THE SUBSTITUTION CIPHER CHAINING MODE

Mohamed Abo El-Fotouh and Klaus Diepold
Institute for Data Processing (LDV), Technische Universität München (TUM), 80333 Munich, Germany

Keywords: Modes of operation, SCC, disk encryption, AES.

Abstract: In this paper, we present a new tweakable narrow-block mode of operation, the Substitution Cipher Chaining mode (SCC), that can be efficiently deployed in disk encryption applications. SCC is characterized by its high throughout compared to the current solutions and it can be parallelized. We used this mode to modify Windows Vista's disk encryption algorithm, to offer some parallelism in its original implementation and to improve its diffusion properties.

1 INTRODUCTION

In todays computing environment, there are many threats to the confidentiality of information stored on end user devices, such as personal computers, consumer devices (e.g. PDA), and removable storage media like (e.g., USB or external hard drive). A common threat against end user devices is device loss or theft, that can lead to identity theft and other frauds. Someone with physical access to a device has many options for attempting to view or copy the information stored on that device. Another concern is insider attacks, such as an employee attempting to access sensitive information stored on another employees device. Malware, another common threat, can give attackers unauthorized access to a device, transfer information from the device to an attackers system, and perform other actions that jeopardize the confidentiality of the information on a device (NIST, 2007).

Usually the data stored on the PC asset is often significantly more valuable to a corporation than the asset itself (Ferguson, 2006). To prevent the disclosure of sensitive data, the data needs to be secured. Disk encryption applications are usually used to protect the data on the disk by encrypting it, where all the data is encrypted with a single/multiple key(s) and encryption/decryption are done on the fly, without user interference.

Disk encryption usually encrypts/decrypts a whole sector at a time. There exist dedicated block ciphers to encrypt the whole sector at a time like Bear, Lion, Beast and Mercy (Anderson and Biham, 1996; Anderson and Biham, 1996; Lucks, 1996; Crowley, 2001). Bear, Lion and Beast are considered to be slow, as they process the data through multiple passes. And Mercy was broken in (Fluhrer, 2002). The other method is to let a block cipher like the Advanced encryption standard AES (Daemen and Rijmen, 1998) (with 16-bytes as a block size) to process the data within a mode of operation. These modes of operation can be divided into two main classes the narrow-block and wide-block modes. The narrow-block modes operate on relatively small portions of data (typically 16-bytes when AES is used), while the wide-block modes encrypt or decrypt a whole sector (typically 512-bytes) at a time (IEEE P1619 homepage, 2007).

The current narrow-block modes of operations are either slow and do not offer error propagation (like XTS (Rogaway, 2003) and LRW (Liskov et al., 2002)) or are subjected to some attacks like (CBC and CFB (Menezes et al., 1996)). A need for a new secure and fast mode of operation, that offer error propagation, has aroused.

In this paper, we propose a novel narrow-block disk encryption mode of operation. We named this mode Substitution Cipher Chaining mode (SCC). The SCC mode is a tweakable block cipher mode of operation, that is base on Cipher Block Chaining mode (CBC) (Menezes et al., 1996) and the Static Substitution Model (SSM) (El-Fotouh and Diepold, 2008a). The SSM model can provide a block cipher with a secondary key. The secondary key is used to replace some bits of the cipher's expanded key. In SCC, each sector has its unique tweak, this tweak and the previous ciphertext block will replace some bits of the expanded key (with the exception to the first block).

Abo El-Fotouh M. and Diepold K. (2008).
THE SUBSTITUTION CIPHER CHAINING MODE.
In *Proceedings of the International Conference on Security and Cryptography*, pages 421-429
DOI: 10.5220/0001917904210429
Copyright © SciTePress

In section 2, we will present the constraints facing the disk encryption applications. In section 3, we will describe the current narrow-block modes of operations used in disk encryption. In section 4, we presented our proposed mode of operation. In section 5, we present a performance analysis and a benchmark among the narrow-block modes of operations. In section 7, we present out proposed modification to Windows Vista's disk encryption algorithm. Finally, we present our conclusions in section 8.

2 DISK ENCRYPTION

Disk encryption is usually used to encrypt all the data on the hard disk, where all the hard disk is encrypted with a single/multiple key(s) and encryption/decryption are done on the fly, without user interference (El-Fotouh and Diepold, 2007). The encryption is on the sector level, that means each sector should be encrypted separately. In the following sub section, we will define the existing constrains.

