A Two-Dimensional Key Vector Based Fast and Lightweight Block Cipher for Constrained Devices

Shirisha Kakarla[1], Geeta Kakarla[1] and S. Phani Kumar[2]

[1] Dept. of CSE (Cyber Security), (Cyber Security), SNIST, Hyderabad
[2]Dept. of CSE, GITAM (Deemed to be University)Hyderabad

Abstract Gathering sensitive patient information is vital in healthcare for providing the proper therapy. The collected data must also be shared with the relevant parties in order to keep the environment open and honest. There is still a significant risk of patient data leakage and misuse for unethical purposes, even with privacy protection mechanisms in place. As a result, we are suggesting a lightweight encryption solution for storing and sharing sensitive data with all parties involved. The data may be easily decrypted and retrieved by the stakeholders who also have the key. Compared to classical encryption approaches, the computing costs are reduced. To illustrate the regulating cryptographic operations, this article uses a 2D key bunch vector with 2048 binary bits. In lesser computer systems, the key length can be conveniently extended with relatively small changes to the computational outlays. In contexts with limited resources, the problem of creating an effective block cypher using the larger key is tackled.

1. Introduction

Concerns about patients' right to privacy and confidentiality have emerged as a major obstacle to the free flow of medical records in the digital age [1, 2, 3]. The health-care industry is not immune to the seismic shifts caused by the widespread use of computers and electronic records. Security is a growing problem because to the widespread use of digital devices by healthcare personnel. These devices are used to record, analyse, and share patients' medical conditions with other stakeholders in the healthcare ecosystem. This helps to successfully provide treatment. Health information systems are becoming more secure as both developed and emerging countries implement new security measures to protect patient privacy and data integrity [4, 5, 6].(1), (2), and (3). Research indicates that healthcare organisations in industrialised nations conduct regular evaluations to identify and fix security vulnerabilities [7]. When security flaws and vulnerabilities are identified, the system and information security teams periodically (re)model the architecture to fix them. Following a series of security solutions that aim to reduce vulnerabilities and provide

deterrent against cyber assaults and integrity violations, the optimised security method is chosen [8]. With larger parametric inputs, many security solutions rely on strong and sophisticated encryption methods [9] Actually, the confidentiality and privacy of digitized health records is no less important in low resource settings, because the internet is global in nature. Utilization of strong security solutions is not practicable [10] because they require a high-end computing infrastructure to install and maintain. Keys and blocks of encrypted input data are the building blocks of any cryptographic system. An improved level of data security is achieved by increasing the length of both the cryptic key and the input data. To implement privacy and security protocols with larger key lengths, it is desirable to have computer resources, such as more powerful processors and storage media. Lack of access to powerful computing equipment makes it difficult for developing countries to implement software security solutions. While many healthcare facilities in developing nations rely on software to keep track of patients' information, including their diagnoses, prognoses, and treatment plans, digitising this data remains their top priority. Those directly involved with the patient also share confidential and sensitive medical information with one another. Since the internet is accessible from anyone, even in areas with limited resources, protecting the privacy and secrecy of electronic health records is crucial. Lightweight cryptosystems with reduced processing and storage requirements are thus preferred for protecting digital assets in areas with limited resources.

2. Two-Dimensional Key Vector Based Fast and Lightweight Block Cipher

The main goal is to create a block cypher that uses a 2d key matrix to encrypt the large data set with many dimensions. Information can be securely encrypted while it is in transit or at rest in storage units using the cryptosystem that is being discussed. Selecting a two-dimensional vector that contains a set of keys is the essence of a key vector [12]. To decipher the encrypted information, a multiplicative inverse is found for every key. To further increase the randomness of the dataset contents' encryption, an extra key is also added to the cryptosystem. In the phase of encryption, an input character is changed by a key taken out from the key vector and an element from additionally chosen key vector. The processes are applied iteratively in order to ensure that the accompanying ciphered information is sufficiently random. The decryption has to find the multiplicative inverse of the key elements and to apply this multiplicative inverse over the reverse actions in order to recover the original input. The purpose is to illustrate the cryptanalysis process, which will determine the cipher's strength and capacity to withstand common attacks.

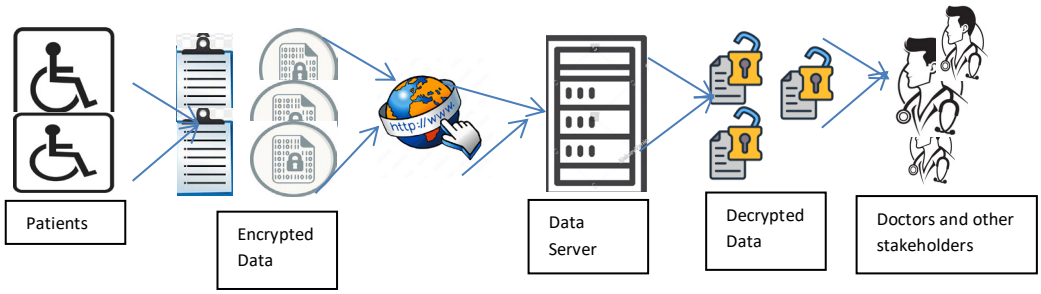

Fig. 1 Information sharing among the primary stakeholders of the healthcare system

The architectural model as depicted in the Fig. 1 show the sharing of information among hr primary stakeholders o the healthcare ecosystem and building blocks for providing data security. Using the block cipher approach, which is covered below, the patient's information is encrypted when they enrol and then sent to the storage units of the healthcare system. It is necessary to decrypt and encrypt the patient data received from various electronic media using separate sets of keys. One can search these encrypted contents with the customised encrypted query. The information obtained is then sent to the remote place where it can be queried over public channels, decrypted by the authorized entity for access.

