Fully Layerd Video Encryption Using Geomety Based Encryption Algorithm

line 1: 1st Given Name Surname
line 2: *dept. name of organization (of Affiliation)*
line 3: *name of organization (of Affiliation)*
line 4: City, Country
line 5: email address or ORCID

line 1: 2nd Given Name Surname
line 2: *dept. name of organization (of Affiliation)*
line 3: *name of organization (of Affiliation)*
line 4: City, Country
line 5: email address or ORCID

line 1: 3rd Given Name Surname
line 2: *dept. name of organization (of Affiliation)*
line 3: *name of organization (of Affiliation)*
line 4: City, Country
line 5: email address or ORCID

line 1: 4th Given Name Surname
line 2: *dept. name of organization (of Affiliation)*
line 3: *name of organization (of Affiliation)*
line 4: City, Country
line 5: email address or ORCID

line 1: 5th Given Name Surname
line 2: *dept. name of organization (of Affiliation)*
line 3: *name of organization (of Affiliation)*
line 4: City, Country
line 5: email address or ORCID

line 1: 6th Given Name Surname
line 2: *dept. name of organization (of Affiliation)*
line 3: *name of organization (of Affiliation)*
line 4: City, Country
line 5: email address or ORCID

Abstract –

Video encoding deals with the video encryption and compression. This paper proposes a symmetric block algorithm with key size of 128-bits to effectively encrypt the video files to make is secure and fast enough for transmission and storage. The proposed algorithm reads the video file as bytes and maps blocks of bytes to a point on a Geometrical 2-D plane. Using the key from the Code-Book, which is shared between the communicating parties the encrypted data can be decrypted at a fast rate as well. As the algorithm would be a Fully Layered Encryption technique it won't affect the MPEG

Introduction

With increasing bandwidth capabilities around the globe and several platforms providing support for live streaming, ensuring real-time security has become a matter of prime importance. The pace at which data is being generated and transmitted today has never been seen in the past. As a lot of this data is being generated by end users, data security at the individual level is much needed.

As far as text data is concerned, several algorithms such as AES, 3DES, Blowfish, etc are used for efficient and secure encryption. On the other hand, algorithms to encrypt multimedia(videos) are comparatively not as efficient. As multimedia needs to be encrypted, compressed, transmitted, decompressed, decrypted in real time, algorithms involved must provide higher throughputs. To accomplish this, several techniques are devised as discussed below.

1. Fully Layered Encryption: The video file is compressed and encrypted using standard algorithms like AES. Though this technique is quite secure, it is slow and therefore cannot be easily used in real-time applications.
2. Selective Encryption: Only some bytes of data are selectively encrypted using standard algorithms to allow real-time processing. This is less secure as some glimpses of images are found to be visible after encryption.
3. Permutation-based Encryption: A permutation list is used as a secret key to scramble some bytes of the video contents. This technique compromises on security as the data is not changed but only scrambled.
4. Perceptual Encryption: This technique partially degrades the quality of audio/visual data such that it is understandable but still the attacker would prefer for an authentic copy. The data after encryption is not really hidden but is just degraded hence the technique cannot be used for sensitive data.

Apart from the situational disadvantages discussed above, all techniques except the Fully layered Encryption are vulnerable to known plain text or/and known ciphertext attacks. Some of these techniques are not MPEG complaint and thus need specialized codecs (hardware or software). Thus, the literature survey suggests a need for a more secure algorithm that has better throughput. This paper proposes an algorithm that can be used for Fully Layered Encryption as it provides enough throughput to make it viable for real-time applications.

The algorithm proposed in this paper gives higher throughput, security and compatibility as it has the following salient features: *(i)* uses multiple keys to ensure security; *(ii)* encrypts bytes instead of frames so can be used for Fully Encryption technique in a MPEG compliant way; *(iii)* maps blocks of data to points in X-Y plane in one-to-many relationship to provide security against known plain-text and cipher-text attacks.