2.1 Disk Encryption Constraints

The main constraints:

Data Size. The ciphertext length should be the same as the plaintext length. In this paper, we will use the current standard (512-bytes) for the plaintext.

Performance. The used mode of operation should be fast enough, as to be transparent to the users (If using the mode of operation results in a significant and noticeable slowdown of the computer there will be great user resistance to its deployment (Ferguson, 2006)).

2.2 General Scheme

In Figure 1, we present our general scheme for encrypting a sector, where the mode of operation takes four inputs to calculate the ciphertext (4096-bits). These inputs are:

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

2.3 Tweak

Usually a block cipher accepts the plaintext and the encryption key to produce the ciphertext. Modes

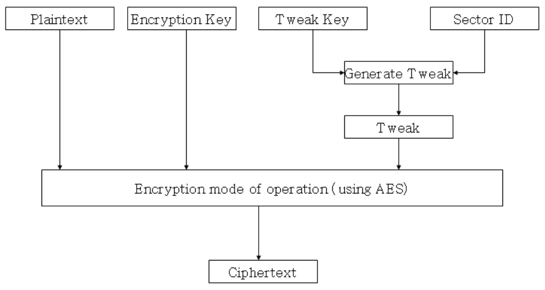

Figure 1: Disk encryption general scheme for encrypting a sector.

of operations have introduced other inputs. Some of these mode use initial vectors like in CBC, CFB and OFB modes (Menezes et al., 1996), counters like in CTR (McGrew, 2002) or nonces like in OCB mode (Rogaway et al., 2001). The idea of using a tweak was suggested in HPC (Schroeppel, 1998) and used in Mercy (Crowley, 2001). In (Liskov et al., 2002), the formal definition of tweakable block ciphers has been introduced. In this paper, the term tweak is associated with any other inputs to the mode of operation with the exception of the encryption key and the plaintext.

2.4 Tweak Calculation

There are different methods to calculate the tweak from the sector ID like ESSIV (Fruhwirth, 2005) and encrypted sector ID (Ferguson, 2006). We are going to use the encrypted sector ID approach, where the sector ID (after being padded with zeros) is encrypted by the tweak key to produce the tweak.

3 NARROW-BLOCK MODES OF OPERATIONS

3.1 Terminologies

The following terminologies are used to describe the modes of operations in this paper.

IN. The input plaintext of size 4096-bits.

EK. The encryption key of size 128 or 256-bits.

TK. The tweak key of size 128 or 256-bits.

BK. The block key of size 128 or 256-bits.

SID. The sector ID encoded as 64-bits unsigned integer.

GetTweak(TK,SID). Encrypts (using AES) SID after padding with zeros with TK and returns the result.

Table 1: CBC listing for disk encryption.

```
Encrypt-CBC(IN,EK,TK,SID)
    T=GetTweak(TK,SID)
    ExKey=Expand-Key(EK)
    $IN_0 = IN_0 \oplus T$
    AES-Encrypt(ExKey,$IN_0$,$OUT_0$)
    for i=1 to 31
        $IN_i = IN_i \oplus OUT_{i-1}$
        AES-Encrypt(ExKey,$IN_i$,$OUT_i$)
    end for
return OUT
```

Table 2: CFB listing for disk encryption.

```
Encrypt-CFB(IN,EK,TK,SID)
    T=GetTweak(TK,SID)
    ExKey=Expand-Key(EK)
    AES-Encrypt(ExKey,T,$OUT_0$)
    $OUT_0 = OUT_0 \oplus IN_0$
    for i=1 to 31
        AES-Encrypt(ExKey,$OUT_{i-1}$,$OUT_i$)
        $OUT_i = OUT_i \oplus IN_i$
    end for
return OUT
```

Table 3: LRW listing for disk encryption.

```
Encrypt-LRW(IN,EK,TK,SID)
    ExKey=Expand-Key(EK)
    for i=0 to 31
        $TK = TK \otimes SID$
        $IN_i = IN_i \oplus TK$
        AES-Encrypt(ExKey,$IN_i$,$OUT_i$)
        $OUT_i = OUT_i \oplus TK$
        SID=SID +1
    end for
return OUT
```

T. The tweak.

ExKey. The expanded AES key.

Expand-Key(EK). Expands the EK with the AES key setup routine and returns the result.

X_i. The ith block of text X, where a block is 128-bits.