This article introduces a new, lightweight method for protecting data saved in storage units, which is particularly useful in healthcare settings when resources are limited. A lightweight security technique is implemented [11] by utilizing the key (vector) bunch matrix, also known as a 2-dimensional key vector, to encrypt data stored in the units, including patient data. Additionally, the cypher is further strengthened by using supplementary key block. For simplicity's sake, we'll use 2048 binary bits for both the key bunch matrix and the extra two-dimensional key vector in this design. Even with little equipped resources, the encryption and decryption operations go smoothly since they are designed to utilize little computational power from the processing units. In contrast to more traditional approaches, the time needed to encrypt and decode data on devices with limited resources is significantly lower in resource-poor contexts. Additionally, when compared to other widely used block ciphers of the same type, it is discovered to be as resilient. Hereunder, the central equation for the cipher's encryption model is

$$C = ([P \bullet Key\_Enc] mod\ 256\ ) \oplus Sup\_Key$$

where 2d key vector is used for encryption Key_Enc and Sup_Key is extra supplementary key vector. And $\forall x \in [Key\_Enc], 0 \le x \le 255,\ \forall x = 2k+1 | 0 \le k \le 127$ and [P] and [C] respectively, where the first one includes the dataset characters as plain text and the corresponding encrypted characters in the blockwise fashion. The Sup_Key have components like z, such that $\forall z \in [Sup\_Key], 0 \le z \le 255.$ Each chosen batch is of size nxn. The dot product between two corresponding operands is denoted by this binary operator '•'. The model of the decryption procedure is

$$P = [(C \oplus Sup\_Key) \bullet Key\_Dec]\ \text{mo d } 256,$$

where decryption vector is Key_Dec, such that $\forall y \in [Key\_Dec], 0 \le y \le 255,$ $\forall y = 2k+1 \mid 0 \le k \le 127$. The relation: $[Key\_Enc \bullet Key\_Dec] \equiv 1 mod\ 256$, act as a

governing relation to inter-relate the two key bunch matrices used in the cryptosystem. Here in the process to enhance the security of the system, the corresponding elements of the block matrices undergo '•' operation which overcome the erstwhile constraint on the key size of the block ciphers.

3. Governing Principles and Mathematical Models

The first step in the development of the cryptosystem is that the records contents, obtained from each end of the pairs patient as well as the major part of the data that is to be stored in the health-care system, ought to be in the commonly exchangeable column based file format, which is typically used to move a data in bulk from a system to another system over a network. Here data is represented in Comma Separated Values (CSV) format, where each record value is separated with the comma. The health-care dataset HC_DS includes the patients' medical records, used for illustration and denoted as follows.

$$HC_{DS} = [rec_{uvw}], u = 1\ to\ r,\ v = 1\ \ to\ d,\ w = 1\ to\ c, \quad (1)$$

The total record count is denoted by r, while the total attributes count is represented by d. The character count is indicated by c. The notation rec_{uvw} refers to the w^{th} character in the value of the v^{th} attribute of the u^{th} record in the HC_DS.

For the purpose of computation, every character owned by record of earlier said HC_DS is transformed to its EBCDIC equivalent of two nibbles of binary digits. The rows of binary digits we obtained of the HC_DS are ordered in the matrices P of dimension $n \times n$, which can be followed in the equation 2.

$$P = \left[p_{ij}\right], i = 1\ to\ n, j = 1\ to\ n \mid p_{ij} = EBCDIC(HC\_DS(rec_{uvw})), \quad (2)$$

where [0, 255] is the domain of each p_{ij}.

Let Key_Enc be the key bunch vector utilized for encrypting the block elements and expressed as

$$Key\_Enc\ = \left[ke_{ij}\right],\ i = 1\ to\ n, j = 1\ to\ n. \quad (3)$$

To reverse the effect of the above operation, a decryption key bunch vector known as Key_Dec is utilized, which can be expressed as

$$Key\_Dec = \left[kd_{ij}\right], j = 1\ to\ n, i = 1\ to\ n, \quad (4)$$

The individual keys, namely ke_{ij} and $kd_{ij} | 1 \leq i, j \leq n$, utilized in our proposed system are multiplicative inverses of one another and are represented as

$$\left(ke_{ij} \times kd_{ij}\right) \equiv 1\ mod\ 256 |\ j = 1\ to\ n, i = 1\ to\ n\ . \quad (5)$$

The values chosen for each ke_{ij} are odd numbers within the range of [1, 255], and the corresponding values calculated for kd_{ij} also fall within the same range of [1, 255] and are unique odd numbers. All the odd numbers from the interval [1, 255] of Key_Enc can be mathematically expressed into a matrix with equation

$$Key\_Enc = [16(i-1) + 2j - 1], j = 1\ to\ 8\ and\ i = 1\ to\ 16, \quad (6)$$

Then the corresponding values can be calculated using (5) and organized in a 16×8 grid format as follows.

