, \ldots, n;$
$O''_i = Ciph(I''_i, K2)$ *For* $i = 2, \ldots, n;$
$C''_i = T''_i \oplus O''_i$ *For* $i = 2, \ldots, n;$

$P_i = C_i \oplus C'_i \oplus C''_i$ *For* $i = 1, \ldots, n-1;$

Final block
$P^*_N = C^*_N \oplus MSB_U\left(C^{*\prime}_N \oplus C^{*\prime\prime}_N\right)$

where IV is the initialization vector; Split is the function that splits initial key; $K1, K2$ *is the 128 bit long key parts obtained with splitting initial key;* T'_i, T''_i *are the* i^{th} *tweaks;* O'_i, O''_i *are the intermediate results of parallel threads ciphering;* C'_i, C''_i *are the intermediate results mixed with tweak;* F'_i, F''_i *are the* i^{th} *fed back blocks (inputs);* I'_i, I''_i *are the* i^{th} *inputs for encryption mixed with tweak;* P_i *is the* i^{th} *plaintext block;* C_i *is the* i^{th} *block of ciphertext;* P^*_N *is the final plaintext block and can be a part of a block too;* C^*_N *is the final ciphertext block and can be a part of a block too; and* $MSB_M(X)$ *is the most significant M bits of bit stream X.*

It could be possible to realize parallel OFB mode by using algorithm shown in Listing (2), but such a mode would be susceptible to meet-in-the-middle attack. To strengthen the algorithm, an idea from XEX mode is used. Proposed parallel mode uses a pair of 128-bit keys for encryption with two threads. Every thread have primary 128-bit key, which is used for data encryption, and secondary 128-bit key, which is used to create tweak by encrypting IV. In the first thread, the secondary key is K2, which is used to create tweak, and the primary key is K1, which is used for data encryption process. In the second thread, the primary key is K2, and it is used for data encryption while the secondary key K1 is used to create tweak.

In the illustrated embodiment, unlike XTS-AES, IV is used instead the logical sector address both to form a tweak and for the functioning of the OFB mode of operation. As presented in Listing (3), in both threads IV is encrypted by using the secondary key forming intermediate results EIV' and EIV". That intermediate results are then repeatedly transformed by using XTS-AES multiplication. Proposed solution uses the same method for multiplication as the XTS-AES, but instead of using sector address it uses encrypted IV (EIV' and EIV") while sequentially increases the value of j.

As illustrated in Listing (3) and Figure A2, both threads are using procedure similar to XTS–AES and XEX procedure. In both threads, the first input for main encryption procedure is formed by xoring encrypted IV's (F'_1 and F''_1) with corresponding tweaks (T'_1 and T''_1) created by shown multiplication procedure. All other inputs for main encryption procedure are created by merging output from encryption of previous block with corresponding tweaks (T'_i and T''_i).

Intermediate results (O'_i and O''_i) from main encryption procedures in both threads are formed by using corresponding primary keys. In both threads, the final results of the single block encryption are created by merging intermediate results (O'_i and O''_i) with corresponding tweaks (T'_i and T''_i).

The proposed mode acts similar to XTS–AES mode of operation—it support encryption of data buffers whose lengths are not integral multiple of the AES block size (128 bits). If final block has less then 128 bits, final partial block is processed in conjunction with the previous one by using ciphertext-stealing as described in [2].

Construction of the proposed algorithm is such that the output blocks from both threads must be generated and then xored with each other and with corresponding plaintext block to produce cyphertext block. If the algorithm was designed in such a way that threads are waiting each other in order to merge intermediate results after each block, according to [23] and [24], that would have a negative impact on performance. To avoid this, algorithm is implemented in such a way that the merging step is performed after treatment of a number of blocks. By using the procedure earlier, the results obtained are up to 10% faster in comparison with serial OFB mode of operation with AES algorithm.

As Rogaway [3] as well as Liskov *et al*. [12] stated, XEX construction has good security against chosen-ciphertext attack, while for chosen-plaintext attack security, one can omit the outer xor. Previously described tweakable parallel OFB mode with XEX construction can be easily transformed into cipher with XE construction, by omitting outer XOR. By using XE construction and two threads with 128-bit AES encryption, the results are up to 19% faster compared with serial 256-bit AES encryption in OFB mode of operation. The Listing (4) of pseudo code and Figure A3 illustrates tweakable parallel OFB mode of operation with XE construction.

