A New Narrow-Block Mode of Operation for Disk Encompression with Tweaked Block Chaining

Debasis Gountia[1] and Dipanwita Roy Chowdhury[2]
(Corresponding author: Debasis Gountia)

[1]Department of Computer Science & Application, College of Engineering & Technology, Bhubaneshwar, India
[2]Department of Computer Science & Engineering, Indian Institute of Technology, Kharagpur, India
dgountia@gmail.com, drc@cse.iitkgp.ernet.in

Abstract: In this paper, a new Disk Encompression (i.e., encryption with compression) with Tweaked Block Chaining mode (DETBC) has been proposed. DETBC is a modified of XTS i.e., Xor-Encrypt-Xor based Tweaked Code Book mode with CipherText Stealing. Unlike XTS, DETBC is faster, memory saving and is better resistant to the attacks. DETBC is characterized by its high throughput compared to the current solutions and improve its diffusion properties.

Keywords: block ciphers, disk encryption, Galois Field multiplier GF (2^{128}), tweakable block ciphers.

1. Introduction

Data encryption has been used for individual precious documents for the security purpose in the past. With the advent of more powerful desktop processors in the last decade, the data throughput of ciphers surpassed that of hard disks. Hence, encryption is no longer a bottle neck and regular users become more interested in the topic of hard disk encryption.

In today's computing environment, there are many threats to the confidentiality of information stored on computers and other devices like USB or external hard drive. Device loss or theft, Malware which give unauthorized access are common threat against end user devices. To prevent the disclosure of sensitive data, the data needs to be secured. Disk encryption is usually used to protect the data on the disk by encrypting it. The whole disk is encrypted with a single/multiple key(s) and encryption/decryption are done on the fly, without user interference. The encryption is on the sector level, that means each sector should be encrypted separately.

There are so many block ciphers dedicated to this task like Bear, Lion, Beast and Mercy [5, 5, 12, 16]. Bear, Lion and Beast are considered to be slow, as they process the data through multiple passes and Mercy was broken in [20]. The current available narrow-block modes of operations that offer error propagation are subjected to manipulation attacks. A need for a new secure and fast mode of operation with less memory consumption, that offers error propagation, has demanded.

In this paper, we propose a new narrow-block disk encryption mode of operation with compression. We decided to build the Tweaked Block Chaining (TBC) mode using Xor-Encrypt-Xor (XEX) [23] to inherit from its security and high performance and use CBC like operations to gain the error propagation property. This design is XEX-based TBC with CipherText Stealing (CTS) rather than Tweaked Code Book mode (TCB) as in case of XTS (XEX-based TCB with CTS). This model includes a Galois Field multiplier GF (2^{128}) that can operate in any common field representations. This allows very efficient processing of consecutive blocks in a sector. To handle messages whose length is greater than 128-bit but not a multiple of 128-bit, a variant of CipherText Stealing will be used for tweaked block chaining. We named this mode Disk Encompression with Tweaked Block Chaining (DETBC).

In section 2, we present Encryption with compression, and the constraints facing in the disk encryption applications. In section 3, we present tweak calculation, efficient multiplication, and exponential. Section 4 describes the implementation of our proposed scheme. Section 5 shows the performance analysis of narrow-block modes of operations that offer error propagation. Finally, section 6 concludes the work with presenting open problem.

2. Disk Encryption

Hard disk encryption is usually used to encrypt all the data on the disk. The whole hard disk is encrypted with a single/multiple key(s) and encryption/ decryption are done on the fly, without user interference. The encryption is on the sector level that means each sector should be encrypted separately.

2.1 Encryption with Compression

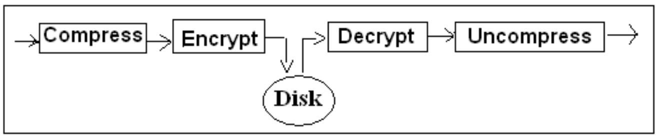

Figure 1. Steps for Disk encryption scheme.

Using a data compression algorithm together with an encryption algorithm makes sense for two reasons:
1. Cryptanalysis relies on exploiting redundancies in the plain text; compressing a file before encryption reduces redundancies.
2. Encryption is time-consuming; compressing a file before encryption speeds up the entire process.