The prevailing principle charted in the proposed system for generating the cipher text from the textual contents of the HC_DS, organized in blocks P, to get the encrypted elements block $C$is as shown below:

$$C = [c_{ij}] = \left([p_{ij} \times ke_{ij}]\ mod\ 256\right)\ xor\ Sup\_Key, \quad (8)$$

$where\ j = 1\ to\ n, i = 1\ to\ n$ and Sup_Key is the extra key vector brought here to introduce further the diffusion and confusion in the encrypted elements. The numbers of Sup_Key are selected in the interval [1-255]. The resulting number of the $[p_{ij} \times ke_{ij}] \bmod 256$ is xored with the corresponding element of the Sup_Key grid matrix.

$$\text{Key\_Dec} = \begin{bmatrix} 1 & 171 & 205 & 183 & 57 & 163 & 197 & 239 \\ 241 & 27 & 61 & 167 & 41 & 19 & 53 & 223 \\ 225 & 139 & 173 & 151 & 25 & 131 & 165 & 207 \\ 209 & 251 & 29 & 135 & 9 & 243 & 21 & 191 \\ 193 & 107 & 141 & 119 & 249 & 99 & 133 & 175 \\ 177 & 219 & 253 & 103 & 233 & 211 & 245 & 159 \\ 161 & 75 & 109 & 87 & 217 & 67 & 101 & 143 \\ 145 & 187 & 221 & 71 & 201 & 179 & 213 & 127 \\ 129 & 43 & 77 & 55 & 185 & 35 & 69 & 111 \\ 113 & 155 & 189 & 39 & 169 & 147 & 181 & 95 \\ 97 & 11 & 45 & 23 & 153 & 3 & 37 & 79 \\ 81 & 123 & 157 & 7 & 137 & 115 & 149 & 63 \\ 65 & 235 & 13 & 247 & 121 & 227 & 5 & 47 \\ 49 & 91 & 125 & 231 & 105 & 83 & 117 & 31 \\ 33 & 203 & 237 & 215 & 89 & 195 & 229 & 15 \\ 17 & 59 & 93 & 199 & 73 & 51 & 85 & 255 \end{bmatrix} \quad (7)$$

The inverse procedure to recover the original contents of the HC_DS in the EBCDIC form, decryption system is implemented as

$$P = [p_{ij}] = [kd_{ij} \times (C\ xor\ Sup\_Key)_{ij}] mod\ 256, \quad i = 1\ to\ n, j = 1\ to\ n. \quad (9)$$

3.1 Quasi-codes for the encryption-decryption process

The illustrative quasi-codes for the encipherment procedures implemented in the proposed system are as shown below:

Each of the key concatenated with the HC_DS_CSV is sent to the mixBits() function and the function is applied on it. This method would lead to dispersal of the binary digits among the block and consequently more randomness, and hence the strength of the developed system. In the ixBits() , handling such plaintext block having dataset fillings into any reiteration round say, P =

$[p_{ij}], j = 1$ to $n, i = 1$ to n, after going through the reiteration and rounds. Seeing $n = 2m$, the chunk of P can be expressed in the form represented in (10).

Pseudo code for Encryption Algorithm

Key-in and convert all elements of dataset HC_DS as
$HC\_DS\_CSV = EBCDIC(HC\_DS\_CSV)$
Select size n *and* itr
Read Key_Enc of square size n
Read Sup_Key of square size n
Arrange given characters into grids; each of nxn, say P
// Add 0s to make last block complete, if partial
For every block P *in* $total\_blocks$ do begin
For $k = 1$ *to* itr do begin
For $i = 1$ *to* n do begin
For $j = 1$ *to* n do begin
$p_{ij} = (ke_{ij} \times p_{ij}) mod\ 256$
loop j end
loop i end
Assemble P by combining $[p_{ij}]$
$P = P \oplus$ Sup_Key
$P = mixBits(P)$
loop k end
$C = P$

Pseudo code for Decryption Algorithm

Read $C, Key\_Enc, Sup\_Key, n, itr$
$Key\_Dec = obtainMultiInverse(Key\ \_Enc)$
For each block C in $total\_blocks$ do begin
for $k = 1$ *to* itr do begin
$C = inverseMixBits(C)$
$C = C \oplus$ Sup_Key
// $Sup\_Key = [sk_{ij}] \mid i = 1$ *to* n, $j = 1$ *to* n
For $i = 1$ *to* n do begin
For $j = 1$ *to* n do begin
Calculate $c_{ij} = (d_{ij} \times c_{ij}) mod\ 256$
loop j end
loop i end
Construct C by merging $[c_{ij}]$
loop k end
$P = C$
Write (P)
end of C *in* $total\_blocks$ for loop

$$P = \begin{bmatrix} p_{11} & p_{12} & \cdot\cdot p_{1m} & p_{1(m+1)} & \cdot & \cdot & p_{1(n-1)} & p_{1n} \\ p_{21} & p_{22} & \cdot\cdot p_{2m} & p_{2(m+1)} & \cdot & \cdot & p_{2(n-1)} & p_{2n} \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ p_{n1} & p_{n2} & \cdot\cdot p_{nm} & p_{n(m+1)} & \cdot & \cdot & p_{n(n-1)} & p_{nn} \end{bmatrix} \quad (10)$$