Listing 4. *Illustration of tweakable parallel OFB mode with XE construction*

OFB parallel encryption
$K1,\ K2 = Split(K_{256})$

First thread
$EIV' = Ciph(IV, K2)$
$F'_1 = EIV'$
$T'_1 = \alpha^1 \otimes EIV'$
$I'_1 = T'_1 \oplus F'_1$
$O'_1 = Ciph(I'_1, K1)$
$C'_1 = O'_1$

$T'_i = \alpha^i \otimes T'_{i-1}$ *For* $i = 2, \ldots, n$;
$F'_i = O'_{i-1}$ *For* $i = 2, \ldots, n$;
$I'_i = T'_i \oplus F'_i$ *For* $i = 2, \ldots, n$;
$O'_i = Ciph(I'_i, K1)$ *For* $i = 2, \ldots, n$;
$C'_i = O'_i$

Second thread
$EIV'' = Ciph(IV, K1)$
$F''_1 = EIV''$
$T''_1 = \alpha^1 \otimes EIV''$
$I''_1 = T''_1 \oplus F''_1$
$O''_1 = Ciph(I''_1, K2)$
$C''_1 = O''_1$

$T''_i = \alpha^i \otimes T''_{i-1}$ *For* $i = 2, \ldots, n$;
$F''_i = O''_{i-1}$ *For* $i = 2, \ldots, n$;
$I''_i = T''_i \oplus F''_i$ *For* $i = 2, \ldots, n$;
$O''_i = Ciph(I''_i, K2)$ *For* $i = 2, \ldots, n$;
$C''_i = O''_i$

$C_i = P_i \oplus C'_i \oplus C"_i$ *For* $i = 1, \ldots, n-1$;

Final block
$C^*_N = P^*_N \oplus MSB_U(C^{*'}_N \oplus C^{*"}_N)$

OFB parallel decryption
$K1,\ K2 = Split(K_{256})$

First thread
$EIV' = Ciph(IV, K2)$
$F'_1 = EIV'$
$T'_1 = \alpha^1 \otimes EIV'$
$I'_1 = T'_1 \oplus F'_1$
$O'_1 = Ciph(I'_1, K1)$
$C'_1 = O'_1$

$T'_i = \alpha^i \otimes T'_{i-1}$ *For* $i = 2, \ldots, n$;
$F'_i = O'_{i-1}$ *For* $i = 2, \ldots, n$;
$I'_i = T'_i \oplus F'_i$ *For* $i = 2, \ldots, n$;
$O'_i = Ciph(I'_i, K1)$ *For* $i = 2, \ldots, n$;
$C'_i = O'_i$

Second thread
$EIV'' = Ciph(IV, K1)$
$F''_1 = EIV''$
$T''_1 = \alpha^1 \otimes EIV''$
$I''_1 = T''_1 \oplus F''_1$
$O''_1 = Ciph(I''_1, K2)$
$C''_1 = O''_1$

$T''_i = \alpha^i \otimes T''_{i-1}$ *For* $i = 2, \ldots, n$;
$F''_i = O''_{i-1}$ *For* $i = 2, \ldots, n$;
$I''_i = T''_i \oplus F''_i$ *For* $i = 2, \ldots, n$;
$O''_i = Ciph(I''_i, K2)$ *For* $i = 2, \ldots, n$;
$C''_i = O''_i$

$P_i = C_i \oplus C'_i \oplus C''_i$ *For* $i = 1, \ldots, n-1$;

Final block
$P^*_N = C^*_N \oplus MSB_U(C^{*\prime}_N \oplus C^{*\prime\prime}_N)$

where IV is the initialization vector, Split is the function that splits initial key, $K1, K2$ *are the 128-bit long key parts obtained with splitting initial key;* T'_i, T''_i *are the* i^{th} *tweaks;* O'_i, O''_i *are the intermediate results of parallel threads ciphering;* C'_i, C''_i *are the intermediate results mixed with tweak;* F'_i, F''_i *are the* i^{th} *fed back blocks (inputs);* I'_i, I''_i *are the* i^{th} *inputs for encryption mixed with tweak;* P_i *is the* i^{th} *plaintext block;* C_i *is the* i^{th} *block of ciphertext;* P^*_N *is the final plaintext block and can be a part of a block too;* C^*_N *is the final ciphertext*

DOI: 10.1002/sec

block and can be a part of a block too; and $MSB_M(X)$ is the most significant M bits of bit stream X.

5. IMPLEMENTATION DETAILS AND ANALYSIS

Listings (3) and (4) present a new modes of operation based on XTS–AES and OFB mode that uses two parallel threads with 128-bit encryption each in order to outperform execution of 256-bit AES algorithm in OFB mode of operation. Both threads form their inputs for encryption by merging 16 bytes of tweak with fed back result of previous block encryption. As shown in Listing (3) and Figure A3, intermediate ciphertexts in both threads are generated by xoring corresponding AES encryption result with corresponding tweak, thus forming XEX construction.