In this work, we use the"LZW 15-bit variable Rate Encoder" [15] for compression of the data. To access data from the disk, we have to first decrypt and then uncompress the decrypted data.

2.2 Disk Encryption Constraints

The common existing disk constraints are:

Data size. The ciphertext length should be the same as the plaintext length. Here, we use the current standard (512-byte) for the plaintext.

Performance. The used mode of operation should be fast enough, as to be transparent to the users. If the mode of operation results in a significant and noticeable slowdown of the computer, there will be great user resistance to its deployment.

3. Disk Encompression with Tweaked Block Chaining

3.1 Goals

The goals of designing the Disk Encompression with Tweaked Block Chaining (DETBC) mode are:

Security: The constraints for disk encryption imply that the best achievable security is essentially what can be obtained by using ECB mode with a different key per block [21]. This is the aim.

Complexity: DETBC complexity should be at least as fast as the current available solutions.

Parallelization: DETBC should offer some kind of parallelization.

Error propagation: DETBC should propagate error to further blocks (this may be useful in some applications).

3.2 Terminologies

The following terminologies are used to describe DETBC:

$\mathbf{P_i}$: The plaintext block i of size 128 bits.

$\mathbf{J_s}$: The sequential number of the 512-byte sector s inside the track encoded as 5-bit unsigned integer.

$\mathbf{I_i}$: The address of block i encoded as 64-bit unsigned integer.

$\mathbf{T_i}$: The tweak i.

$\boldsymbol{\alpha}$: Primitive element of $GF\ (2^{128})$.

←: Assignment of a value to a variable.

‖: Concatenation operation.

$\mathbf{PP_i}$: $P_i \oplus T_{i-1}$

$\mathbf{K_1}$: Encryption key of size 128-bit used to encrypt the PP.

$\mathbf{K_2}$: Tweak key of size 128-bit used to produce the tweak .

$\mathbf{EK_1}$: Encryption using AES algorithm with key $K_{1.}$

$\mathbf{DK_1}$: Decryption using AES algorithm with key $K_{1.}$

$\mathbf{C_i}$: The ciphertext block i of size 128 bits.

$\oplus$: Bitwise Exclusive-OR operation.

$\otimes$: Multiplication of two polynomials in $GF\ (2^{128})$.

3.3 Tweak Calculation

In our proposed scheme, the mode of operation takes four inputs to calculate the ciphertext (4096-bit). These inputs are:

1. The plaintext of size 4096-bit.
2. Encryption key of size 128 or 256-bit.
3. Tweak key of size 128 or 256-bit.
4. Sector ID of size 64-bit.

Usually a block cipher accepts the plaintext and the encryption key to produce the ciphertext. Different modes of operation have introduced other inputs. Some of these modes use initial vectors IV like in CBC, CFB and OFB modes [7], counters like in CTR [8] or nonces like in OCB mode [9]. The idea of using a tweak was suggested in HPC [10] and used in Mercy [16]. The notion of a tweakable block cipher and its security definition was formalized by Liskov, Rivest and Wagner [11]. The idea behind the tweak is that it allows more flexibility in design of modes of operations of a block cipher. There are different methods to calculate tweak from the sector ID like ESSIV [13] and encrypted sector ID [14].

In this work, the term tweak is associated with any other inputs to the mode of operation with the exception of the encryption key and the plaintext. Here, an initial tweak T_0, which is equal to the product of encrypted block address, where the block address (after being padded with zeros) is encrypted using AES by the tweak key, and α^{Js}, where Js is the sequential number of the 512-byte sector s inside the track encoded as 5-bit unsigned integer and α is the primitive element of $GF\ (2^{128})$, will be used as the initialization vector (IV) of CBC. The successive tweaks are the product of encrypted block address and the previous cipher text instead of α^{Js}. When next sector comes into play, again initial tweak is used, and the successive tweaks are again the product of encrypted block address and previous ciphertext. This is done so assuming that each track has 17 sectors and each sector has 32 blocks as per the standard disk structure. This procedure continues till end of the input file.

