

# Fully Layered Video Encryption Using Geometry Based Encryption Algorithm

line 1: 1<sup>th</sup> 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: 2<sup>th</sup> 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: 3<sup>th</sup> 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: 4<sup>th</sup> 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: 5<sup>th</sup> 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: 6<sup>th</sup> 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**—With increasing trends of live streaming multimedia, real-time security of data has become of prime importance. Due to low performance, existing encryption algorithms can only be used to selectively encrypt a few parts of data. This paper proposes a high-performance symmetric block encryption algorithm based on the coordinate system, that can be used with Fully Layered Video encryption technique. This ensures that the data is concealed. The algorithm can also be used with existing encoders as a post-processing step, making it MPEG compliant. The algorithm has been successfully implemented with a key size of 128 bits which can be extended if needed. The proposed algorithm provides a throughput high enough for multimedia streaming

**Keywords**—fully layered video encryption, video encryption, symmetric encryption, coordinate system

## I. 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. A. Tosun *et. al.* [1] states that the acceptance or rejection of a commercial platform that offers video and audio-based services is determined by its efficiency and on how secure the encryption algorithm is. There are algorithms that secure these streams but require significant processing power, making it not ideal.

As far as text data is concerned, several algorithms - such as AES, 3DES, Blowfish - are used for efficient and secure encryption.[2][3][4] However, algorithms to encrypt multimedia(videos) are comparatively not as efficient. As multimedia needs to be encrypted, compressed, transmitted, decompressed, decrypted in real time, the algorithms involved must provide higher throughputs. As described by J. Shah *et. al.* [5] to accomplish this, several techniques are devised as follows.

1. Fully Layered Encryption: The video file is compressed and encrypted using standard algorithms such as 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 security as the data is not changed, but only scrambled.
4. Perceptual Encryption: This technique partially degrades the quality of audio/visual data with the result that it is understandable but still the attacker would prefer for an authentic copy. After encryption the data is not actually hidden, instead is degraded. Hence, this technique cannot be used for sensitive data.[6]

In addition to the situational disadvantages discussed above, most of the techniques – with the exception of the Fully Layered Encryption - are vulnerable to known plain text or/and known ciphertext attacks. Moreover, some of these techniques are not MPEG compliant, thus they require specialized codecs (hardware or software). In this regard, the literature surveyed 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. J. Harsha *et. al.* [7] showed the possibility of using the polar and the cartesian planes as a basis for cryptosystem and gave an asymmetric algorithm that was file-type, size, and platform independent in nature. It surpassed most of the presently used cryptosystems based on the Weighted Space-Time-Complexity index. The proposed algorithm in this paper also uses the cartesian plane as reference for mapping data values to random points. Elliptic Curve Cryptography (ECC) is another asymmetric algorithm that uses the cartesian system. Although its trapdoor function is quite efficient, it requires a good source of entropy and there are a few bad curves that produce unprotected values. [8][9][10]

The algorithm proposed in this paper gives higher throughput, security, and compatibility as it has the following salient features: (i) it uses multiple keys to ensure security; (ii) it encrypts bytes instead of frames, therefore, it can be used for Fully Encryption technique in a MPEG compliant

way; (iii) it 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 II; parameters affecting algorithms' security in Section III; core algorithm in Section IV; padding, compression, and parallelization in Section V; performance analysis and attacks in Section VI; and the conclusion in Section VII.

## II. . COMPATIBILITY WITH MPEG CODECS

As explained by I. Agi *et. al.* [11] a MPEG codec compresses a continuous video stream to a serial bit stream which can be transmitted over a channel. Several versions of MPEG codecs ranging from the first release, MPEG-1, to the latest one, MPEG-4 AVC (H.264), have been put 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 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. This not only makes the algorithm fully compliant with MPEG codecs, but it can also be used with existing codec implementations.[12][13][14]