$\oplus$. Bitwise xor operation.

AES-Encrypt(ExKey,IN,OUT). Encrypts IN, using the AES encryption routine with ExKey as the expanded key, and returns OUT.

$\otimes$. Multiplication operation in finite field.

+. Integer addition operation modulo 2^{64}.

Substitute(T,ExKey,i). Replaces the ith round subkeys in ExKey with T (Note that: the first round of the AES is round zero and it is the pre-whitening process).

len(X). Returns the length of the string X in bits.

3.2 CBC: Cipher Block Chaining

CBC is the most used cipher mode of operation for hard disk encryption (Fruhwirth, 2005), where usually each sector is encrypted with a different initial vector (IV). The listing of CBC is in table 1. Where for the first block the plaintext is xored with the tweak before it is encrypted, and for the other blocks the plaintext is xored with the last ciphertext block before it is encrypted. This recursive structure does not allow parallelization.

For CBC, an attacker with read/write access to the encrypted disk can copy a ciphertext sector from one position to another, and an application reading the sector of the new location will still get the same plaintext sector (except perhaps the first 128-bits) (IEEE P1619 homepage, 2007). In CBC, the attacker can flip arbitrary bits in one block at the cost of randomizing the previous block (Ferguson, 2006). Other attacks like Watermarking attack is also applicable (Fruhwirth, 2005). Due these weaknesses we do not recommend using CBC in disk encryption applications.

3.3 CFB: Cipher Feedback

CFB turns a block cipher into a self-synchronizing stream cipher. The listing of CFB is in table 2. Where the first ciphertext block is the result of the xor process between the first plaintext block and the tweak after encryption, and for the other blocks the plaintext is xored with the encryption of the last ciphertext block to produce the next ciphertext block. This recursive structure does not allow parallelization.

Using CFB without message authentication code (MAC) (Menezes et al., 1996) is trivially malleable, and an attacker with write access to the encrypted media can flip any bit of the plaintext simply by flipping the corresponding ciphertext bit (IEEE P1619 homepage, 2007). Due this weakness we do not recommend using CFB in disk encryption applications.

3.4 LRW

LRW is a tweakable mode of operation that can be easily parallelized. The listing of LRW is in table 3. It xors the plaintext before and after encryption with the tweak key (Note that the tweak key is multiplied "in the finite field of AES" in each loop with a counter "initialized by the sector ID").

The IEEE Security in Storage work group (SISWG) used it in its early draft as a standard for narrow-block encryption mode (when AES is the underlying cipher). Then it was replaced by XTS due to the following security issue. An attacker can de-

Table 4: XTS listing for disk encryption.

```
Encrypt-XTS(IN,EK,TK,SID)
    T=GetTweak(TK,SID)
    ExKey=Expand-Key(EK)
    for i=0 to 31
        X = T $\otimes$ $\alpha^i$
        IN$_i$ = IN$_i$ $\oplus$ X
        AES-Encrypt(ExKey,IN$_i$,OUT$_i$)
        OUT$_i$= OUT$_i$ $\oplus$ X
    end for
return OUT
```

rive the LRW tweak key TK from the ciphertext, if the plaintext contains TK$||0^n$ or $0^n||$TK. Here $||$ is the concatenation operator and 0^n is a zero block. This may be an issue for software that encrypts the partition of an operating system under which this encryption software is running (at the same time). The operating system could write the LRW tweak key to encrypted swap/hibernation file. If the tweak key TK is known, LRW does not offer indistinguishability under chosen plaintext attack anymore, and the same input block permutation attacks of ECB mode are possible (IEEE P1619 Email Archive, 2007). Due these weaknesses we do not recommend using LRW in disk encryption applications.

3.5 XTS

XTS is a tweakable mode of operation that can be easily parallelized. The listing of XTS is in table 4. It xors the plaintext before and after encryption with the tweak value (Note that the tweak value is multiplied in the finite field GF(2^{128}) in each loop with the ith power of α "a primitive element of GF(2^{128})").

XTS is now the current narrow-block standard of SISWG, it is slower than LRW. XTS is an instantiation of XEX (Rogaway, 2003) mode of operation and this mode of operation has a problem when a lot of data is encrypted with the same key, because a collision will appear. In case of a collision between block i and block j, the attacker can use his ability to create legally encrypted data for position i and his ability to modify ciphertext in position j to modify the ciphertext at j so it will decrypt to an arbitrary attacker-controlled value. This security leak appears, when the same key is used to encrypt more than a terabyte of data (IEEE P1619 homepage, 2007).