3.4 Efficient Multiplication in GF (2^{128})

Efficient multiplication in $GF\ (2^{128})$ may be implemented in numerous ways, depending on whether the multiplication is hardware or software and optimization scheme. In this work, we perform 16-byte multiplication. Let a, and b are two 16-byte operands and we consider the 16-byte output. When these blocks are interpreted as binary polynomials of degree 127, the procedure computes p = a*b mod P, where P is a 128-degree polynomial $P_{128}(x) = x^{128} + x^7 + x^2 + x + 1$.Multiplication of two elements in the finite field $GF\ (2^{128})$ is computed by the polynomial multiplication and taking the remainder of the Euclidean division by the chosen irreducible polynomial. In this case, the irreducible polynomial is $P_{128}(x)= x^{128} + x^7 + x^2 + x + 1$.

Table 1. Algorithm for Multiplication in GF (2^{128}).
Computes the value of p = a * b, where a, b and p ϵ GF (2^{128})

```
Algorithm PolyMult16( a, b) {
   p = 0;   /* Product initialized to zero*/
   while (b) {
      if (b & 1)  p = p ⊕ a; /*p xor a if the LSB of b is 1*/
      if (a127  = = 0) a << = 1; /*Left shift of bits in a by 1*/
      else a = (a <<= 1) ⊕ 0x87;/* x^128 + x^7 + x^2 + x + 1  */
      b >>= 1;/* Right shift of bits in the multiplier by 1 */
   }
   return p;
```

}

3.5 Efficient Modular Exponentiation

Compute efficiency: $z = x^c \bmod n$

Express c as follows: $c = \sum_{i=0}^{L-1} (c_i * 2^i)$,

where c_i = 0 or 1, value of i from 0 to (L-1) and L is the number of bits to represent c in binary.

The well-known Square-And-Multiply algorithm reduces the number of modular multiplications required to compute $x^c \bmod n$ to at most 2L. It follows that $x^c \bmod n$ can be computed in time $O(Lk^2)$. Total number of modular multiplications is at least L and at most 2L. Therefore, time complexity is the order of $[(\log c)*k^2]$, where n is a k-bit integer.

Efficient exponent in the finite field GF (2^{128}) is computed by the polynomial multiplication and taking the remainder of the Euclidean division by the chosen irreducible polynomial. In this case, the irreducible polynomial is $P_{128}(x) = x^{128}+x^7+x^2+x+1$.

Table 2. Algorithm for computing of $z = x^c \bmod n$, where x, c and z ϵ GF(2^{128})

```
Algorithm Square_And_Multiply ( x, c, n){
   z = 1; /* z initialized to one*/
   for (i = (L - 1); i > = 0; i--) {
         z = (z * z) mod n;
         if ( c_i = = 1)
               z = (z * x) mod n;
   }
   return (z);
}
```

4. Implementation of the Proposed Scheme

The design includes the description of the DETBC transform in both encryption and decryption modes, as well as how it should be used for encryption of a sector with a length that is not an integral number of 128-bit blocks.

4.1 Encryption of a Data Unit.

The encryption procedure for a 128-bit block having index j is modeled with Equation (1):

C_i ← DETBC-AES-blockEnc (Key, P_i, I, j) ………..(1)

where

- Key is the 256-bit AES key
- P_i is a block of 128 bits (i.e., the plaintext)
- I is the address of 128-bit block inside the data unit
- j is the logical position or index of the 128-bit block inside the sector
- C_i is the block of 128 bits of ciphertext resulting from the operation

The key is parsed as a concatenation of two fields of equal size called Key_1 and Key_2 such that:

$Key = Key_1 \| Key_2$.

The plaintext data unit is partitioned into m blocks, as follows:

$P = P_1 \| \ldots \| P_{m-1} \| P_m$

where m is the largest integer such that 128(m-1) is no more than the bit-size of P, the first (m -1) blocks $P_1,\ldots, P_{m-1}$ are each exactly 128 bits long, and the last block P_m is between 0 and 127 bits long (P_m could be empty, i.e., 0 bits long).