For the operation of this algorithm, it is not necessary to ensure that both threads have completed their execution in order to continue processing of the next block. If a number of plaintext blocks can be loaded in some input array, then each individual thread may continue its execution independently. As encryption of one block is completed, each thread stores the result of its work in its own array, which is designed for that purpose. In these arrays, it is possible to store the same number of blocks as in the input array, and their purpose is to keep the intermediate ciphertexts. After keystream generation is completed, the input array is merged with two output arrays produced by both threads to eventually create ciphertext (plaintext). By designing the implementation in this way, the moment of synchronization is postponed, thus reducing the time spent for threads waiting for each other.

A Java application was intentionally developed for practical probe of the introduced concept potential. This application implements a standard AES algorithm [33] without AES-NI set of instructions. Implementation of AES encryption routine with 256-bit key and 14 rounds was used in serial OFB mode of operation using one thread. For parallel OFB mode of operation with XTS-like tweak, the same routine with 128-bit key and 10 rounds is used in two threads. Threads involved in the tests were designed, started, and stopped in the same way. In both cases, time taken by encryption is measured without the time spent in communication with hard disk and in class initialization. Measured result includes the time taken for threads creating, starting and executing, tweak creation, and merging arrays by using xor operation and time spent in AES encryption routine. In short, we measured elapsed time from the start of the first thread until the last one completed [36].

The measuring was carried out on two test platforms. The first one was laptop with Core i7-4700MQ processor having four physical and eight logical cores with 4 GB DDR3-1600 RAM, with installed 64-bit Windows 8.1, Java(TM) SE Runtime Environment (build 1.8.0_25-b18), and Java HotSpot(TM) 64-Bit Server VM (build 25.25-b02, mixed mode). In the rest of the text, this platform will be referred to as i74700. The second one was laptop with Core i7-2630QM processor having four physical and eight logical cores with 6 GB DDR3-1333 RAM, with installed 64-bit Windows 7 Ultimate OS and same Java engine. In the remaining text, the latter test platform will be referred to as i72630.

The presented application is used to measure elapsed time spent for encryption of 128 MB file. To explore influence of thread creation and synchronization on performance, encryption is performed by using input arrays (input buffers) of different sizes.

In the implementation of the concept, we used a Java cached thread pool mechanism (Executor class, newCachedThreadPool method) that executes several threads in parallel, and then waits on completion of all threads running. Cached thread pool creates a number of threads required in parallel execution, and the existing threads can be reused in new tasks.

Georges, Eeckhout, and Buytaert [19,20] state that JVM (Java virtual machine) is rather challenging to benchmarking because Java performance is affected in various complex ways by the application and its input, as well as by the virtual machine (JIT optimizer, garbage collector, thread scheduler, etc.). That is why we carried out all tests by using the rigorous replay compilation [20] to measure time needed for file encryption. The tests were conducted using various length input arrays and one VM call for each encryption cycle. This way, file was processed 10 times for each input array size, and the final result is obtained by using arithmetic mean.

The measuring on PC platform was carried out under circumstances in which the number of running OS processes was minimal. The achieved parallel execution results were fluctuating with the change of the previously mentioned input buffer size. The test results comparing serial OFB mode, referred as 1x256 OFB in later text, with parallel OFB mode of operation with XEX tweak, referred as 2x128 XEX in later text, are shown in Tables B1 and B2.

Depending on the input array size, the results obtained in the tests on i74700 platform (Table B1) show that it is possible to achieve acceleration up to 10.14%. However, the illustrated results show significant performance degradation with further input array size reduction.

The measured times indicate that worse performance is obtained whenever shorter input buffers are used. At the end of each fragment processing, threads will potentially have to wait each other in order to finish their execution. If shorter input array is used, then same file must be processed by using greater number of fragments, which leads to performance penalties. In contrast, when using a longer input array, the better results are achieved.

The results obtained on the second i72630 platform (Table B2) do not show any improvements. Moreover, the results of measurements on that test platform indicate that there has been a significant performance degradation varying from 2.15% to 23.04%.

As pointed out in Section 2, degradation of multithreaded Java application performance could be generated by lock contentions and low performance of memory systems. Additionally, the garbage collection mechanism and thread creation could be important factors of performance degradation.

The second test platform (i72630) had RAM slower than the one built in the first platform (i74700). In order to check impact of memory speed combined with additional XOR operation on algorithm performance, another series of measurements is performed with slightly modified implementation with omitted outer XOR, as described in Listing (4). The Tables B3 and B4 make comparison between the serial 1x128 OFB mode and parallel tweakable OFB mode of operation with XE construction, referred as 2x128 XE in later text.