After converting block P elements to its EBCDIC equivalence and then to its binary, a vector is created with the dimensions of $n \times 8n$, as illustrated in (11). The decimal integers are generated by uniquely reading the binary bits, as summarized below. From the 1st column, the 1st eight bits are read to convert into the corresponding decimal number.

$$\begin{bmatrix} b_{111}b_{112}..b_{118} & \ldots & b_{1m1}b_{1m2}..b_{1m8} & b_{1(m+1)1}b_{1(m+1)2}..b_{1(m+1)8} & \ldots & b_{1n1}b_{1n2}..b_{1n8} \\ b_{211}b_{212}..b_{218} & \ldots & b_{2m1}b_{2m2}..p_{2m8} & b_{2(m+1)1}b_{2(m+1)2}..b_{2(m+1)8} & \ldots & b_{2n1}b_{2n2}..b_{2n8} \\ \cdot & \ldots & \cdot & \cdot & \ldots & \cdot \\ \cdot & \ldots & \cdot & \cdot & \ldots & \cdot \\ b_{n11}b_{n12}..b_{n18} & \ldots & b_{nm1}b_{nm2}..b_{nm8} & b_{n(m+1)1}b_{n(m+1)2}..b_{n(m+1)8} & \ldots & b_{nn1}b_{nn2}..b_{nn8} \end{bmatrix} \quad (11)$$

Further eight bits of the same column are read in a similar manner to generate the 2nd integer, and this continues until the 1st column of the vector is fully processed. To create the subsequent number, the 1st eight binary digits of the $(m+1)^{th}$ column of the vector are used. This procedure is continued till the column ends. Additional integers are generated by applying the same method to the 2nd column, $(m+2)^{th}$ column, 3rd, $(m+3)^{rd}$ column, and so forth. The resulting decimal numbers are arranged in a grid format with a square size of n. If the matrix has fewer than eight rows, the binary digits from the first column in both the first half and the second half are utilized to generate the number. This process ensures a meticulous mixing of the binary numbers, thereby enhancing the dispersal of the ciphered elements in each iteration. The inverse process is carried out in the $inverseMixBits()$ function, which is part of the decryption process's pseudo code.

4. Case Study with Outcomes

For the demonstrative hands-on implementation, the healthcare dataset [13], labelled as HC_DS_CSV, is comprised of approximately 57000 characters and have twenty-nine dimensions, mostly with categorical and expressive data. The first two records of the dataset are shown as follows for the reference:

Name,Age, Gender,Blood Type,Medical Condition,Date of Admission,Doctor,Hospital,Insurance Provider,Billing Amount,Room Number,Admission Type,Discharge Date,Medication,Test Results

Bobby JacksOn,30,Male,B-,Cancer,1/31/2024,Matthew Smith,Sons and Miller,Blue Cross,18856.28131328,Urgent,2/2/2024,Paracetamol,Normal

LesLie TErRy,62,Male,A+,Obesity,8/20/2019,Samantha Davies,Kim Inc,Medicare,33643.32729265,Emergency,8/26/2019,Ibuprofen,Inconclusive (12)

The first 256 characters, shown in (13), are selected from (12) to create a plaintext block consisting of dimensions 16×16. These 256 characters are converted to EBCDIC values, arranges in a row-wise plaintext matrix as in (14) and used as raw input.

Name,Age,Gender,Blood Type,Medical Condition,Date of Admission,Doctor,Hospital,Insurance Provider,Billing Amount,Room Number,Admission Type,Discharge Date,Medication,Test ResultsBobby JacksOn,30,Male,B-,Cancer,1/31/2024,Matthew Smith,Sons and Miller,Blue (13)