4 PROPOSED MODE

4.1 Goals

The goals of designing the SCC 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 (IEEE P1619 homepage, 2007). This is our aim.

Performance. SCC should be at least as fast as the current solutions.

Error Propagation. SCC should propagate error to further blocks (this may be useful in some applications).

4.2 Keys

The secret key in SCC is divided into three different keys (each of them can be either 128- or 256-bits):

1. **EKey.** which is used to generate the expanded key, used in encrypting the blocks .
2. **TK.** which is used to encrypt the sector ID to produce the tweak.
3. **BK.** which is used to generate the **BT** array, where **BT** is an array of sixty four 128-bits words.
    - **BT** is constructed once at the initialization of SCC mode.
    - **BT** is constructed using the AES in the counter mode, where the counter is initialized with zero and **BK** is the encryption key for the counter mode.

4.3 Design

We decided to build the SCC mode using the SSM model (El-Fotouh and Diepold, 2008a) to inherit from its security and high performance and use CBC like operations to gain the error propagation property. The listing of SCC is in table 5 and it works as follows:

- The tweak **T** is calculated by encrypting the sector ID with the tweak key **TK**, due to this step the value of the tweak is neither known nor controlled by the attacker.
- The expanded key **ExKey** is calculated.
- the values of x, y and z are determined by the encryption key size.
- For the first block:
    - The secret tweak **T** replaces the subkeys of the yth round.

Table 5: SCC listing for disk encryption.

```
Encrypt-SCC(IN,EK,Keylen,TK,SID)
    T=GetTweak(TK,SID)
    ExKey=Expand-Key(EK)
    KL=len(EK)
    if(KL==128)
        x=4    y=5    z=6
    else
        x=5    y=7    z=10
    end if
    Substitute(T,ExKey,y)
    Substitute($BT_0$,ExKey,x)
    Substitute($BT_1$,ExKey,z)
    AES-Encrypt(ExKey,$IN_i$,$OUT_i$)
    for i=1 to 31
        Substitute($BT_{2\times i}$,ExKey,x)
        Substitute($BT_{2\times(i+1)}$,ExKey,z)
        TT=$OUT_{i-1} \oplus$ T
        Substitute(TT,ExKey,y)
        AES-Encrypt(ExKey,$IN_i$,$OUT_i$)
    end for
return OUT
```

  - The secret 128-bits $\mathbf{BT}_0$ replace the subkeys of the xth round.
  - The secret 128-bits $\mathbf{BT}_1$ replace the subkeys of the zth round.
  - The first block is encrypted by the new expanded key.
- A loop that runs 31 times (where i takes the values from 1 to 31):
  - The secret 128-bits $\mathbf{BT}_{2\times i}$ replace the subkeys of the xth round.
  - The secret 128-bits $\mathbf{BT}_{(2\times i)+1}$ replace the subkeys of the zth round.
  - A variable **TT** is calculated by xoring ciphertext of the previous block with T.
  - **TT** acts as the *active tweak* and replaces the subkeys of the yth round in the expanded key.
  - The ith block is encrypted by the new expanded key.

4.4 Discussion of SCC Mode

The goal of SCC is to encrypt each block on the hard drive in a different way. This was achieved by using the SSM model, where:

- The active tweak **TT** is placed in the middle of the expanded key, to offer full diffusion and full confusion properties in both the encryption and decryption directions (i.e any difference between two active tweaks, will be associated with full confusion and full diffusion in both the encryption and decryption directions, eliminating the bit-flipping attack of the CBC mode). Note that AES requires only four rounds to obtain full bit confusion (or mixing) and diffusion (each input bit affecting each output bit) properties (May et al., 2002).
- Note that the active tweak **TT** is the result of xoring:
  1. The tweak **T** (which is unique, secret and not controlled by the attacker).
  2. The ciphertext of the previous block (which is known and controlled by the attacker).
  3. From the above two notes, the attacker does not know the value of **TT**, but can flip its bits. But as **TT** replaces the subkeys of the yth round (which is in the middle of the cipher), any change in **TT** will be associated with full confusion and full diffusion in both the encryption and decryption directions.
- Replacing the subkeys of the xth and zth rounds offers full diffusion and full confusion in the encryption and decryption directions among the blocks of the same sector. Note that all the values of **BT** are unique and key dependent.