In parallel 2x128 XEX OFB mode, both threads are doing extra work compared with the serial OFB mode. The first round in both threads consists of extra encryption of one block, then an additional XTS multiplication, additional XOR operation on tweak first with 16 input bytes and then additional XOR with the output 16 bytes. In every round except the first, each tread performs an additional XTS multiplication, additional XOR operation on tweak with 16 input bytes and additional XOR operation with the output 16 bytes. However, parallel 2x128 XEX OFB mode improves performance under certain conditions while processing simultaneously one data block for 10 rounds with two threads, unlike serial 1x128 OFB mode that spends 14 rounds to encrypt each block.

As illustrated in Tables B3 and B4, by using XE construction, a significant acceleration is achieved on both test platforms. At the first test platform (i74700), it is obtained maximum improvement of 19.35% compared with a serial implementation. At the second test platform (i72630), it is obtained improvement of nearly 12.98% compared with a serial OFB mode. When shortest input array included in tests (8 KB) is used in combination with XE construction, there still is degradation of performance but slightly smaller then in case of XEX construction.

Cipher block chaining , CFB, and OFB are known as feedback modes, while ECB and CTR are known as no feedback modes [37]. Bearing in mind the constraints of the proposed method(s), and our initial goal to improve performance of the standard OFB mode, just for performance comparision with fully parallelizable CTR mode, experimental results of the parallelized CTR mode execution time are shown in the Tables B1, B2, B3, and B4. As for the experiment, CTR mode is parallelized so that it uses two threads to encrypt two plaintext fragments at the same time. Thanks to this; the parallelized CTR mode attains far better results than the proposed solutions. Having compared the code-size, the proposed solution is 5 kB longer then the parallelized CTR using two threads. The size of a JAVA class file that implements the proposed modes is 54 kB, while the size of a JAVA class file implementing a parallelized CTR solution is 49 kB.

One of the OFB mode characteristics is that the transmission faults are not propagated. This property is very useful for satellite communications where the transmission channels are very noisy. Hence, the advantage of OFB over the CBC and CFB modes lies in fact that any bit errors that might occur inside cipher data are not propagated to affect the decryption of subsequent blocks [27–29]. Because of that, bit flipping appears as a security flaw of OFB mode on the one hand, and on the other hand, it is an important mechanism of error correction in satellite communications.

However, it is worth mentioning that there is no evidence in the available literature that OFB mode has ever been parallelized.

6. CONCLUSION

In this paper, we examine possibilities of achieving performance improvement of the standard OFB mode by using new parallel tweakable OFB modes of operation. The conducted research exploits the fact that the design of some cryptographic modes of operation is very alike stream ciphers. OFB mode of operation produces a keystream that is xored with plaintext in order to generate ciphertext. Another feature of this mode of operation is that it does not require existence of inverse blockcipher function for its operation. These facts were used to design parallel tweakable OFB mode of operation in two variants. In creation of these new modes, XEX and XE constructions and XTS-AES multiplication are used. According to this proposal, two threads working in parallel utilize AES encryption in tweakable OFB mode of operation. We use postponed thread synchronization to improve performance of proposed modes. The results obtained show that it is possible to achieve performance accelerations of the standard OFB mode up to nearly 20% under different test platforms.

The presented research, implementation, and experimental results point out several facts concerning the possibility of parallelisation of any symmetric block algorithm. Performance is greatly depended on issues such as thread creation, synchronization, and specific hardware features like memory speed. As we have shown, by postponing the moment of thread synchronization and by using the faster main memory great accelerations can be achieved in relation to the standard OFB mode, which is extremely valuable for satellite communications, where transmission channel is noisy.

Further, course of the research into parallelisation involving AES algorithm in OFB mode of operation might be directed toward AES-NI instruction set utilization. Another direction of the research might be analysis of some advanced programming techniques such as OpenMP and Message Passing Interface possibilities of utilization, or exploring advanced parallel hardware architectures as field-programmable gate arrays, vector processors, or application-specific integrated circuits.

DOI: 10.1002/sec

ACKNOWLEDGEMENT

The work presented here was partially supported by the Serbian Ministry of Science and Technological development (under project of Multimodal Biometry in Identity Management, contract no TR-32013).