$$P = \begin{bmatrix} 215 & 153 & 150 & 165 & 137 & 132 & 133 & 153 & 64 & 201 & 196 & 107 & 200 & 150 & 162 & 151 \\ 137 & 163 & 129 & 147 & 64 & 213 & 129 & 148 & 133 & 107 & 193 & 132 & 132 & 153 & 133 & 162 \\ 162 & 107 & 195 & 137 & 163 & 168 & 107 & 226 & 163 & 129 & 163 & 133 & 107 & 233 & 201 & 215 \\ 64 & 195 & 150 & 132 & 133 & 107 & 195 & 150 & 164 & 149 & 163 & 168 & 64 & 213 & 129 & 148 \\ 133 & 107 & 215 & 136 & 150 & 149 & 133 & 64 & 213 & 164 & 148 & 130 & 133 & 153 & 107 & 200 \\ 150 & 162 & 151 & 137 & 163 & 129 & 147 & 64 & 227 & 168 & 151 & 133 & 107 & 200 & 150 & 162 \\ 151 & 137 & 163 & 129 & 147 & 64 & 214 & 166 & 149 & 133 & 153 & 162 & 136 & 137 & 151 & 107 \\ 197 & 148 & 133 & 153 & 135 & 133 & 149 & 131 & 168 & 64 & 226 & 133 & 153 & 165 & 137 & 131 \\ 133 & 162 & 107 & 212 & 133 & 133 & 163 & 162 & 64 & 131 & 153 & 137 & 163 & 133 & 153 & 137 \\ 129 & 64 & 134 & 150 & 153 & 64 & 148 & 133 & 129 & 149 & 137 & 149 & 135 & 134 & 164 & 147 \\ 64 & 164 & 162 & 133 & 64 & 150 & 134 & 64 & 197 & 200 & 217 & 162 & 107 & 200 & 150 & 162 \\ 151 & 137 & 163 & 129 & 147 & 64 & 150 & 165 & 133 & 153 & 129 & 147 & 147 & 64 & 153 & 129 \\ 163 & 137 & 149 & 135 & 107 & 200 & 150 & 162 & 151 & 137 & 163 & 129 & 147 & 64 & 150 & 165 \\ 133 & 153 & 129 & 147 & 147 & 64 & 153 & 129 & 163 & 137 & 149 & 135 & 64 & 134 & 150 & 150 \\ 163 & 149 & 150 & 163 & 133 & 107 & 212 & 150 & 153 & 163 & 129 & 147 & 137 & 163 & 168 & 64 \\ 149 & 129 & 163 & 137 & 150 & 149 & 129 & 147 & 64 & 131 & 150 & 148 & 151 & 129 & 153 & 137 \end{bmatrix} \quad (14)$$

As shown in (15), the randomly selected encryption key bunch square matrix has a size of 16 and contains the odd numbers from 0 to 255.

$$Key\_Enc = \begin{bmatrix} 217 & 169 & 61 & 255 & 65 & 65 & 111 & 7 & 33 & 73 & 133 & 71 & 223 & 53 & 173 & 43 \\ 75 & 163 & 103 & 207 & 205 & 107 & 191 & 113 & 245 & 123 & 143 & 129 & 205 & 227 & 43 & 51 \\ 125 & 103 & 117 & 209 & 57 & 221 & 159 & 213 & 181 & 199 & 13 & 113 & 95 & 121 & 21 & 143 \\ 9 & 241 & 137 & 41 & 73 & 93 & 167 & 97 & 207 & 79 & 185 & 47 & 35 & 25 & 147 & 23 \\ 171 & 65 & 39 & 107 & 123 & 73 & 51 & 7 & 15 & 125 & 77 & 71 & 75 & 235 & 47 & 117 \\ 137 & 37 & 149 & 93 & 11 & 197 & 187 & 107 & 37 & 87 & 233 & 195 & 217 & 17 & 9 & 111 \\ 61 & 129 & 5 & 189 & 63 & 17 & 35 & 199 & 223 & 45 & 49 & 191 & 209 & 241 & 9 & 97 \\ 179 & 123 & 229 & 193 & 75 & 117 & 79 & 197 & 107 & 99 & 219 & 113 & 15 & 123 & 35 & 79 \\ 33 & 179 & 219 & 53 & 253 & 131 & 169 & 213 & 45 & 221 & 59 & 67 & 197 & 29 & 171 & 113 \\ 89 & 105 & 225 & 159 & 173 & 135 & 213 & 43 & 149 & 75 & 83 & 149 & 7 & 69 & 163 & 71 \\ 29 & 91 & 203 & 155 & 7 & 97 & 187 & 85 & 199 & 163 & 89 & 219 & 19 & 223 & 93 & 139 \\ 93 & 81 & 83 & 73 & 175 & 57 & 249 & 243 & 105 & 31 & 39 & 9 & 205 & 215 & 73 & 103 \\ 131 & 129 & 65 & 163 & 163 & 11 & 195 & 243 & 135 & 171 & 25 & 15 & 113 & 197 & 109 & 215 \\ 105 & 179 & 215 & 89 & 209 & 245 & 223 & 193 & 141 & 75 & 121 & 157 & 7 & 197 & 41 & 11 \\ 189 & 51 & 245 & 99 & 227 & 193 & 235 & 155 & 73 & 107 & 35 & 223 & 213 & 115 & 29 & 43 \\ 191 & 33 & 63 & 13 & 59 & 23 & 161 & 169 & 31 & 133 & 73 & 251 & 39 & 13 & 41 & 131 \end{bmatrix} \quad (15)$$

To increase the level of diffusion while enciphering the plaintext data, an additional key is built containing random positive integers of range [0-255] taken in a square matrix of size 16. This key is named as shown in (16).