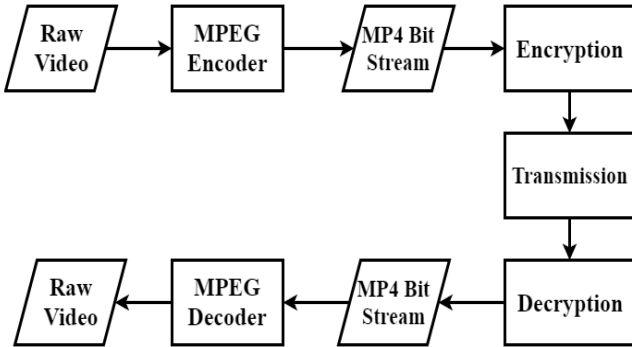


Figure 1: Codec Operation

## III. PARAMETERS AFFECTING ALGORITHM

This algorithm needs the encrypting party and the decrypting party to agree upon some configurations. The security of an algorithm depends on these configuration options. Also, all these options are either specified in the codebook or can be derived from other parameters.

**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,

$$0 \leq V < 2^{l_b} \quad (1)$$

**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

$$l_k = 2 \times l_b$$

**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 an 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  $\text{KeyFetch}(P) = K$ , where K is the key and P belongs to grid G. KeyFetch is further discussed in the encryption section.

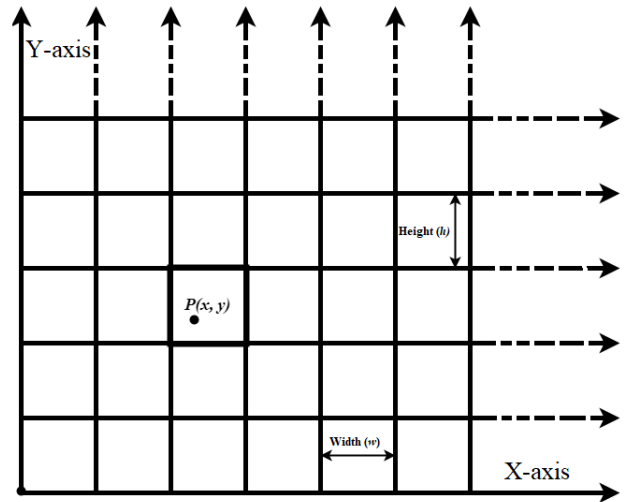


Figure 2: Grid

**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,

$w, h \in \mathbb{N}$ , where  $\mathbb{N}$  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

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

&

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

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.

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

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

$$M(P) = ((x_{fract} \bullet y_{fract}) \oplus K) \bmod 2^{l_b} \quad (3)$$

Where,

$P$ : a point in XY plane

$K$ : KeyFetch( $P$ )

$l_b$ : block-size

$x_{fract}$ : fractional part of the  $x$  coordinate of  $P$

$y_{fract}$ : fractional part of the  $y$  coordinate of  $P$

$V$ : Block value

This relation denoted by  $M$  is the building block of the 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 a 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.

##### A. Encryption

The encryption process takes a block of data and returns a point in the X-Y plane. Encryption function  $E$  can be termed as, "If  $E$  is a relation on  $B \times G$ , 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".

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

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 workaround, "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  $\text{KeyFetch}(P) = \text{KeyFetch}(Q)$ , where  $P, Q$  belong to grid  $G$  and  $V$  is the desired block value.

Also, it is necessary that the same key is used throughout the shifting process i.e.,  $\text{KeyFetch}(P)$  should be equal to  $\text{KeyFetch}(Q)$ . 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  $y$ . Hence the proposed algorithm can broadly be divided into two phases, the key fetching, and the point shifting.

##### 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), the key can be fetched as

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

##### **Function 1: Key Fetching**

*Function fetchKey( $X_{int}, Y_{int}, \text{Codebook}$ )*

$X_{min} \leftarrow X_{int} - (X_{int} \bmod w)$

$Y_{min} \leftarrow Y_{int} - (Y_{int} \bmod h)$

$\text{Key} \leftarrow \text{Codebook} [X_{min}][Y_{min}]$

*Return Key*

##### 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  $\text{KeyFetch}(P) = \text{KeyFetch}(Q)$ , where  $P, Q$  belong to grid  $G$ , and  $V$  is the desired block value.