REFERENCES

1. Chakraborty D, Rodriguez-Henriquez F. Block cipher modes of operation from a hardware implementation perspective. In *Cryptographic Engineering*, Koc CK (ed). Springer Science+Business Media, LLC: New York, USA, 2009; 321–363.
2. IEEE Std. 1619-2007. 2008, Cryptographic protection of data on block-oriented storage devices IEEE.
3. Rogaway P. Efficient instantiations of tweakable blockciphers and refinements to mode OCB and PMAC. *Proceedings in Asiacrypt 2004*, Jeju Island, Korea, Springer, Berlin Heidelberg New York, 2004; 16–31.
4. Katz J, Lindell Y. *Introduction to modern cryptography*. Taylor & Francis group, Boca Raton, 2008; 72–103.
5. Rogaway P. Evaluation of some blockcipher modes of operation. *Technical Report*, Cryptography Research and Evaluation Committees, CRYPTREC '10, Government of Japan, 2011.
6. Dworkin M. Recommendation for block cipher modes of operation. *Technical Report*, NIST Special Publication 800-38A. NIST, Gaithersburg, USA, 2001.
7. Ferguson N, Schneider B, Kohno T. *Cryptography Engineering: Design Principles and Practical Applications*. Wiley Publishing Inc: Indianapolis, USA, 2010. 63–77.
8. Dent AW, Mitchell CJ. *Users Guide to Cryptography and Standards*. Artech House: Boston, London, 2005. 71–87.
9. Menezes AJ, van Oorschot PC, Vanstone SA. *Handbook of Applied Cryptography*. CRC Press, Inc.: Boca Raton, 1996.
10. Chakraborty D, Sarkar P. A general construction of tweakable block ciphers and different modes of operations. *Inscrypt'06 Proceedings of the Second SKLOIS conference on Information Security and Cryptology*, Springer-Verlag Berlin, Heidelberg, 2006; 88–102.
11. Liskov M, Rivest R, Wagner D. Tweakable block ciphers. *Proceedings Advances in Cryptology CRYPTO 02*, vol. 2442 of Lecture Notes in Computer Science, Santa Barbara, USA, Springer Verlag, Berlin Heidelberg, 2002; 31–46.
12. Dworkin M. Recommendation for block cipher modes of operation: the XTS-AES mode for confidentiality on storage devices. *Technical Report*, NIST Special Publication 800-38E, NIST, Gaithersburg, USA, 2010.
13. Lipmaa H, Rogaway P, Wagner D. Comments to NIST concerning AES-modes of operations: CTR-mode encryption. *Symmetric Key Block Cipher Modes of Operation Workshop*, Baltimore, Maryland, USA, 2000. (Available from: http://csrc.nist.gov/groups/ST/toolkit/BCM/documents/proposedmodes/ctr/ctr-spec.pdf) [Accessed on 5 January 2015].
14. Tran NP, Lee M, Hong S, Lee SJ. Parallel execution of AES-CTR algorithm using extended block size. *Proceedings in Computational Science and Engineering (CSE)*, IEEE Computer Society Washington, DC, 2011; 191–198.
15. Di Biagio A, Barenghi A, Agosta G, Pelosi G. Design of a parallel AES for graphics hardware using the CUDA framework. *Proceedings in IEEE International Parallel & Distributed Processing Symposium IPDPS'09*, IEEE Computer Society Washington, DC, 2009.
16. Roch JL, Jacquin L, Ali MA, Roca V. Parallel arithmetic encryption for high-bandwidth communications on multicore/GPGPU platforms. *Proceedings of the 4th International Workshop on Parallel and Symbolic Computation (PASCO '10)*, Grenoble, France, 2010; 73–79.
17. Berk Guder C. 2009. *AES on CUDA*. (Available from: http://github.com/cbguder/aes-on-cuda) [Accessed on 5 January 2015].
18. Zola WMN, De Bona LCE. Parallel speculative encryption of multiple AES contexts on GPUs. *Proceedings of the Innovative Parallel Computing (InPar 2012)*, San Jose, CA, 2012; 1–9.
19. Georges A, Eeckhout L, Buytaert D. Statistically rigorous java performance evaluation. *Proceedings of the 22rd ACM SIGPLAN Conference on Object Oriented Programming Systems Languages and Applications (OOPSLA '07)*, Montreal, Canada, 2007; 57–76.
20. Georges A, Eeckhout L, Buytaert D. Java performance evaluation through rigorous replay compilation. *Proceedings of the 23rd ACM SIGPLAN Conference on Object Oriented Programming Systems Languages and Applications (OOPSLA '08)*, Nashville, TN, USA, 2008; 367–384.
21. Boyer B. *Robust Java benchmarking, Part 1: Statistics and Solutions, Introducing a Ready-to-run Software Benchmarking Framework*. IBM developerWorks: New York, USA, 2008. (Available from: http://www.ibm.com/developerworks/library/j-benchmark1) [Accessed on 5 January 2015].
22. Krieger CD, Strout MM. Performance evaluation of an irregular application parallelized in Java. *Proceedings of the 39th International Conference on*

Parallel Processing Workshops (ICPPW), San Diego, CA, USA, 2010, 2010; 227–235. DOI: 10.1109/ICPPW.2010.40.