The paper discusses algorithm's compatibility with MPEG codecs in Section I, parameters affecting algorithms' security in Section II, core algorithm in Section III, discussion about padding, compression, parallelization in Section IV, performance analysis and attacks in Section V, and conclusion in Section VI.

I. COMPATIBILITY WITH MPEG CODECS

Several versions of MPEG codecs ranging from MPEG-1 being the first release to MPEG-4 AVC (H.264) being the latest one has been into practice. Over the years, the encoding techniques of these versions have evolved to provide better throughputs and compression. In fact, MPEG-4 AVC (H.264) has throughput enough to support high-quality streaming. Most Selective encryption algorithms need to be applied before or during the encoding process. Thus, Selective encryption technique can only use algorithms that can encrypt frames without interfering much with the encoding process. Whereas in the case of Fully layered encryption techniques, as can be observed in Fig 1, the data is encrypted after encoding and decrypted before decoding. Not only this makes the algorithm fully compliant with MPEG codecs but also it can be used with existing codec implementations.

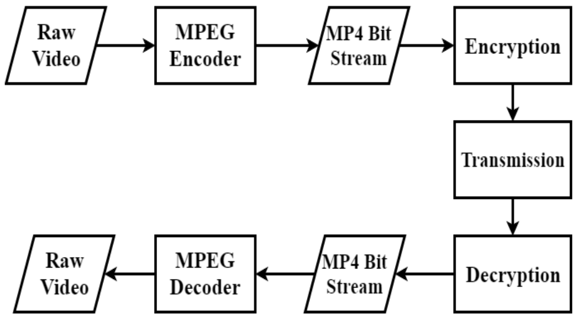


Figure 1: Codec Operation

II. PARAMETERS AFFECTING ALGORITHM

This algorithm needs the encrypting party and the decrypting party to agree upon some configuration options, as below.

Block (b): While encrypting a file, data is processed in chunks. This chunk of data is referred to as a block (b).

Block size (l_b): The size of the block is defined as block-size (l_b). When l=64, it means, 64bits of data is taken as an entity and encrypted to a give a point Q using key K.

Block Value (V): The value of a block, calculated on radix 10 and denoted by V. The algorithm does not manipulate bits of data, rather it encrypts the value of a block. Also, it is empirical that for a block of size l, the range of V can be described as,

$$0 \leq V < 2^{l} \qquad (1)$$

Key (K): A bit array used to encrypt a block of data. All keys are generated beforehand and shared between the communicating parties.

Key Length (l_k): The length of the key in bits. Ideally, a key of 128 or 256 bits is preferred as it offers enough security. The key length is related to block-size l_b. This relation can be defined as

$$l_k = 2 \times l_b$$

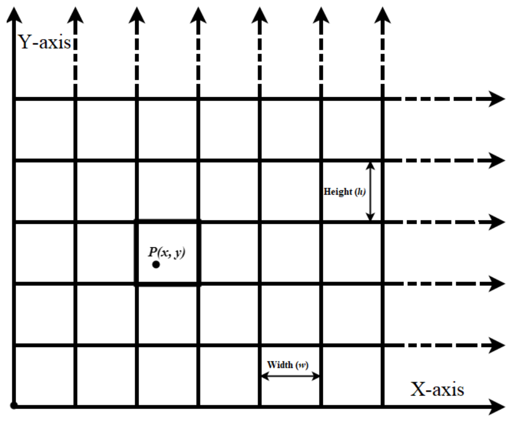


Figure 2 Grid

Grid (G): The algorithm uses multiple keys while encrypting a file. To facilitate easy and accurate access to these keys, they can be imagined to be distributed in a Grid(G), where G is a X-Y plane. Each cell in the grid has a different (not necessarily unique) key assigned to it. Any point P lying in a cell C of grid G will always use the key assigned to the cell it lies in. To further simplify, all cells are of uniform size. The key associated with any point is given by Key

Fetcher function KeyFetch(P) = K, where K is the key and P belongs to grid G. KeyFetch is further discussed in encryption section.