Notes:

1. By introducing the tweak, the attacker can not perform the mix-and-match attack (IEEE P1619 homepage, 2007) among blocks of different sectors, as each sector has a unique secret tweak. The tweak replaces the subkeys of the middle round of the AES to assure that any difference between two tweaks, will be associated with full confusion and full diffusion in both the encryption and decryption directions. Thus, encrypting two equal blocks in different sectors will produce two different ciphertexts and decrypting two equal blocks in different sectors will produce different plaintexts.
2. By introducing the **BT** array values (that replaces certain words in the expanded key) the attacker can not perform the mix-and-match attack among the blocks within the same sector. As each sector has two distinct 128-bits in the expanded key. This requirement is achieved in both the encryption and decryption directions. As equal plaintext blocks (within the same sector), will have the same state until the xth encryption round then the state will change. And equal ciphertext blocks (within the same sector), will have the same state until the zth decryption round then the state will change.

4.5 Pros of SCC

Security. Each sector is encrypted in a different way, so replacing ciphertext between different sectors

Table 6: Theoretical operational time for each mode of operation.

	Encryption calls	Multiplication calls
CBC	33	0
CFB	33	0
LRW	32	32
XTS	33	32
SCC	33	0

will not help the attacker, as they are encrypted with a different expanded keys and each block within the sector is encrypted in a different way, due to the use of **BT** (so the attacker will not benefit from changing the positions of the blocks).

Performance. SCC possesses high performance as it uses only simple and fast operations.

Parallelization. SCC can be parallelized, actually it favors the dual core processors, it can be parallelized as following (using two processor cores):

- The second processor will compute till the yth round for all the plaintext blocks (except the first block), to produce intermediate ciphertext blocks.
- The first processor will compute the first ciphertext block form the first plaintext block.
- The first processor will compute the other ciphertext blocks form intermediate ciphertext blocks processed by the second processor.

In this way the ciphertext will be produced after 16.5 encryptions.

Error Propagation. As each sector depends on its previous sector, error propagation is met.

5 PERFORMANCE ANALYSIS

5.1 Operational Time

5.1.1 Theoretical

Usually the theoretical time is calculated with how many encryption calls are needed by each mode of operation, as the other operations like the xor or addition is considered negligible. But LRW and XTS uses finite field multiplication, which requires about half the time needed by the AES encryption (128-bits version) (McGrew, 2004). Table 6 summarizes the number of AES and finite field multiplication calls needed by each mode.

Table 7: Number of clock cycles reported by each mode of operation.

	Key length 128-bits	Key length 256-bits
CBC	12630	16898
CFB	12585	16935
LRW	19778	24015
XTS	24420	28846
SCC	12660	16867

5.1.2 Practical Operational Time

The Speed presented in table 7, are obtained form the optimized Gladman's C implementation (Gladman, 2006), Running on a PIV 3 GHz (Note that the values reported are in processor clock cycles). Note that the reported values are the minimum of 1000 measurements, to eliminate any initial overheads or cache misses factors.

5.1.3 Parallel Implementation

With the wide spread of multi-core processors, speeding up encryption using parallelization is made possible and parallelization is not a luxury anymore and can increase the performance significantly. Encryption mode of operation should support parallelization. CBC and CFB can not be parallelized, while LRW, XTS and SCC can be parallelized.

5.2 Benchmark

To summarize our results, we present a benchmark. The results of the benchmark is presented in table 8, where under column:

Speed. we reported the speed ranking of each mode of operation (note: CBC, CFB and SCC possess the same rank as they require almost the same number of clock cycles).

Parallelization. we reported the number of encryption calls required for each mode on a dual-core processor (after the tweak is calculated).

Error Propagation. we reported if the mode possesses the error propagation property or not.

From the results in table 8, it is clear that SCC mode is superior than all the other modes in all the benchmark metrics and we recommend its use in disk encryption applications.

6 ELEPHANT

Windows Vista Enterprise and Ultimate editions use Bitlocker Drive Encryption as its disk encryption al-

Table 8: Benchmark of the modes of operations.