The fractional part of the  $x$  and  $y$  are concatenated (represented by the symbol  $\bullet$ ) and then an  $XOR$  operation (represented by the symbol  $\oplus$ ) is performed on them with the key. Length of  $\alpha$  is going to be the same as that of the key.

$$\alpha = X_{fract} \bullet Y_{fract} \quad (5)$$

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

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

$$\gamma = \beta \bmod 2^{l_b} \quad (7)$$

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

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

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

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

$$\alpha' = X'_{fract} \bullet Y'_{fract} \quad (11)$$

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

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

Using point shifting and key fetching, encryption can be done in the following steps as shown in Fig. 3

- 1) Select a random point  $P$  on grid  $G$
- 2) Fetch the key associated with point  $P$  using a 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.

Function 2: Point Shifting
<p>Function <math>shiftPoint(X_{fract}, Y_{fract}, V, key, l_b)</math></p> <p><math>\alpha \leftarrow (X_{fract} \ll l) + Y_{fract}</math></p> <p><math>\beta \leftarrow \alpha \text{ XOR } key</math></p> <p><math>\gamma \leftarrow \beta \bmod exponent(2, l_b)</math></p> <p>if <math>\gamma &gt; V</math></p> <p style="padding-left: 20px;"><math>offset \leftarrow exponent(2, l_b) - \gamma + V</math></p> <p>else</p> <p style="padding-left: 20px;"><math>offset \leftarrow V - \gamma</math></p> <p>endif</p> <p><math>\beta' \leftarrow \beta + offset</math></p> <p><math>\alpha' \leftarrow \beta' \text{ XOR } key</math></p> <p><math>X'_{fract} \leftarrow \alpha' \gg l</math></p> <p><math>X'_{fract} \leftarrow \alpha' \&amp; (exponent(2, l_b) - 1)</math></p> <p>Return <math>X'_{fract}, Y'_{fract}</math></p>

#### Algorithm: Block Encryption

Function  $EncryptBlock(V, Codebook, l_b)$

$X_{int} \leftarrow \text{Random integer}$

$X_{fract} \leftarrow \text{Random } l_b \text{ bit integer}$

$Y_{int} \leftarrow \text{Random integer}$

$Y_{fract} \leftarrow \text{Random } l_b \text{ bit integer}$

$Key \leftarrow \text{fetchKey}(X_{int}, Y_{int}, Codebook)$

$X'_{fract}, Y'_{fract} \leftarrow \text{shiftPoint}(X_{fract}, Y_{fract}, V, Key, l_b)$