Grid-width(w_g) & Grid-height (h_g): The maximum possible value along X and Y axis are denoted as the grid-width (w_g) & grid-height (h_g) respectively.

Width (w) & Height (h): As each cell is of the same size, the width and height are uniform across the grid, denoted as *w* & *h* respectively. Also due to the nature of the algorithm, it should be noted that,

$w, h \in N$, where N is the set of Natural numbers.

x_{min} & y_{min}: To denote each cell, the minimum values of x and y are used. These values depend on the height(h) and width(w) of cells. The domain of x_{min} & y_{min} can be defined as

$$x_{min} \in \{0, w, 2w, 3w, 4w, \ldots, nw \mid nw < w_g\}$$

&

$$y_{min} \in \{0, h, 2h, 3h, 4h, \ldots, nh \mid nh < h_g\}$$

where,

w: width of cells
h: height of cells
w_g: width of the grid
h_g: height of the grid

Codebook (CB): A 2-D data structure indexed on x_{min} and y_{min} which stores the key associated with each cell.

III. Core Algorithm

The proposed algorithm encrypts a block value V to a point Q in X-Y plane using a key K. As the encrypted data is in form of points, to avoid reverse engineering the mapping relation needs to satisfy the following conditions.

(i) A key should be needed to get a block value from a point
(ii) The mapping should be a many-to-one relationship.

Block Value Mapper function (M):

To fulfil the aforementioned conditions with low computational complexity, following relation was devised. The relation M between a point P and block-value V can be generalized as

$$M(P) = V \qquad (2)$$

$$M_{(P)} = ((x_{fract} \bullet y_{fract}) \oplus K) \bmod 2^{l} \qquad (3)$$

Where,

P: a point in XY plane
K: KeyFetch(P)
l: block-size
x_{fract}: fractional part of x coordinate of P
y_{fract}: fractional part of y coordinate of P
V: Block value

This relation, denoted by M is the building block of algorithm. As can be observed from equation 3, M is a function of point P, key K and block-size *l*. The dependency on key ensures privacy after encryption and dependency on block-size provides many-to-one relationship between points and block-value. The many to one relationship also ensure that the key is secure even after known plain-text attacks.

3.1 Encryption

Encryption process takes a block of data and returns a point in X-Y plane. Encryption function E can be termed as, "If E is a relation on B × G, then (V, P) belongs to E if M(*P*,*K*,*l*)=V where G is the grid, B is set of all possible blocks-values of length *l* and M is the block value mapper function".

$$E(V) = P(x, y) \qquad (4)$$

such that, M(*P*) = V

Where,

V = block value

P = point in grid G

This implies, E is the inverse of M. Since M is a many-to-one relation, inverse of M is not possible. Thus, as a work-around "Point shifting" is applied while encrypting.

Point shifting is a process of shifting a given point P to get a point Q such that M(Q)=V and KeyFetch(P) = KeyFetch(Q), where P, Q belong to grid G and V is the desired block value.

Also, it is necessary that same key is used throughout shifting process i.e., KeyFetch(P) should be equal to KeyFetch(Q). This can be done by changing only fractional part of coordinates of P while shifting, and using only integer part while key Fetching. This is also the reason why Equation 3 uses fractional values of *x* and *y*.