	Speed	Parallelization (dual core)	Error propagation
CBC	1	32	√
CFB	1	32	√
LRW	4	16	X
XTS	5	16	X
SCC	1	16.5	√

gorithm, and at its heart is the AES-CBC + Elephant diffuser encryption algorithm (ELEPHANT). Bitlocker uses existing technologies like the AES in the CBC mode and TPM (Trusted Computing Group, 2008), together with two new diffusers. Figure 2 shows an overview of ELEPHANT (Ferguson, 2006). There are four steps to encrypt a sector:

1. The plaintext is xored with a sector key K$_s$ (1).
2. The result of the previous step run through diffuser A.
3. The result of the previous step run through diffuser B.
4. The result of the previous step is encrypted with AES in CBC mode using IV$_s$ (2), as the initial vector.

$$K_s = E(K_{sec}, e(s)) \mid\mid E(K_{sec}, e'(s)) \quad (1)$$

$$IV_s = E(K_{AES}, e(s)) \quad (2)$$

Where E() is the AES encryption function, K$_{sec}$ is a key used to generate K$_s$, K$_{AES}$ is the key used to generate the sector IV$_s$ and used in the AES-CBC process, e() is an encoding function that maps each sector number s into a unique 16-byte value. The first 8 bytes of the result are the byte offset of the sector on the volume. This integer is encoded in least-significant-byte first encoding. The last 8 bytes of the result are always zero and e$'$(s) is the same as e(s) except that the last byte of the result has the value 128.
Note that the plaintext and key are parameterized. In our study we used the following parameters:

1. Plaintext of size 4096-bits (the current standard sector size).
2. We examined the 256-bits key version of the AES (that supports maximum security), that means both K$_{sec}$ and K$_{AES}$ of size 256-bits.

6.1 The Diffusers

Diffuser A and diffuser B are very similar. The following notations are used to define the diffusers:

1. d$_i$ is the ith 32-bits word in the sector, if i falls outside the range then d$_i$ =d$_{i \bmod n}$, where n is the number of the 32-bits in the sector.

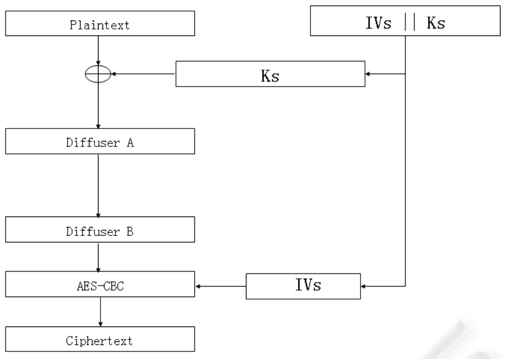

Figure 2: Overview of AES-CBC with Elephant Diffuser.

Table 9: Diffuser A and diffuser B.

Diffuser A:	**Diffuser B:**
for j=1 to AC	for j=1 to BC
for i=n-1,...,2,1,0	for i= n-1,...,2,1,0
t=(d$_{i-5}$ << RA$_{i \bmod 4}$)	t=(d$_{i+5}$ << RB$_{i \bmod 4}$)
t=t ⊕ d$_{i-2}$	t=t ⊕ d$_{i+2}$
d$_i$= d$_i$ - t	d$_i$= d$_i$ - t

2. AC and BC are the number of cycles of diffuser A and B, they are defined to be 5 and 3 respectively.
3. RA = [9, 0, 13, 0] and RB = [0, 10, 0, 25] hold the rotation constants of diffuser A and B respectively.
4. $\oplus$ is the bitwise xor operation.
5. << is the integer 32-bits left rotation operation, where the rotation value is written on its right size.
6. - is integer subtraction modulo 2^{32}.

Table 9 presents the description of diffuser A and diffuser B.

6.2 Proposed Modification

We propose to replace the CBC layer in ELEPHANT with a SCC layer. Pros of this modification over the original design:

- The SCC can be parallelized on a dual core machine, which can increase the performance.
- SCC has a better error propagation effect than CBC in decryption direction, as more bits of plaintext will be affected by a single bit change (typically 256-bits for the SCC and 129-bits for CBC).
- We named our variants ELEPHANT$^+$ (when AC=5 and BC=3) and ELEPHANT* (when AC=BC=3).