The ciphertext C_i for the block having index j shall then be computed by the following or an equivalent sequence of steps (see Figure 2):

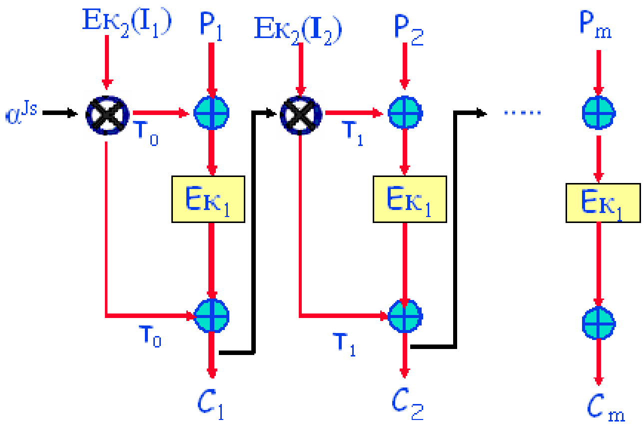

Figure 2. Encryption of data unit using DETBC.

Algorithm DETBC-AES-blockEnc(Key, P_i, I_i, j)

Case1 (j = 0):

1. T_{i-1} ← AES-enc (Key_2, I_i) ⊗ α^{Js}
2. PP_i ← $P_i \oplus T_{i-1}$
3. CC_i ← AES-enc(Key_1, PP_i)
4. C_i ← $CC_i \oplus T_{i-1}$

Case2 (j > 0):

1. T_{i-1} ← AES-enc (Key_2, I_i) ⊗ C_{i-1}
2. PP_i ← $P_i \oplus T_{i-1}$
3. CC_i ← AES-enc(Key_1, PP_i)
4. C_i ← $CC_i \oplus T_{i-1}$

AES-enc(K, P) is the procedure of encrypting plaintext P using AES algorithm with key K, according to FIPS-197.The multiplication and computation of power in step (1) is executed in *GF* (2^{128}), where α is the primitive element defined in 3.2(see 3.4 & 3.5).

The cipher text C is then computed by the following or an equivalent sequence of steps:

Algorithm DETBC-Encrypt(Key, *P*, *I*)

1. for i ← 0 to m-3 do
   a) j ← i % 32
   b) C_{i+1} ← DETBC-AES-blockEnc (Key, P_{i+1}, I_{i+1}, j)
2. r ← bit-size of P_m
3. if r = 0 then do
   a) j ← (m-2) % 32
   b) C_{m-1} ← DETBC-AES-blockEnc (Key, P_{m-1}, I_{m-1}, j)
   c) C_m ← empty

4. else do
   a) $j \leftarrow (m-2)$ % 32
   b) $CC_{m-1} \leftarrow$ DETBC-AES-blockEnc(Key, P_{m-1}, I_{m-1}, j)
   c) C_m ← first leftmost r bits of CC_{m-1}
   d) C' ← last rightmost (128-r) bits of CC_{m-1}
   e) $PP_{m-1} \leftarrow P_m \parallel C'$
   f) $j \leftarrow (m-1)$ % 32
   g) $C_{m-1} \leftarrow$ DETBC-AES-blockEnc(Key, PP_{m-1}, I_m, j)
5. $C \leftarrow C_1 \parallel \ldots \parallel C_{m-1} \parallel C_m$

An illustration of encrypting the last two blocks $P_{m-1}P_m$ in the case that P_m is a partial block (r > 0) is provided in Figure 3.

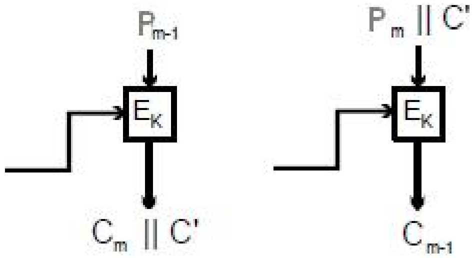

Figure 3. DETBC encryption of last two blocks when last block is 1 to 127 bits.

4.2 Decryption of a Data Unit.

The decryption procedure for a 128-bit block having index j is modeled with Equation (2):

$P_i \leftarrow$ DETBC-AES-blockDec(Key, C_i, I, j) ……….(2)

where

Key is the 256-bit AES key
C_i the 128-bit block of ciphertext
I is the address of the 128-bit block inside the data unit
j is the logical position or index of the 128-bit block inside the sector
P_i is the block of 128-bit of plaintext resulting from the operation

The key is parsed as a concatenation of two fields of equal size called Key_1 and Key_2 such that:

$Key = Key_1 \parallel Key_2$.