$$Sup\_Key = \begin{bmatrix} 223 & 13 & 0 & 233 & 56 & 23 & 140 & 234 & 161 & 55 & 98 & 31 & 119 & 196 & 44 & 56 \\ 180 & 230 & 62 & 107 & 111 & 136 & 8 & 247 & 25 & 36 & 224 & 170 & 138 & 224 & 84 & 0 \\ 31 & 243 & 145 & 205 & 29 & 252 & 62 & 109 & 127 & 125 & 62 & 10 & 179 & 103 & 18 & 125 \\ 65 & 89 & 47 & 241 & 119 & 217 & 145 & 149 & 10 & 171 & 246 & 68 & 100 & 32 & 151 & 57 \\ 182 & 19 & 4 & 165 & 67 & 40 & 193 & 53 & 65 & 178 & 136 & 87 & 157 & 163 & 194 & 105 \\ 105 & 208 & 126 & 193 & 53 & 212 & 125 & 237 & 165 & 209 & 39 & 105 & 166 & 114 & 40 & 110 \\ 50 & 134 & 56 & 136 & 54 & 189 & 1 & 7 & 111 & 34 & 136 & 129 & 7 & 39 & 97 & 2 \\ 200 & 60 & 10 & 210 & 35 & 43 & 48 & 36 & 214 & 184 & 79 & 71 & 38 & 68 & 13 & 145 \\ 137 & 143 & 244 & 135 & 114 & 16 & 170 & 89 & 114 & 172 & 158 & 77 & 36 & 19 & 37 & 58 \\ 141 & 115 & 117 & 215 & 87 & 171 & 112 & 53 & 33 & 166 & 71 & 99 & 89 & 36 & 5 & 245 \\ 138 & 168 & 181 & 198 & 207 & 1 & 122 & 45 & 13 & 104 & 27 & 115 & 16 & 229 & 184 & 104 \\ 12 & 107 & 26 & 123 & 27 & 178 & 219 & 243 & 54 & 193 & 226 & 100 & 239 & 205 & 12 & 222 \\ 129 & 221 & 188 & 215 & 55 & 157 & 220 & 53 & 23 & 141 & 101 & 136 & 141 & 168 & 221 & 240 \\ 136 & 168 & 54 & 218 & 184 & 239 & 157 & 70 & 210 & 26 & 45 & 22 & 188 & 115 & 39 & 80 \\ 37 & 90 & 94 & 76 & 155 & 246 & 185 & 49 & 118 & 149 & 81 & 36 & 7 & 92 & 89 & 59 \\ 55 & 247 & 88 & 53 & 141 & 94 & 102 & 176 & 211 & 235 & 238 & 182 & 216 & 80 & 101 & 32 \end{bmatrix} \quad (16)$$

The following is the representation of the corresponding decryption key bunch vector, say Key_Dec, for the given (15), as determined by applying the principle of the multiplicative inverse

$$Key\_Dec = \begin{bmatrix} 105 & 153 & 21 & 255 & 193 & 193 & 143 & 183 & 225 & 249 & 77 & 119 & 31 & 29 & 37 & 131 \\ 99 & 11 & 87 & 47 & 5 & 67 & 63 & 145 & 93 & 179 & 111 & 129 & 5 & 203 & 131 & 251 \\ 213 & 87 & 221 & 49 & 9 & 117 & 95 & 125 & 157 & 247 & 197 & 145 & 159 & 201 & 61 & 111 \\ 57 & 17 & 185 & 25 & 249 & 245 & 23 & 161 & 47 & 175 & 137 & 207 & 139 & 41 & 155 & 167 \\ 3 & 193 & 151 & 67 & 179 & 249 & 251 & 183 & 239 & 213 & 133 & 119 & 99 & 195 & 207 & 221 \\ 185 & 173 & 189 & 245 & 163 & 13 & 115 & 67 & 173 & 103 & 89 & 235 & 105 & 241 & 57 & 143 \\ 21 & 129 & 205 & 149 & 191 & 241 & 139 & 247 & 31 & 165 & 209 & 63 & 49 & 17 & 57 & 161 \\ 123 & 179 & 237 & 65 & 99 & 221 & 175 & 13 & 67 & 75 & 83 & 145 & 239 & 179 & 139 & 175 \\ 225 & 123 & 83 & 29 & 85 & 43 & 153 & 125 & 165 & 117 & 243 & 107 & 13 & 53 & 3 & 145 \\ 233 & 217 & 33 & 95 & 37 & 55 & 125 & 131 & 189 & 99 & 219 & 189 & 183 & 141 & 11 & 119 \\ 53 & 211 & 227 & 147 & 183 & 161 & 115 & 253 & 247 & 11 & 233 & 83 & 27 & 31 & 245 & 35 \\ 245 & 177 & 219 & 249 & 79 & 9 & 73 & 59 & 217 & 223 & 151 & 57 & 5 & 231 & 249 & 87 \\ 43 & 129 & 193 & 11 & 11 & 163 & 235 & 59 & 55 & 3 & 41 & 239 & 145 & 13 & 101 & 231 \\ 217 & 123 & 231 & 233 & 49 & 93 & 31 & 65 & 69 & 99 & 201 & 181 & 183 & 13 & 25 & 163 \\ 149 & 251 & 93 & 75 & 203 & 65 & 195 & 147 & 249 & 67 & 139 & 31 & 125 & 187 & 53 & 131 \\ 63 & 225 & 191 & 197 & 243 & 167 & 97 & 153 & 223 & 77 & 249 & 51 & 151 & 197 & 25 & 43 \end{bmatrix} \quad (17)$$