6.3 Bit Dependency Tests

- **BD-Encryption()** (El-Fotouh and Diepold, 2008b): is passed, when each bit in the ciphertext depends on every bit in the plaintext. It tries to answer: *does each bit in ciphertext depend on all the bits in the plaintext?*
- **BD-Decryption()** (El-Fotouh and Diepold, 2008b): is passed, when each bit in the plaintext depends on every bit in the ciphertext. It tries to answer: *does each bit in plaintext depend on all the bits in the ciphertext?*

The Bit-dependency functions are measured as following:

1. A dependency matrix M is constructed of size B $\times$ B (where B is the number of bits in the plaintext/ciphertext, here B = 4096).
2. The diagonal is initialized by 1 and all other bits are set to zero, as initially each bit depends only on itself.
3. Depending on the applied operation the matrix M is updated, BD-Encryption applies the operation in the encryption direction and BD-Decryption applies them in the decryption direction.
4. If an output bit is dependent on an input bit(s), the column of the output bit is ORed with that (those) of the input bit(s). For example:
   (a) Xor operation: each output bit is dependent on the corresponding input bit.
   (b) Addition and subtraction modulo 2^{32} operations are approximated to xor operation for simplicity and generality.
   (c) AES operation in CBC: each bit in the input 128-bit is dependent on the other 127-bits.
   (d) AES operation in SCC: each bit in the previous block (with the exception of the fist block) affects all the bits of the encrypted block (as full confusion and full diffusion properties are met).
   (e) 32-bits rotation: the columns change there order depending on the rotation amount and direction.
5. All the operation of the tested ciphers are applied and the matrix M is updated.
6. At the end the sum of all ones in the matrix is added and is divided by B^2.
7. If the returned value in the previous step is 1, this means that each bit of the output bits depends on all the bits of the input and the function is passed, it fails otherwise.

Table 10: BD-Encryption and BD-Decryption results.

	Pass	**AC′**	**BC′**	**AC**	**BC**	**SF**	**Speed**
ELEPHANT	true	2	1	5	3	2.7	22147
ELEPHANT$^+$	true	0	2	5	3	4	22098
ELEPHANT*	true	0	2	3	3	3	21635

The results of applying **BD-Encryption** and **BD-Decryption** functions are found in table 10, where we reported the minimum values of AC and BC each algorithm needs to pass these tests (under columns AC′ and BC′), together with the used values. The results show that ELEPHANT needs at least AC=2 and BC=1 to pass BD-Encryption and BD-Decryption functions, while ELEPHANT* and ELEPHANT$^+$ need only BC=2 to pass them and the SCC layer does the rest of the diffusion.

7 PERFORMANCE

We studied the performance of the optimized C versions of the ciphers. For the diffusers we used loop unrolling mechanism (Davidson and Jinturkar, 1995) and for the AES we used optimized Gladmann's implementation (Gladman, 2006). The results are listed in table 10 under column speed, note that the reported measurements are done on a PIV 3 GHz processor running on Windows Vista, where the programming environment was Microsoft VC++ 6. Here we report the number of clock cycles needed by each cipher, which is the minimum of 1000 iteration to remove any initial overheads or cache misses. The results show that ELEPHANT and ELEPHANT$^+$ possess the same speed, while ELEPHANT* is about 23% faster than them.

We define the Safety Factor (SF) in (3), which is the ratio between the total number of used diffusers' cycles over the minimum required. SF represents how safe is the current number of diffusers' cycles, under any circumstances this ratio should not be less than one. The values of SF are reported in table 10. These values show that ELEPHANT$^+$ and ELEPHANT* possess a higher SF than ELEPHANT.

$$SF = (AC + BC) \div (AC' + BC') \qquad (3)$$

From (Ferguson, 2006), suppose an attacker is attacking two identical hard drives, one encrypted with ELEPHANT and the other one encrypted with CBC. We are going to give the attacker the tweak key (K_{sec}), this means the attacker can now perform the diffusion layer for any plaintext. In other words, the diffuser layer becomes transparent to the attacker. All what is left now for the attacker is to attack the CBC layer, which is the same problem that he has when attacking the other hard drive (encrypted only using CBC). Although we helped the attacker significantly by providing him with the tweak key, he still have to attack the CBC layer. This shows that attacking ELEPHANT is not easier than attacking just CBC, and ELEPHANT is at least as secure as CBC.

By applying the same methodology, ELEPHANT$^+$ and ELEPHANT* are at least secure as SCC, however they possess a higher SF than ELEPHANT.

8 CONCLUSIONS

In this paper, we proposed a novel mode of operation for disk encryption applications. We analyzed this mode with the state of the art modes. Our proposed mode is superior than the other modes, as it possesses a high throughput. Although, it was designed based on the CBC mode, it can be parallelized and does not suffer from the bit-flipping attack. We used this mode to modify Windows Vista's encryption algorithm, to enhance some of its diffusion properties together with the ability to be partially parallelized.