3.1.1 Key Fetching

As all the points in a cell C always use the key associated with C, for a Grid with cell width *w* and cell height *h* using the codebook (CB) , key can be fetched as

$$K = CB\,[x_{int} - (x_{int} \bmod w)]\,[y_{int} - (y_{int} \bmod h)]$$

Function 1: *Key Fetching*

```
Function fetchKey(Xint, Yint, Codebook)
    Xmin ← Xint – (Xint mod w)
    Ymin ← Yint – (Yint mod h)
    Key ← Codebook [Xmin][Ymin]
    Return Key
```

3.1.2 Point Shifting:

Point shifting is a process of shifting a given point P to get a point Q such that M(Q)=V and KeyFetch(P) = KeyFetch(Q), where P, Q belong to grid G and V is the desired block value. The point Q can be derived in following way

The fractional part of the *x* and *y* are concatenated (represented by the symbol ●) and then an *XOR* operation (represented by the symbol ⊕) is performed on them with the key. Length of *α* is going to be the same as that of the key.

$$\alpha = x_{fract} \bullet y_{fract} \qquad (5)$$

$$\beta = \alpha \oplus Key \qquad (6)$$

The *α* value needs to be adjusted such that it represents the block value to be encoded. So, on adding the offset to *β* and again operating an *XOR* operation on it with the key, it gives a value *α*', which is then split into two values.

$$\gamma = \beta \bmod 2^l \qquad (7)$$

$$offset = \gamma - V \qquad (8)$$

$$\beta' = \beta + offset \qquad (9)$$

$$\alpha' = \beta' \oplus Key \qquad (10)$$

The first half represents x'_{fract} and the second half represents y'_{fract}. These bits have the encoded data in them.

$$\alpha' = x'_{fract} \bullet y'_{fract} \qquad (11)$$

The x'_{fract} and y'_{fract} values are now merged with the integer parts of *x* and *y* giving a point Q.

$$P(x_{int}, x_{fract}, y_{int}, y_{fract}) \rightarrow Q(x_{int}, x'_{fract}, y_{int}, y'_{fract})$$

Function 2: *Point Shifting*

```
Function shiftPoint(Xfract,Yfract,V,key, l)
    A ← (Xfract << l) +Yfract
    B ← A XOR key
    C ← B mod exponent(2, l)
    if C > V
        offset ← exponent(2, l) - C + V
    else
        offset ← V-C
    endif
    B' ← B + offset
    A' ← B' XOR key
    X'fract ← A' >> l
    Y'fract ← A' & (exponent(2, l)-1)
    Return X'fract, Y'fract
```

Using point shifting and key fetching, encryption can be done in following steps as shown in figure

1) Select a random point P on grid G
2) Fetch the key associated with point P using codebook
3) Shift the point P using the fetched key K and block value V, to get a point Q.
4) Point Q is the output of the process.

Algorithm: *Block Encryption*

```
Function EncryptBlock(V, Codebook, l)
    Xint ← Random integer
    Xfract ← Random l bit integer
    Yint ← Random integer
    Yfract ← Random l bit integer
    Key ← fetchKey(Xint,Yint,Codebook)
    X'fract, Y'fract ← shiftPoint (Xfract,
        Yfract, V, Key, l)
    Return Xint, X'fract, Yint, Y'fract
```