The ciphertext is first partitioned into m blocks, as follows:

$C = C_1 \parallel \ldots \parallel C_{m-1} \parallel C_m$

where m is the largest integer such that 128(m-1) is no more than the bit-size of C, the first (m-1) blocks $C_1, \ldots, C_{m-1}$ are each exactly 128 bits long, and the last block C_m is between 0 and 127 bits long (C_m could be empty, i.e., 0 bits long).

The plaintext P_i for the block having index j shall then be computed by the following or an equivalent sequence of steps (see Figure 4):

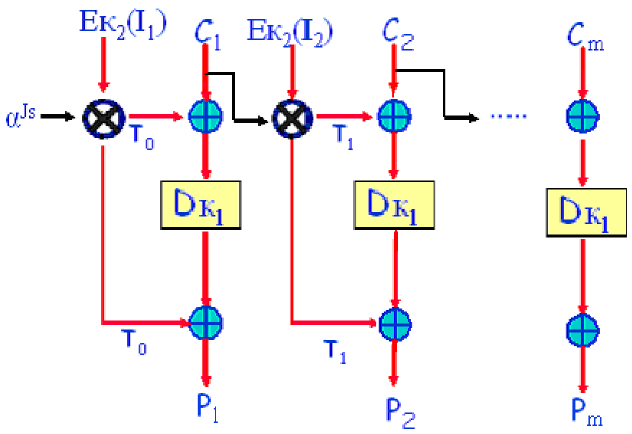

Figure 4. Decryption of ciphertext blocks using DETBC.

Algorithm DETBC-AES-blockDec(Key, C_i, I_i, j)

Case1 (j = 0):

1. $T_{i-1} \leftarrow$ AES-enc (Key_2, I_i) $\otimes \alpha^{Js}$
2. $CC_i \leftarrow C_i \oplus T_{i-1}$
3. $PP_i \leftarrow$ AES-dec(Key_1, CC_i)
4. $P_i \leftarrow PP_i \oplus T_{i-1}$

Case2 (j > 0):

1. $T_{i-1} \leftarrow$ AES-enc (Key_2, I_i) $\otimes C_{i-1}$
2. $CC_I \leftarrow C_i \oplus T_{i-1}$
3. $PP_i \leftarrow$ AES-dec(Key_1, CC_i)
4. $P_i \leftarrow PP_i \oplus T_{i-1}$

AES-dec (K,C) is the procedure of decrypting ciphertext C using AES algorithm with key K, according to FIPS-197. The multiplication and computation of power in step (1) is executed in *GF* (2^{128}), where α is the primitive element defined in 3.2 (see 3.4 & 3.5).

The plaintext P is then computed by the following or an equivalent sequence of steps:

Algorithm DETBC-Decrypt (Key, C, *I*)

1. for i ← 0 to m-3 do
   a) $j \leftarrow i$ % 32
   b) $P_{i+1} \leftarrow$ DETBC-AES-blockDec (Key, C_{i+1}, I_{i+1}, j)
2. r ← bit-size of C_m
3. if r = 0 then do
   a) $j \leftarrow (m-2)$ % 32
   b) $P_{m-1} \leftarrow$ DETBC-AES-blockDec (Key, C_{m-1}, I_{m-1}, j)
   c) P_m ← empty
4. else do
   a) $j \leftarrow (m-1)$ % 32
   b) $PP_{m-1} \leftarrow$ DETBC-AES-blockDec(Key, C_{m-1}, I_m, j)
   c) P_m ← first leftmost r bits of PP_{m-1}
   d) C' ← last rightmost (128-r) bits of PP_{m-1}
   e) $CC_{m-1} \leftarrow C_m \parallel C'$
   f) $j \leftarrow (m-2)$ % 32
   g) $P_{m-1} \leftarrow$ DETBC-AES-blockDec(Key, CC_{m-1}, I_{m-1}, j)
5. $P \leftarrow P_1 \parallel \ldots \parallel P_{m-1} \parallel P_m$

An illustration of encrypting the last two blocks $C_{m-1}C_m$ in the case that C_m is a partial block (r > 0) is provided in Figure 5.