REFERENCES

Anderson, R. and Biham, E. (1996). Two practical and provable secure block ciphers: BEAR and LION. In *Dieter Gollmann, editor, Fast Software Encryption: Third International Workshop (FSE'96)*.

Crowley, P. (2001). Mercy: a fast large block cipher for disk sector encryption. In *Bruce Schneier, editor, Fast Software Encryption: 7th International Workshop, FSE 2000*.

Daemen, J. and Rijmen, V. (1998). AES Proposal: Rijndael. http://citeseer.ist.psu.edu/daemen98aes.html.

Davidson, J. and Jinturkar, S. (1995). An Aggressive Approach to Loop Unrolling. Technical report, Department of Computer Science. University of Virginia. Charlottesville.

El-Fotouh, M. and Diepold, K. (2007). Statistical Testing for Disk Encryption Modes of Operations. Cryptology ePrint Archive, Report 2007/362.

El-Fotouh, M. and Diepold, K. (2008a). Dynamic Substitution Model. In *22nd Annual IFIP WG 11.3 Working Conference on Data and Applications Security (DBSEC'08)*, Naples, Italy.

El-Fotouh, M. and Diepold, K. (2008b). The Analysis of Windows Vista Disk Encryption Algorithm. In *The Fourth International Conference on Information Assurance and Security (IAS'08)*, London, UK.

Ferguson, N. (2006). 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.

Fluhrer, S. (2002). Cryptanalysis of the Mercy block cipher. In *Mitsuru Matsui, editor, Fast Software Encryption, 8th International Workshop, FSE 2001*.

Fruhwirth, C. (2005). New Methods in Hard Disk Encryption. http://clemens.endorphin.org/nmihde/nmihde-A4-ds.pdf.

Gladman, B. (2006). AES optimized C/C++ code. http: // fp.gladman.plus.com/ AES /index.htm.

IEEE P1619 Email Archive (2007). http:// grouper.ieee.org/ groups/ 1619/ email/ thread.html.

IEEE P1619 homepage (2007). Draft 18 for P1619: Standard Architecture for Encrypted Shared Storage Media. http://attachments.wetpaintserv.us/ Wbr7V2GY67Sxaxbw6ZFBeQ %3D%3D262488.

Liskov, M., Rivest, R., and Wagner, D. (2002). Tweakable Block Ciphers. In *CRYPTO '02: Proceedings of the 22nd Annual International Cryptology Conference on Advances in Cryptology*.

Lucks, S. (1996). BEAST: A fast block cipher for arbitrary blocksizes. In *Patrick Horster, editor, Communications and Multimedia Security II, Proceedings of the IFIP TC6/TC11 International Conference on Communications and Multimedia Security*.

May, L., Henricksen, M., Millan, W., Carter, G., and Dawson, E. (2002). Strengthening the Key Schedule of the AES. In *ACISP '02: Proceedings of the 7th Australian Conference on Information Security and Privacy*, pages 226–240, London, UK. Springer-Verlag.

McGrew, D. (2002). Counter Mode Security: Analysis and Recommendations. http://citeseer.ist.psu.edu/mcgrew02counter.html.

McGrew, D. (2004). PRP Modes Comparison IEEE P1619.2. http://grouper.ieee.org/ groups/1619/email/pdf00050.pdf.

Menezes, A., Oorschot., P. V., and Vanstone, S. (1996). *Handbook of Applied Cryptography*. CRC Press.

NIST (2007). Guide to Storage Encryption Technologies for End User Devices. http://csrc.nist.gov/publications/nistpubs/800-111/SP800-111.pdf.

Rogaway, P. (2003). Efficient Instantiations of Tweakable Block ciphers and Refinements to Modes OCB and PMAC. http://citeseer.ist.psu.edu/ rogaway03efficient.html.

Rogaway, P., Bellare, M., Black, J., and Krovetz, T. (2001). OCB: a block-cipher mode of operation for efficient authenticated encryption. In *ACM Conference on Computer and Communications Security*, pages 196–205.

Schroeppel, R. (1998). The Hasty Pudding cipher. The first AES conference, NIST.

Trusted Computing Group (2008). TCG TPM Specification Version 1.2. www. trustedcomputinggroup.org.