Return  $X_{int}, X'_{fract}, Y_{int}, Y'_{fract}$

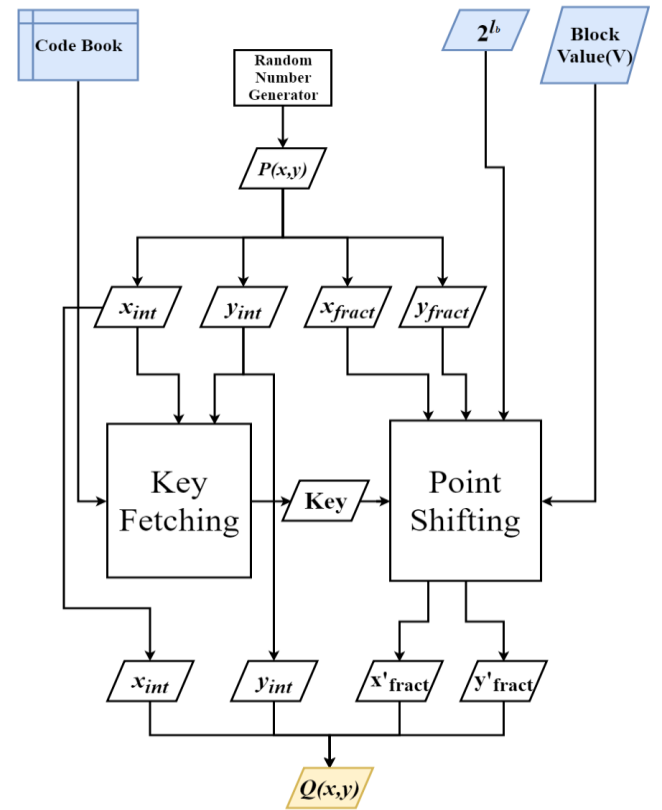


Figure 3: Encryption process

#### B. Decryption –

Decryption process takes a point in the X-Y plane and returns a block of data. Decryption function  $D$  can be termed as, “If  $D$  is a relation on  $G \times B$ , 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”.

$$D(P) = V, \text{ such that } M(P) = V$$

$$D(P) = M(P) \quad (12)$$

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 encrypting, 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.

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

<b>Algorithm: Decryption</b>				
<i>Function</i> $DecryptBlock(x_{int}, x_{fract}, y_{int}, y_{fract}, Codebook, l_b)$				
$Key \leftarrow fetchKey(x_{int}, y_{int}, Codebook)$				
$\alpha \leftarrow (x_{fract} \ll l_b) + y_{fract}$				
$\beta \leftarrow \alpha \text{ XOR } Key$				
$V \leftarrow \beta \text{ mod } exponent(2, l_b)$				
<i>Return</i> V				

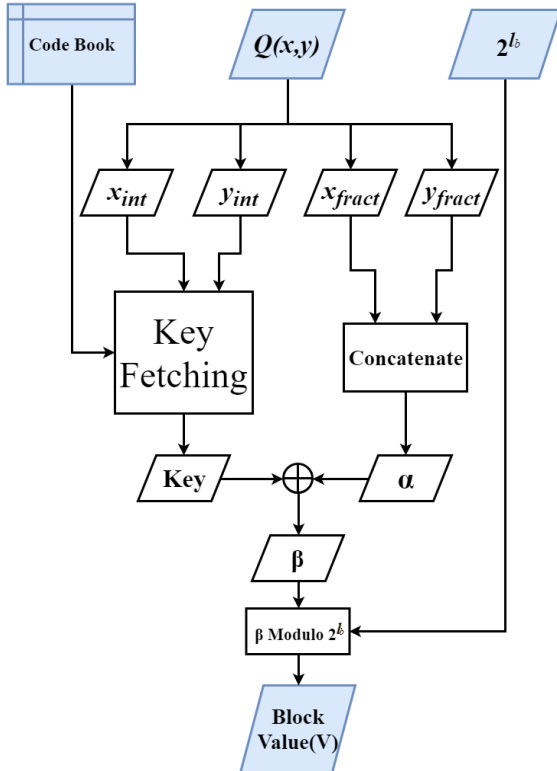


Figure 4: Decryption Process

## V. DISCUSSION

### A. 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, 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. Consequently, when the encrypted file is read during decryption, this value is used to appropriately cut short the last block of the decrypted data.

### B. 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 generating key and point, it is necessary that proper randomness is ensured. In testing 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.