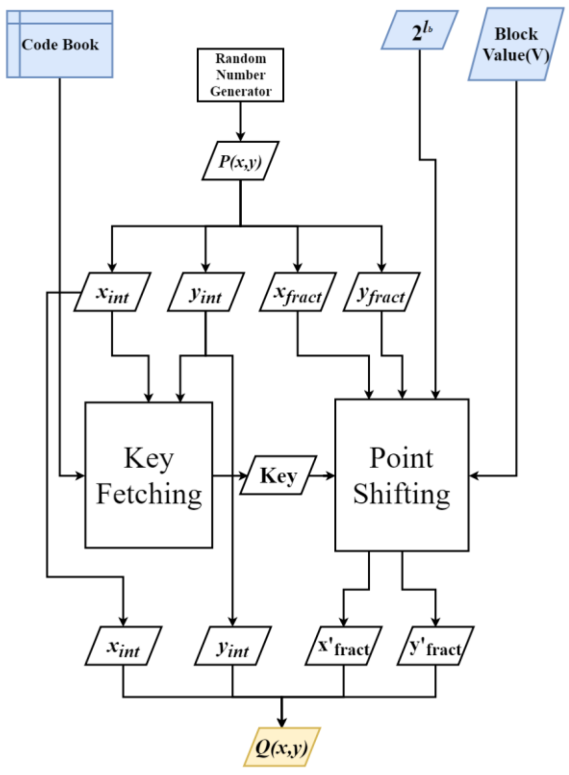


Figure 3 Encryption process

3.2 Decryption –

Decryption process takes a point in XY plane and returns back a block of data. Decryption function D can be termed as, "If D is a relation on G × B, then (P, V) belongs to D if M (P, K, l) =V where G is the grid, B is set of all possible blocks-values of length l and M is the block value mapper function".

D(P) = V, such that M(P) = V

$$D(P) = M(P) \tag{12}$$

Where,

V = block value

P = point in grid G

This implies that, relation D is same as relation M.

The decryption process starts with the encrypted point (Q) as input. As the integer portion of the point does not change while encryption, the key-fetching for decryption can be done in the same way as in the encryption process. Once the key is fetched, the block value can be retrieved from the fractional portions of Point $Q(x_{int},x'_{fract},y_{int},y'_{fract})$ by backtracking.

$$\alpha = (x'_{fract} \bullet y'_{fract}) \tag{13}$$

$$\beta = \alpha \oplus Key \tag{14}$$

$$V = \beta \bmod 2^{l} \tag{15}$$

V here is the block value that the sender intended to send to the receiver. The block bits can be obtained by formatting V to l bits size with leading zeroes.

Algorithm: Decryption

```
Function DecryptBlock(Xint, Xfract,
Yint, Yfract, Codebook, l)
    Key ← fetchKey(Xint,Yint,Codebook)
    A ← (Xfract << l) + Yfract
    B ← A XOR Key
    V ← B mod exponent(2, l)
  Return V
```