23. Chen KY, Chang JM, Hou TW. Multithreading in Java: performance and scalability on multicore systems. *IEEE Transactions on Computers* 2011; **60**(11): 1521–1534.
24. Tarvo A, Reiss SP. Automated analysis of multithreaded programs for performance modeling. *Proceedings of the 29th IEEE/ACM International Conference on Automated Software Engineering (ASE 2014)*, Västerås, Sweden, 2014; 7–18.
25. Zhang L, Krintz C, Nagpurkar P. Language and virtual machine support for efficient fine-grained futures in Java. *Proceedings of the 22nd International Conference on Parallel Architectures and Compilation Techniques*, Brasov, Romania, 2007; 130–139.
26. Gu Y, Lee BS, Cai W. Evaluation of Java thread performance on two different multithreaded kernels. *ACM SIGOPS Operating Systems Review*, ACM New York, NY, USA, 1999; 34–46.
27. Tirro S. *Satellite Communication Systems Design*. Springer Science+Business Media LLC: New York, 1993; 101.
28. Cheltha JNC, Velayutham R. A novel error-tolerant method in AES for satellite images. *IEEE International Conference on Emerging Trends in Electrical and Computer Technology (ICETECT)*, Tamil Nadu, 2011; 937–940.
29. Ramadevi G, Sujatha R. Robust code based fault tolerant architecture using OFB mode for onboard EO satellites. *International Journal of Soft Computing and Engineering (IJSCE)* 2013; **3**(2): 146–149.
30. ISO/IEC Std. 10116:2006. *Information technology - security techniques - modes of operation for an nbit block cipher*, 2006. International Organization for Standardization.
31. FIPS PUB 81. *DES Modes of Operation*. Federal Information Processing Standards Publication 81, 1980.
32. Zhang X, Heys HM, Li C. Energy efficiency of encryption schemes applied to wireless sensor networks. *Security and Communication Networks* 2012; **5** (7): 789–808.
33. FIPS PUB 197. *Specification for the advanced encryption standard (AES)*. Federal Information Processing Standards Publication 197, 2001.
34. Khan M, Azam NA. Right translated AES gray S-boxes. *Security and Communication Networks* 2015; **8**(9): 1627–1635.
35. Cazorla M, Gourgeon S, Marquet K, Minier M. Survey and benchmark of lightweight block ciphers for MSP430 16-bit microcontroller. *Security and Communication Networks* 2015; **8**(18): 3564–3579.
36. Bader DA, Moret BME, Sanders P. *Algorithm Engineering for Parallel Computation*, Experimental algorithmics. Springer-Verlag New York: NY, USA, 2002.
37. Gaj K, Chodowiec P. Fast implementation and fair comparison of the final candidates for advanced encryption standard using field programmable gate arrays. *Proceedings of the 2001 Conference on Topics in Cryptology: The Cryptographer's Track at RSA*, San Francisco, USA, 2001.

APPENDIX A: FIGURES

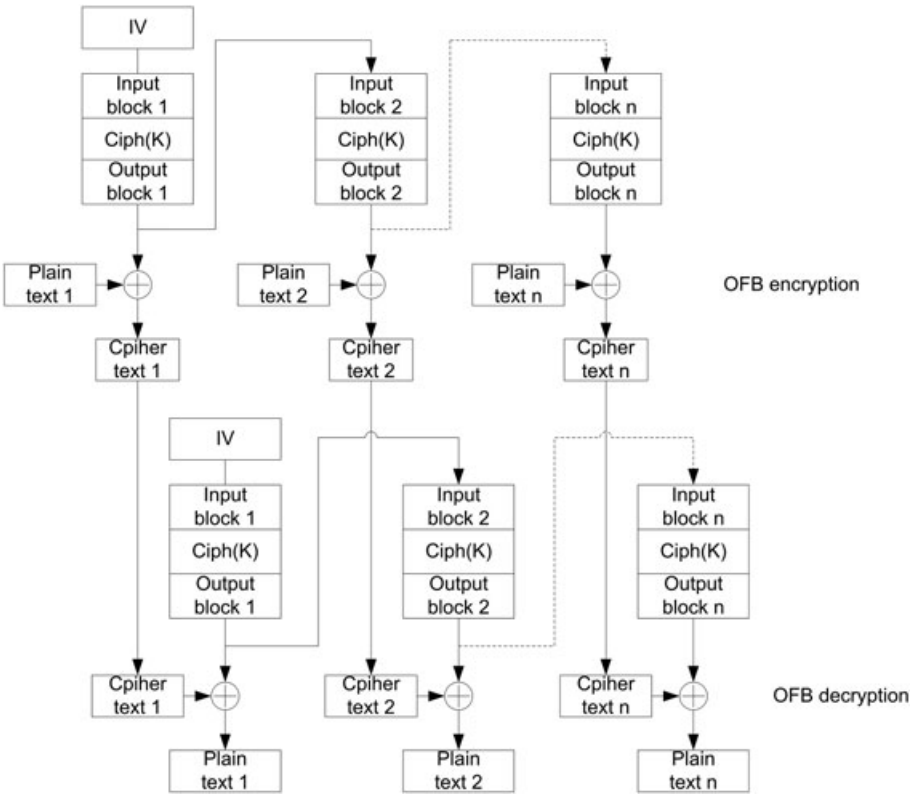

Figure A1. Output feedback mode of operation.