### C. 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.

### D. Compression


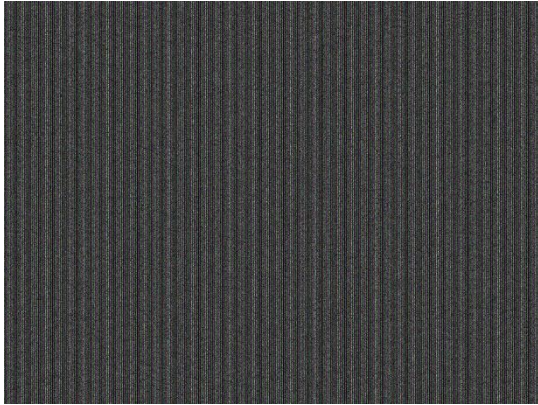



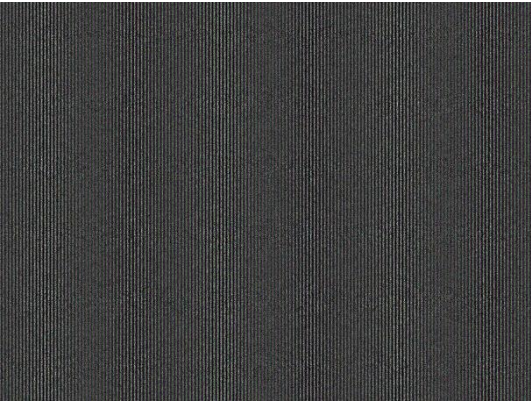
On using the proposed algorithm, the size of the encoded file becomes thrice. On encryption of the encoded video file, the blocks of data change to points in the X-Y plane, making the encrypted file viable to be compressed again. As the paper discusses video streaming, the compression algorithm to be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

## VI. 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. As can be seen in Table 1: A comparison of original Images and images encrypted using the proposed algorithm, the encrypted images are white noise images. Although this experiment does not provide sufficient evidence on the quality of encryption as that can only be thoroughly tested by cryptanalysis techniques, it still attests that no traces of the original image can be observed in the encrypted image. Also, in order to quantify, the structural similarity of encrypted and decrypted images was compared to the original images. As can be observed from Table 2 the similarity after decryption is 100%, whereas it is only about 2% in the case of the encrypted image.



*Table 1: A comparison of original Images and images encrypted using the proposed algorithm*

ORIGINAL IMAGE	ENCRYPTED IMAGE
	
	
	

*Table 2: Structural Similarity of Plain and Cipher Images*

Test Image	Original	Encrypted	Decrypted
Beach	1	0.01207	1
Flower	1	0.01233	1
Peacock	1	0.01280	1

## VII. PERFORMANCE ANALYSIS AND ATTACKS

### A. Performance Measure

As discussed above, the algorithm is parallelizable which 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 3: *Performance (KB/s) when  $l_b$  is 32 and 64*.

While testing, file sizes of 1KB to 106 KB (~100MB) were encrypted several times to calculate an average performance. The second column in Table 3 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.

As Fig. 5 and 6 depict, the decryption performance is better than the encryption performance. Though there is not a major difference, as file size increases the difference becomes observable. 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 4 and Table 5. As may be observed, increasing the block size does not affect the performance notably, but it increases the security twofold.[15]

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

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

File Size (KB)	Max No. of keys used per iterations	No. of iterations	Total Time Taken	Performance (KB/s)
1	128	100000	13.81	7240
10	1280	10000	12.49	8005
100	12800	1000	12.17	8219
1000	128000	100	13.64	7329
10000	1280000	10	12.20	8196
100000	12800000	1	12.09	8271