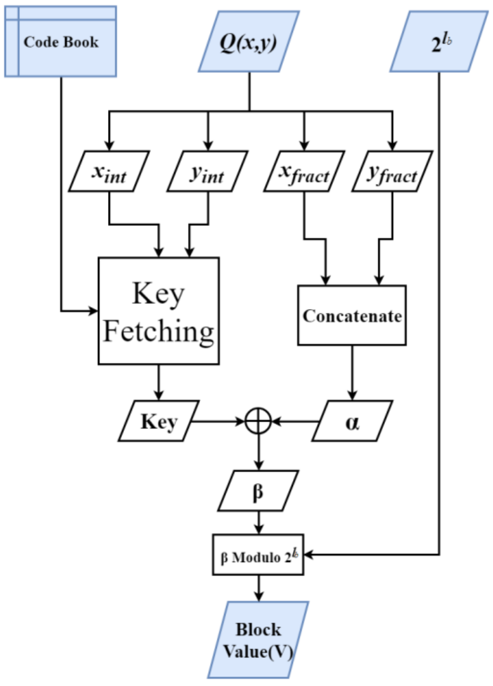
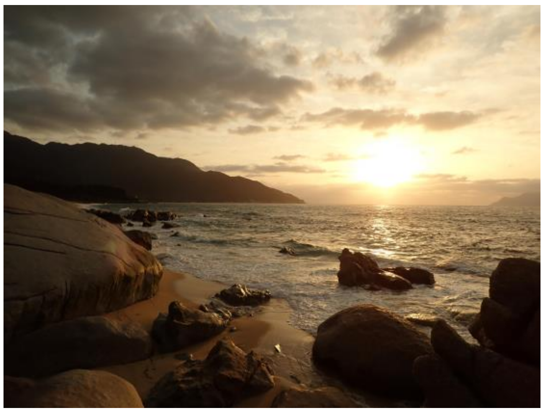
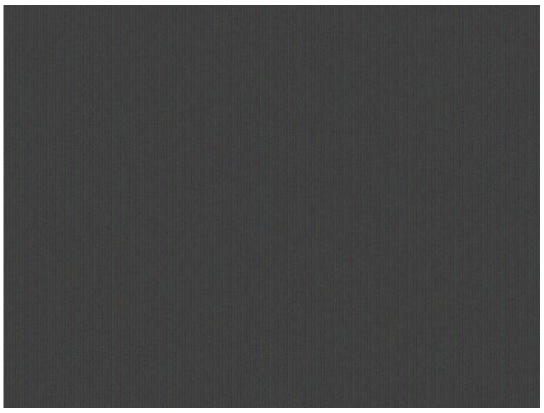
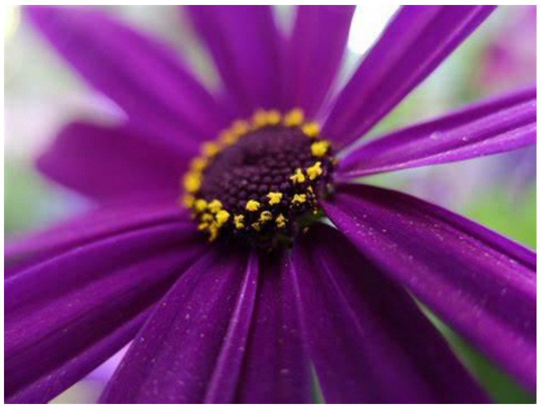
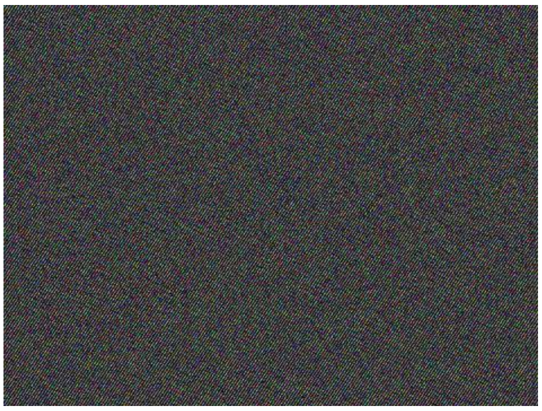
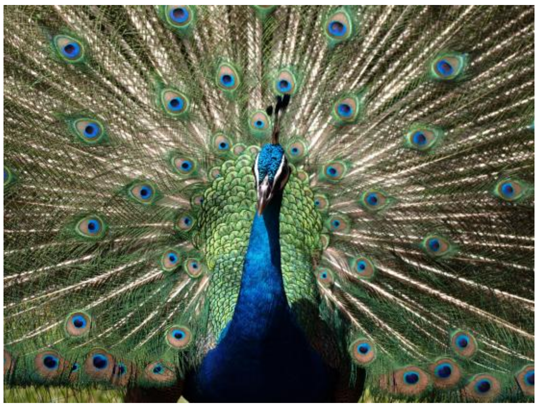
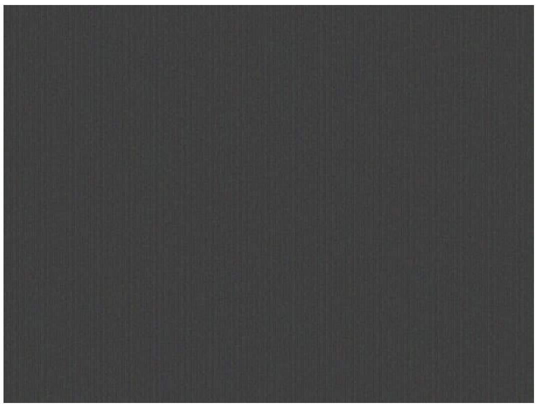


Figure 4: Decryption Process

Table 1 A comparison of original images and images encrypted using proposed Algorithm

ORIGINAL IMAGE	ENCRYPTED IMAGE
	
	
	

IV. Experiment

In order to test the quality of encryption at a preliminary level, a few images were encrypted using 64bit block-size. Once encrypted they were again visualized using PIL library of python. As can be seen in table 1, the encrypted images looked only like white noise. Although this experiment does not say much about the quality of encryption as that can only be thoroughly tested by cryptanalysis techniques, it can still state that, no traces of original image can be observed in encrypted image.