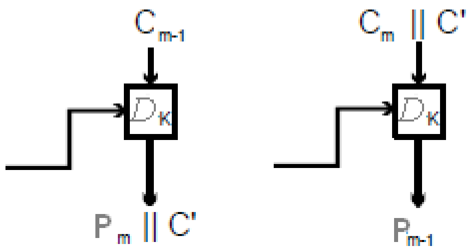

Figure 5. DETBC decryption of last two blocks when last block is 1 to 127 bits.

5. Performance Analysis

Security: Each block is encrypted with a different tweak T, which is the result of a non-linear function (multiplication) of encrypted file address and previous ciphertext (α^{Js} for 1^{st} block); due to this step the value of the tweak is neither known nor controlled by the attacker. By introducing the tweak, the attacker can not perform the mix-and-match attack [21] among blocks of different sectors, as each sector has a unique secret tweak. Any difference between two tweaks result full diffusion in both the encryption and decryption directions. These enhance the security.

Here we also give option for the value of α to the user; it reduces the probability of getting plaintext from ciphertext. This is so because same plaintext produces different ciphertext if we choose different value for α. This also increases confusion.

Complexity: DETBC possesses high performance as it uses only simple and fast operations as standard simple shift and add (xor) operators are used in the multiplication in the finite field *GF* (2^{128}) having O(1) time complexity. Compression before encryption also enhances the speed and hence performance.

Parallelization: DETBC can be parallelized on the sector level as each sector is encrypted independently to other sectors. Also a plaintext can be recovered from just two adjacent blocks of ciphertext. As a consequence, decryption can be parallelized.

Error propagation: As each block depends on its previous block, a one-bit change in a plaintext affects all following ciphertext blocks. Hence, error propagation is met.

DETBC meets all its design goals.

5.1 Comparison

In this section, we compare our model with existing models [18]. The speed presented in table 3 for our mode (DETBC), is obtained from C implementation and taking a binary file as input, running on a 3 GHz Intel Pentium IV processor.

Table 3. Number of clock cycles reported by different mode of operation.

Mode	Key Length 128-bits
DETBC	4158
CBC	12630
CFB	12585
ESCC	12660

Note that the reported values are the minimum from measurements of different input files, to eliminate any initial overheads or cache misses factors. It is clear that DETBC possesses high throughput.

6. Conclusions and Open Problem

In this paper, we present a new mode of operation for disk encryption applications. The proposed scheme possesses a high throughput. Although, it is designed based on the CBC mode, it can be parallelized and does not suffer from the bit flipping attack. This mode also utilizes less memory space as the input file is first compressed and then it is encrypted.

There still remain many open problems in the search for efficient and secure disk encryption. There is a lack of good Boolean functions for the tweak generator which are efficient and also resist the cryptanalytic attacks, in particular algebraic and fast algebraic attacks.

References

[1] Bruice Schneier, Applied Cryptography, Wiley Press, Second Edition.
[2] Douglas R. Stinson, Cryptography Theory and Practice, CRC Press, Second Edition.
[3] Mark Nelson, Jean-Loup Gailly, The Data Compression Book, M&T Press, Second Edition.
[4] William Stallings, Cryptography and Network Security, Pearson Education, Fourth Edition.
[5] Anderson, R., Biham, E.: Two practical and provable secure block ciphers: BEAR and LION. In: Gollmann, D. (ed.) FSE 1996. LNCS, vol. 1039, pp. 113-120. Springer, Heidelberg (1996)
[6] S. Halevi and P. Rogaway, A tweakable enciphering mode, in Lecture Notes in Computer Science, D. Boneh, Ed. Berlin, Germany: Springer-Verlag, 2003, vol. 2729, pp. 482-499.
[7] A. Menezes, P. V. Oorschot., and S. Vanstone. Handbook of Applied Cryptography. CRC Press, 1996.
[8] D. McGrew. Counter Mode Security: Analysis and Recommendations. http:// citeseer.ist.psu.edu/mcgrew02counter.html, 2002.
[9] P. Rogaway, M. Bellare, and J. Black. OCB: A blockcipher mode of operation for efficient authenticated encryption. ACM Trans. Inf. Syst. Secur., 6(3):365-403, 2003.