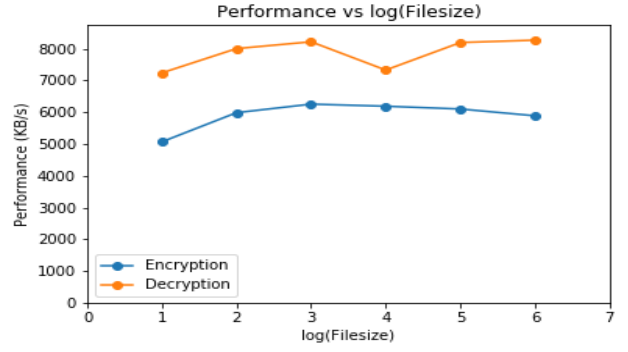


Figure 5: Performance Measure where  $l = 64$

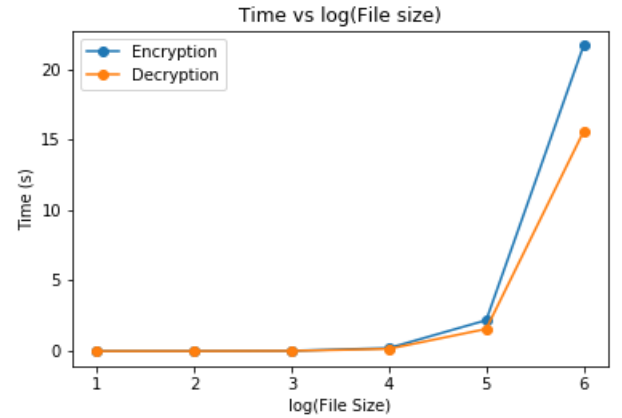


Figure 6: Time taken vs  $\log_{10}(\text{file size})$

### B. 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. [16]

Diehard uses the chi-squared goodness-to-fit technique to calculate the p-values. These p-values 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.[17]

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

### C. Attacks

#### 1) 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  $3.4 \times 10^{38}$ . 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.

#### 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.

### VIII. CONCLUSION

The paper proposes a high-performance algorithm that uses points in the cartesian plane to encrypt a multimedia file in real-time. The algorithm proposed, was able to provide a throughput of 5916 KB/s (~5.8 MB/s) while encryption and 7877 KB/s (~7.7MB/s) while decryption. Further, it can also 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 fulfill the integrity, authentication, non-repudiation and other aspects of information security.

### REFERENCES

- [1] A. S. Tosun and W.-C. Feng, "Efficient multi-layer coding and encryption of MPEG video streams," 2002.
- [2] D. Gullasch *et al.*, "197: Announcing the advanced encryption standard (AES)," ... *Technol. Lab. Natl. Inst. Stand.* ..., 2001.
- [3] W. Diffie, W. Diffie, and M. E. Hellman, "New Directions in Cryptography," *IEEE Trans. Inf. Theory*, 1976.
- [4] B. Schneier, "Description of a new variable-length key, 64-bit block cipher (Blowfish)," 2012.
- [5] J. Shah and V. Saxena, "Video Encryption: A Survey," *Int. J. Comput. Sci. Issues*, 2011.
- [6] B. Bhargava, C. Shi, and S. Y. Wang, "MPEG video encryption algorithms," *Multimed. Tools Appl.*, 2004.
- [7] H. S. Jois, N. Bhaskar, and M. N. Shesha Prakash, "A 3-d advancement of PythoCrypt for any file type," *J. Open Innov. Technol. Mark. Complex.*, 2015.
- [8] J. W. Bos, J. A. Halderman, N. Heninger, J. Moore, M. Naehrig, and E. Wustrow, "Elliptic curve cryptography in practice," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2014.
- [9] N. P. Smart, "The discrete logarithm problem on elliptic curves of trace one," *J. Cryptol.*, 1999.
- [10] R. Bhanot and R. Hans, "A review and comparative analysis of various encryption algorithms," *Int. J. Secur. its Appl.*, 2015.
- [11] I. Agi and L. Gong, "An empirical study of secure MPEG video transmissions," in *Proceedings of Internet Society Symposium on Network and Distributed Systems Security*, pp. 137–144.
- [12] S. Lian, J. Sun, G. Liu, and Z. Wang, "Efficient video encryption scheme based on advanced video coding," *Multimed. Tools Appl.*, 2008.
- [13] F. Liu and H. Koenig, "A survey of video encryption algorithms," *Comput. Secur.*, vol. 29, no. 1, pp. 3–15, Feb. 2010.
- [14] L. M. Varlakshmi, G. F. Sudha, and G. Jaikishan, "An efficient scalable video encryption scheme for real time applications," in *Procedia Engineering*, 2012.
- [15] D. S. Abd Elminaam, H. M. A. Kader, and M. M. Hadhoud, "Evaluating the performance of symmetric encryption algorithms," *Int. J. Netw. Secur.*, 2010.
- [16] G. Marsaglia and W. W. Tsang, "Some Difficult-to-Pass Tests of Randomness," *J. Stat. Softw.*, 2015.
- [17] M. M. Alani, "Testing Randomness in Ciphertext of Block-Ciphers Using DieHard Tests," *J. Comput. Sci.*, 2010.