By utilizing inputs (15) and (16) as key elements and applying the enciphering procedure outlined in the previous section, the resulting ciphertext block with dimensions as 16×16 is derived from (14) and presented in (18) as follows:

$$C = \begin{bmatrix} 91 & 250 & 17 & 39 & 180 & 73 & 80 & 215 & 70 & 58 & 144 & 249 & 36 & 18 & 37 & 114 \\ 135 & 64 & 145 & 81 & 22 & 161 & 130 & 252 & 144 & 56 & 226 & 190 & 164 & 100 & 25 & 111 \\ 90 & 13 & 118 & 221 & 195 & 202 & 225 & 215 & 9 & 148 & 97 & 200 & 110 & 196 & 120 & 225 \\ 225 & 85 & 159 & 69 & 209 & 102 & 142 & 34 & 154 & 114 & 155 & 163 & 148 & 241 & 53 & 76 \\ 5 & 104 & 252 & 73 & 27 & 137 & 51 & 246 & 150 & 12 & 204 & 238 & 83 & 232 & 147 & 45 \\ 188 & 125 & 255 & 178 & 127 & 167 & 89 & 104 & 186 & 127 & 84 & 106 & 73 & 191 & 18 & 175 \\ 121 & 199 & 224 & 52 & 252 & 35 & 121 & 233 & 29 & 132 & 23 & 80 & 202 & 225 & 4 & 51 \\ 46 & 162 & 141 & 133 & 47 & 24 & 15 & 83 & 111 & 185 & 224 & 208 & 119 & 43 & 196 & 117 \\ 89 & 50 & 0 & 181 & 26 & 184 & 68 & 65 & 121 & 214 & 36 & 112 & 15 & 239 & 201 & 255 \\ 36 & 154 & 150 & 216 & 183 & 63 & 220 & 144 & 221 & 191 & 8 & 78 & 178 & 230 & 130 & 117 \\ 213 & 143 & 49 & 166 & 15 & 45 & 122 & 25 & 114 & 66 & 178 & 31 & 200 & 148 & 41 & 18 \\ 246 & 39 & 76 & 40 & 111 & 230 & 46 & 194 & 183 & 144 & 96 & 141 & 3 & 91 & 223 & 70 \\ 201 & 97 & 63 & 151 & 11 & 81 & 122 & 249 & 17 & 36 & 80 & 170 & 55 & 127 & 137 & 157 \\ 174 & 215 & 153 & 178 & 176 & 69 & 44 & 156 & 36 & 239 & 7 & 138 & 3 & 45 & 122 & 211 \\ 223 & 254 & 94 & 102 & 86 & 25 & 213 & 138 & 39 & 82 & 44 & 15 & 112 & 42 & 159 & 190 \\ 190 & 24 & 254 & 72 & 53 & 52 & 237 & 34 & 166 & 168 & 1 & 155 & 20 & 199 & 67 & 77 \end{bmatrix} \quad (18)$$

The decryption process and the transmitted Sup_Key Sup _ Key are followed at the recipient end, as shown in (16), using the computed (17) on the block of ciphertext to recover the initial P, as seen in (14).

5. Performance and Simulation Scrutiny

Simulated on an intel processor with 8 GB RAM and 1.80 GHz, 64-bit Windows 11 OS using Java programming language of version 22 using ECB mode, each block allocated of the keys-encryption key, supplementary key and the plaintext is of 256 integers individually formatted as 16×16 decimals vector and seen as 2048 binary digits. The encryption and decryption processes are configured to run for 16 rounds to effectively blend the bits, using $mix()$ and create substantial confusion as well as dispersion. The key generation module takes seconds to produce 256 random odd decimals for a square key with 16 across rows and columns. The programs are tested on various .csv formatted datasets from different public domains to compare the elapsed

times for the enciphering and deciphering the 2-dimensional vector size of 2048 bits. Table 1 and Figure 2 display the time expended for the development of fast and light-weighted block cipher model.

Table 1 Duration incurred in the encipherment of rapid block cipher

Dataset size in MB	encryption	decryption
2.04	1.77	1.85
4.2	3.24	4.52
10.1	6.526	7.63
21.6	9.30	9.47
41	16.97	17.29