V. Discussion

5.1 Padding

In an ideal case, each file will have file size divisible by a block of size; but in real time scenario, this is rarely the case. For the algorithm to work on any file size, padding needs to be appended to the encrypted file. In this case, the padding is the number of bytes in the last block.

For example, let's consider a file size of 1029 bytes to be encrypted with a block size of 8 bytes (l = 64 bits), then the last block will have 5 bytes of data. This makes the padding value to be 5. This value is added at the beginning of the encrypted file. So, during decryption when the encrypted file is read, and this value is used to cut short the last block of the decrypted data appropriately.

5.2 *Random Number Generator*

In any encryption algorithm, randomness/entropy is of prime importance. Many implementations of Random number generators are not purely random and so are prone to attacks. While key and point generation, it is necessary that proper randomness is ensured. During testing of the proposed algorithm, Python's Secret library is used for the key generation and the boost library of C++11 is used to get the Random Points.

5.3 *Parallelization*

As each block uses a distinct key, there is no need for chaining as is in AES and other algorithms. The encryption process of each block is isolated from other blocks. This makes the algorithm highly parallelizable. While testing, OpenMP of C++ was used for parallelizing the algorithm which increased the performance two times on a dual-core CPU.

5.4 *Compression*

On using the proposed algorithm, the size of the encoded file becomes thrice. On encryption of the encoded video file, the blocks of data change to points in the X-Y plane, making the encrypted file viable to be compressed again. As the paper discusses video streaming, the compression algorithm to be used should be able to efficiently compress smaller chunks of data with an adequate throughput. While testing, library Z-std was used to compress the encrypted file.

VI. Performance Analysis and Attacks

6.1 *Performance Measure*

As discussed above, the algorithm is parallelizable this makes the algorithm computationally fast. When tested on a machine with 64-bit, i7 processor with 4 logical processors, 6500U CPU @ 2.5GHz, 8GB RAM, the results obtained were as shown in Table 2.

While testing, file sizes of 1KB to 10^6 KB (~100MB) were encrypted several times to calculate an average performance. The second column in table 2 describes the maximum number of keys that the algorithm will take for encrypting the file once. Thus, for a 1KB file, around 128 keys each of size 128-bits will be used.

Table 1: Performance Analysis for l = 64-bit

File Size (KB)	Max No. of keys used per iterations	No. of iterations	Total Time Taken	Performance (KB/s)
1	128	100000	19.72	5069
10	1280	10000	16.70	5987
100	12800	1000	15.98	6256
1000	128000	100	16.15	6191
10000	1280000	10	16.38	6103
100000	12800000	1	16.97	5890

As Fig. 7 and 8 depict, the decryption performance is better than encryption performance. Though there is not a major difference, but as file size increases the difference becomes observable.

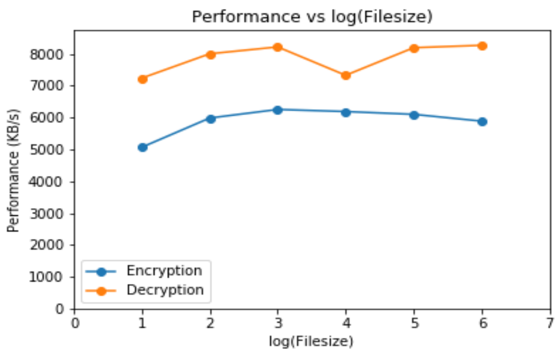


Figure 5: Performance Measure where l = 64

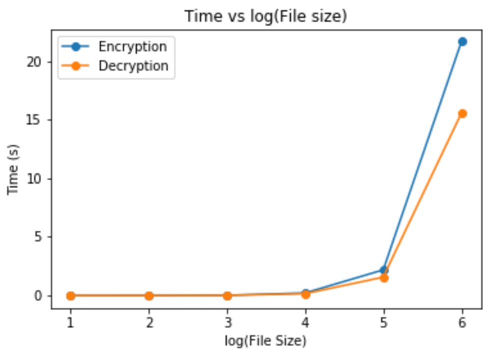


Figure 6: Time taken vs $\log_{10}$(file size)