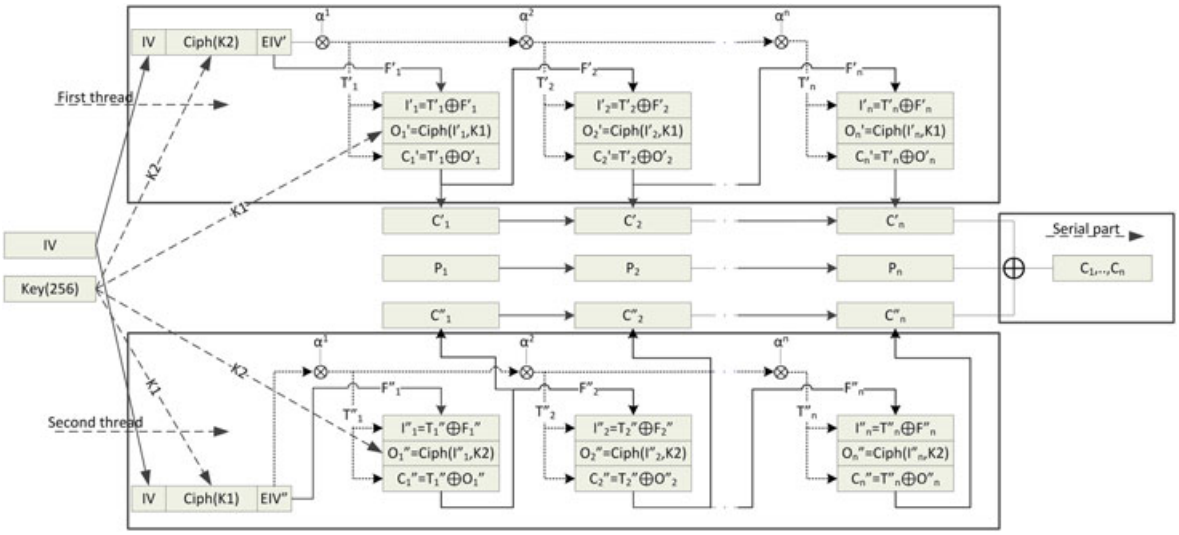

Figure A2. Tweakable parallel output feedback mode with xor encrypt xor construction.

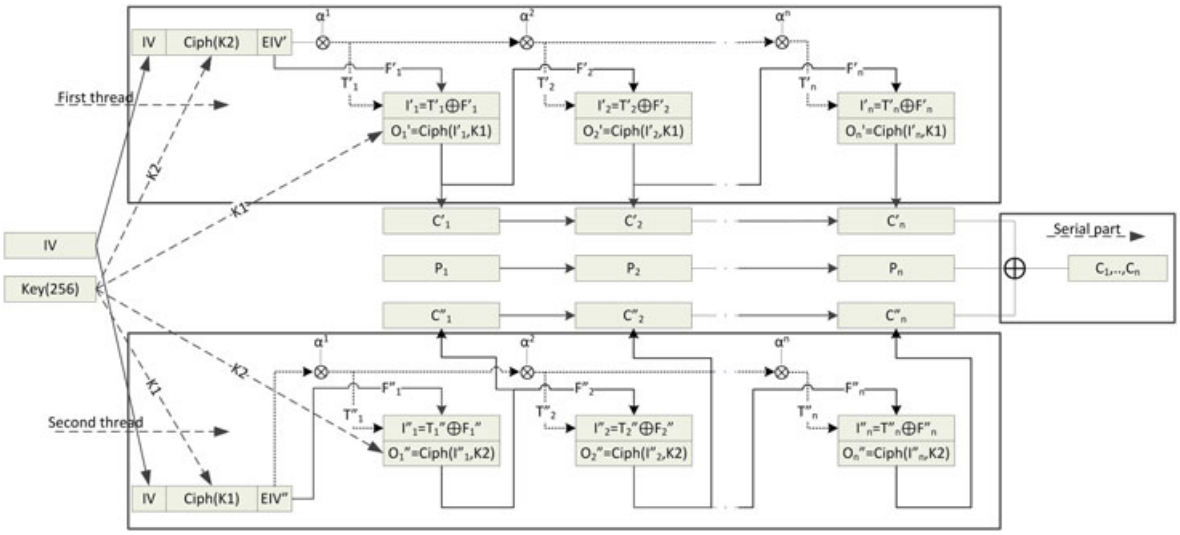

Figure A3. Tweakable parallel Output feedback mode with XE construction.