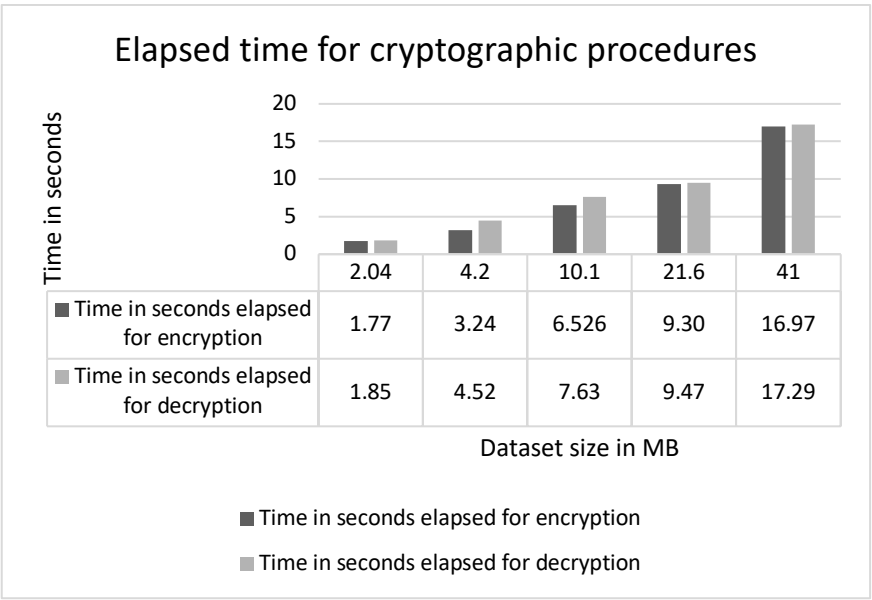

Fig. 2 Elapsed time in seconds for the cryptographic procedures: encryption and decryption

6. Conclusion

In this article, we build a cipher system that can protect the integrity, confidentiality, and availability of subtle patient data in healthcare organisations' enormous, multi-dimensional databanks, even in environs with limited resources. Using an extra key vector, multiplicative inverses, and an encryption key block with a number of keys, the created cryptic processes carry

out the jobs. The legacy systems' low-configured machinery is suitable for the established operations. The computing cost can be easily increased by increasing the magnitude of the key bunch block used for encryption, supplementary key vector, and plaintext block, all of which contribute to better security. In order to create enigmatic text that is impervious to cryptanalytic attacks, the iterative procedure transforms the binary bits of each character multiple times using modular products and xored actions wintin the matrices and input blocks. With little adjustment to the computational cost and the appropriate pre-processing of the raw input data, the created cryptosystem can be easily modified to securely disburse healthcare images and three-dimensional datasets and other formatted information, thanks to the procedures' high effectiveness even in low-resource settings.

References

1. Adler-Milstein J, DesRoches CM, Kralovec P, Foster G,Worzala C, et al. Electronic health record adoption in US hospitals: progress continues, but challenges persist. Health Aff. 34:2174–80, 2015
2. Adler-Milstein J, Lin SC, Jha AK. The number of health information exchange efforts is declining, leaving the viability of broad clinical data exchange uncertain. Health Aff. 35:1278–85, 2016
3. Hilal Atasoy, Brad N. Greenwood, Jeffrey Scott McCullough, “The Digitization of Patient Care: A Review of the Effects of Electronic Health Records on Health Care Quality and Utilization”, Annual Review of Public Health 40:1, 487-500, 2019
4. Braa, J., Hanseth, O., Heywood, A., Mohammed, W., & Shaw, V. (2007). Developing health information systems in developing countries: The flexible standards strategy. MIS Quarterly, 31, 381–402.
5. Braa, J., & Sahay, S. Integrated health information architecture: Power to the users: Design, development and use. New Delhi: Matrix Publishers, 2012
6. Gebre-Mariam, M., & Fruijtier, E. Countering the ‘dam effect’: The case for architecture and governance in large scale developing country health information systems. Information Technology for Development, 1–26, 2017
7. Mihailescu, M., Mihailescu, D., & Schultze, U. The generative mechanisms of healthcare digitalization. Thirty Sixth International Conference on Information Systems. Fort Worth, 2015
8. Manda, T. D. Developing capacity for maintenance of HIS in the context of loosely coordinated project support arrangements. IST-Africa Conference (pp. 1– 10) 2015
9. Mingers, J., & Standing, C. Why things happen–developing the critical realist view of causal mechanisms. Information and Organization, 27, 171–189, 2017
10. Mosse, D. Cultivating development. London: Pluto Press, 2005
11. Kimaro, H. C., & Nhampossa, J. L. Analyzing the problem of unsustainable health information systems in less-developed economies: Case studies from Tanzania and Mozambique. Information Technology for Development, 11, 273–298, 2005

12. Dr. Shirisha Kakarla, “Securing large datasets involving fast-performing key bunch matrix block cipher”, Healthcare Data Analytics and Management, Advances in Ubiquitous Sensing Applications for Healthcare, Elsevier Publications, Paperback ISBN: 9780128153680, https://doi.org/10.1016/C2017-0-03245-7, Vol 2, pp. 111-132 ,2019
13. K. Shirisha, V.U.K. Sastry, “A Block Cipher Involving the Elements of a Key bunch Matrix as Powers of the Plaintext Elements”, International Journal of Computer Networks and Security, Recent Science Publications, ISSN: 2051-6878, Vol. 23, Issue No. 2, pp. 1192 -1197, December 2013
14. Demographic Statistics By Zip Code, Retrieved from https://catalog.data.gov/dataset/demographic-statistics-by-zip-code-cfc9/resource/e43f1938-3c4a-4501-9aaf-46891bb21553 , 2017