[10] R. Schroeppel. The Hasty Pudding Cipher. The first AES conference, NIST, 1998. http://www.cs.arizona.edu/~rcs/hpc
[11] M. Liskov, R. L. Rivest, and D.Wagner, Tweakable block ciphers, in Lecture Notes in Computer Science, M. Yung, Ed. Berlin, Germany: Springer-Verlag, 2002, vol.2442, pp. 31-46.
[12] S. Lucks, BEAST: A fast block cipher for arbitrary block sizes. In: Horster, P. (ed.) Communications and Multimedia Security II, Proceedings of the IFIP TC6/TC11 International Conference on Communications and multimedia Security (1996)
[13] C. Fruhwirth, New Methods in Hard Disk Encryption. http://clemens.endorphin.org/nmihde/nmihde-A4-ds.pdf, 2005.
[14] N. Ferguson. AES-CBC + Elephant diffuser: A Disk Encryption Algorithm for Windows Vista. http://download.microsoft.com/download/0/2/3/0238acaf-d3bf-4a6d-b3d6-0a0be4bbb36e/BitLockerCipher200608.pdf,2006.
[15] Lempel-Ziv-Welch. http://en.wikipedia.org/wiki/Lempel-Ziv-Welch
[16] P. Crowley. Mercy, A fast large block cipher for disk sector encryption. In Bruce Schneier, editor, Fast Software Encryption: 7th International Workshop, FSE 2000, 2001.
[17] Mitsuru Matsui. The first experimental cryptanalysis of the data encryption standard. In Y. Desmedt, editor, Advances in Cryptology-CRYPTO 1994, number 839 in Lecture Notes in Computer Science, pages 1-11. Springer-Verlag, 1994.
[18] M. Abo El-Fotouh and K. Diepold, Extended Substitution Cipher Chaining Mode. http://eprint.iacr.org/2009/182.pdf
[19] P. Sarkar, Efficient Tweakable Enciphering Schemes from (Block-Wise) Universal Hash Functions. http://eprint.iacr.org/2008/004.pdf
[20] S. Fluhrer, Cryptanalysis of the Mercy block Cipher. In: Matsui, M. (ed.) FSE 2001. LNCS, vol. 2355, p. 28. Springer, Heidelberg (2002)
[21] I. P1619. IEEE standard for cryptographic protection of data on block oriented storage devices. IEEE Std. 1619-2007, April 2008. http://axelkenzo.ru/downloads/1619-2007-NIST-Submission.pdf
[22] P. Rogaway. Efficient Instantiations of Tweakable Block ciphers and Refinements to Modes OCB and PMAC. In Pil Joong Lee, editor, Advances in Cryptology - ASIACRYPT ’04, volume 3329 of LNCS, pages 16-31, 2004.
[23] Disk encryption theory. http://en.wikipedia.org/wiki/Disk_encryption_theory
[24] Latest SISWG and IEEE P1619 drafts for Tweakable Narrow-block Encryption. http://grouper.ieee.org/groups/1619/email/pdf00017.pdf

Debasis Gountia received the Bachelor of Computer Science and Engineering degree from Biju Patnaik University of Technology, Rourkela, India , in 2003. He received the Master of Technology degree in Computer Science and Engineering from the Indian Institute of Technology, Kharagpur, India in 2010.

Since January 2006, he has been a Lecturer with the College of Engineering & Technology, Bhubaneshwar, India. His research interests include cryptography, formal language and automata theory, operating systems, and distributed systems.

Dipanwita Roy Chowdhury received the Bachelor and the Master of Technology degree in Computer Science, both from University of Kolkata, India, in 1987 and 1989, respectively. She received the Ph.D. degree from the Indian Institute of Technology, Kharagpur, India in 1994.

She is a Professor with the Indian Institute of Technology, Kharagpur, India. Her research interests include cryptography, error correcting code, cellular automata, and VLSI design and testing.