APPENDIX B: TABLES

Table BI. Comparison: serial OFB encryption of 128 MB file versus tweakable XEX OFB, parallel tweakable XEX OFB and parallel CTR ON I74700 platform.

Mode	Input array size (Bytes)						
	512000	64000	20000	16000	14000	12000	8000
1x 256 OFB (ms)	1653.2	2380	3093.6	3274.2	3421.83	3530	4441
2x 128 XEX (ms)	1552.6	2160.8	2972.8	3250.0	3408.2	3665.0	4715.2
spedup	1.065	1.101	1.041	1.007	1.004	0.963	0.942
%	6.48	**10.14**	4.06	0.74	0.40	–3.68	–5.82
1x 128 XEX (ms)	1290.6	1948.4	2603.9	2667.7	2846	3141.2	3804.3
1x 256 XEX (ms)	1611.1	2341.4	3047.5	3325.5	3472.5	3673.3	4747
2x 256 CTR (ms)	858.2	1274.2	1730.2	1919	2074.1	2183.6	2677.4

OFB, output feedback; XEX, xor Encrypt xor.

Table BII. Comparison: serial OFB encryption of 128 MB file versus tweakable XEX OFB, parallel tweakable XEX OFB and parallel CTR ON I72630 platform.

Mode	Input array size (Bytes)						
	512000	64000	20000	16000	14000	12000	8000
1x 256 OFB (ms)	4128.4	4715.8	5510.3	5726.8	5797	5835.5	6439.5
2x 128 XEX (ms)	4219.0	4895.0	6445.4	6794.0	7077.8	7275.0	8367.0
spedup	0.979	0.963	0.855	0.843	0.819	0.802	0.770
%	–2.15	–3.66	–14.51	–15.71	–18.10	–19.79	–23.04
1x 128 XEX (ms)	3159.2	3741.4	4611.7	4811.9	4826.8	4980	5699.8
1x 256 XEX (ms)	4041.2	4749.9	5685	5908.2	5997.8	6193.8	6699
2x 256 CTR (ms)	967.2	1512.2	1976.5	2281.3	2397.8	2646.6	3360.6

OFB, output feedback; XEX, xor Encrypt xor.

Table BIII. Comparison: serial OFB encryption of 128 MB file versus tweakable XE OFB, parallel tweakable XE OFB and parallel CTR ON I74700 platform.

Mode	Input array size (Bytes)						
	512000	64000	20000	16000	14000	12000	8000
1x 256 OFB (ms)	1653.2	2380	3093.6	3274.2	3421.83	3530	4441
2x 128 XE (ms)	1458	1994.2	2850.8	3141.6	3347.2	3673.8	4667.8
spedup	1.134	1.193	1.085	1.042	1.022	0.961	0.951
%	13.39	**19.35**	8.52	4.22	2.23	–3.91	–4.86
1x 128 XE (ms)	1268.6	1951.8	2480.3	2778.7	2840.6	3068.7	3856.8
1x 256 XE (ms)	1872.1	2345.9	3083.6	3213.2	3440.4	3683.5	4678.6
2x 256 CTR (ms)	858.2	1274.2	1730.2	1919	2074.1	2183.6	2677.4

OFB, output feedback.

Table BIV. Comparison: serial OFB encryption of 128 MB file versus tweakable XE OFB, parallel tweakable XE OFB and parallel CTR ON I72630 platform.

Mode	Input array size (Bytes)						
	512000	64000	20000	16000	14000	12000	8000
1x 256 OFB (ms)	4128.4	4715.8	5510.33	5726.8	5797	5835.5	6439.5
2x 128 XE (ms)	3654.2	4290	5632	5960.4	6198.8	6537.6	7520.4
spedup	1.130	1.099	0.978	0.961	0.935	0.893	0.856
%	**12.98**	9.93	–2.16	–3.92	–6.48	–10.74	–14.37
1x 128 XE	3107.6	3787.6	4745.2	4881.2	4958.3	5120.6	5672.4
1x 256 XE	4058.5	4771.8	5644.1	6033	6047.1	6115.5	6691
2x 256 CTR	967.2	1512.2	1976.5	2281.3	2397.8	2646.6	3360.6

OFB, output feedback.