Table 2: Performance (KB/s) when l is 32 and 64

Block Size	32-bit	64-bit
Encryption	4393.69	4589.32
Decryption	6228.82	6064.17

The performance with 32-bit and 64-bit block size on a 64-bit, i5 processor with 4 logical processors, 5200U CPU @ 2.2GHz, 8GB RAM were as shown in table 3. As we can see that increasing the block size does not affect much on the performance but increase the security two times.

5.2 Diehard Test

Diehard tests are a set of statistical tests for measuring the quality of a random number generator. Though they are targeted to test Pseudo Random Number Generators (PRNGs), they can be used to test ciphers too. It should be noted that instead of testing whether a sequence is random, Diehard suite tests if it is not random. A random number sequence that 'fails to fail' diehard test may not be random. Thus, passing the Diehard tests can be considered a necessary but not sufficient condition for a sequence to be random.

Diehard uses chi-squared goodness-to-fit technique to calculate the p-value. These p-value range from 0 to 1. Closer the value is to 0 or 1 more is the deviation from randomness for a sequence. So, a p-value of 0.5 is considered ideal for a sequence to be random. The proposed algorithm passed all the tests with enough randomness indicated by the p-values. The original data files have p-values zero indicating that they don't have any randomness, which is the expected result.

Test Name	**p-value**
Birthday Spacing	0.24847631
OPERM5	0.47522383
Binary Rank Test 32x32	0.33404450
Binary Rank Test 6x8	0.57719925
Bitstream Test	0.92223519
Overlapping-Pairs-Sparse-Occupancy Test	0.87421383
Overlapping-Quadruples-Sparse-Occupancy Test	0.88129995
DNA Test	0.79204178
Count-the-1's Stream Test	0.64120560
Count-the-1's Byte Test	0.39412008
Parking Lot test	0.18589551
2-d Sphere (minimum distance test)	0.27793887
3-d Sphere (minimum distance test)	0.12818751
Squeeze Test	0.18190138
Overlapping Sums Test	0.24031421
Diehard Runs Test	0.21269547
	0.03665191
Diehard Craps Test	0.34554749
	0.87519848
Marsaglia and Tsang GCD Test	0.03018864
	0.00984652
STS Monobit Test	0.78929803
STS Runs Test	0.30018053

5.3 *Attacks*

5.3.2 *Brute Force Attack*

For a 64-bit block size, the key size is 128 bits. This makes the total number of possible keys to be 2^{128} which is around 3.4 x 10^{38}. Furthermore, each cell has its own 128-bit distinct key assigned to it, which makes the number of possible combinations to be,

$$n^{2^{128}}$$

Where *n* is the number of total cells in the grid G. Hence making the computational complexity of breaking the algorithm very high.

5.3.2 *Known Plain Text Attack*

As each block is mapped on different points of different cells which have different keys associated with them, known plain-text attacks should also not be able to predict the key. The algorithm will be only partially breached even if many keys are predicted somehow. As all the blocks can be mapped to any of the cells therefore, they cannot be associated with any specifically defined region of the Grid.

VI. Conclusion

The algorithm proposed, was able to provide a throughput of 5916 KB/s (~5.8 MB/s) and can be used with existing MPEG codecs. It also conforms the requirements for video encryption and is also suitable for video streaming and storing. The algorithm has been successfully implemented with a key size of 128 bits which can be extended if needed, though it is secure enough now. For future work, Message Authentication Code (MAC) can be added to fulfil the integrity, authentication, non-repudiation and other aspects of information security.

VII. References