Indonesian Journal of Electrical Engineering and Computer Science
Vol. 24, No. 1, October 2021, pp. 428~443
ISSN: 2502-4752, DOI: 10.11591/ijeecs.v24.i1.pp428-443

A novel secure biomedical data aggregation using fully homomorphic encryption in WSN

Chethana G., Padmaja K. V.
Department of Electronics and Communication Engineering, RV College of Engineering, VTU, Karnataka, India

Article Info

Article history:

Received Oct 6, 2020
Revised Jul 29, 2021
Accepted Aug 4, 2021

Keywords:

Generalized inverse
Homomorphic encryption
Secure data aggregation
Sign mod
Signed finite field

ABSTRACT

A new method of secure data aggregation for decimal data having integer as well as fractional part using homomorphic encryption is described. The proposed homomorphic encryption provides addition, subtraction, multiplication, division and averaging operations in the cipher domain for both positive and negative numbers. The scheme uses integer matrices in finite field Zp as encryption and decryption keys. An embedded digital signature along with data provides data integrity and authentication by signature verification at the receiving end. The proposed scheme is immune to chosen plaintext and chosen ciphertext attacks. In the case of homomorphic multiplication, the ciphertext expansion ratio grows linearly with the data size. The computational complexity of the proposed method for multiplication and division is relatively less by 22.87% compared to Brakerski and Vaikantanathan method when the size of the plaintext data is ten decimal digits.

Corresponding Author:

Chethana G.
Department of Electronics and Communication Engineering
RV College of Engineering, 8th Mile, Mysore Road, Bengaluru-560059
Affiliated under Visveswaraya Technological University, Belagavi, India
Email: chethanag@rvce.edu.in

1. INTRODUCTION

The Basic purpose of data aggregation (DA) in WSN is to combine the data collected from the sensor nodes into a suitable aggregate. The aggregate may be sum, average, min, max, median or any other metric of the collected dataset. The aggregate type depends on nature of the problem and the requirements of end users (EU). In general, DA eliminates unnecessary, inconsequential, redundant and outdated data values. In most of the cases, DA compresses the data size without affecting the core information. This in turn reduces the traffic load from aggregator to the next intended destination that results in lower energy consumption and consequent increase in the life of the WSN.

- Secure data aggregation

In this paper, the biomedical data like body temperature, pulse rate, and breath rhythm are collected from wearable sensors along with additional pathological data like blood pressure (systolic and diastolic), and blood sugar, for aggregation. The cluster head (CH) collects the data from the sensors and stores them in a reputed cloud server (CS). Here, the CH is the data owner and the aggregation operation is delegated to the CS. Even though cloud servers have built in security against external threats, an honest but curious insider may access the stored health records without authorization. To prevent this, all the sensitive data are sent to CS, stored at CS and sent out by the CS in the encrypted form. The aggregate operations such as *sum, product, average* is etc. are carried out in the CS. Secure data aggregation (SDA) can be implemented using non-homomorphic or homomorphic methods. In this work we use fully homomorphic encryption, using

matrix keys, to implement SDA. The SDA operations based on homomorphic operations are carried out at the CS.

- Related work

The impact of data aggregation from the sensor nodes into a suitable aggregate is proposed in 2002 [1]. Randhawa and Jain [2] expalined current status and future directions in data aggregation in wireless sensor networks. In [3]-[5], the authors have comprehensively reviewed several secure data aggregation (SDA) schemes which include non-homomorphic as well as homomorphic methods. SDA using homomorphic methods based on matrix keys are described in [6]-[8]. In [6], authors have described homomorphic addition having very good security measures. In [7], the authors have presented homomorphic addition of matrix data which is suitable for digital images. In [8], an SDA scheme suitable for large-scale wireless sensor networks is comprehensively described. Fully homomorphic encryption (FHE) schemes based on different mathematical approaches are extensively described in the survey papers [9]-[11]. In these papers, the authors have reviewed most of the available classical and lattice also known as matrix based FHE methods with appropriate comparison among those methods. In [11], Martins *et al.* have discussed various FHE methods from the engineering point of view. In [12], Dijk, *et al.* have presented one of the earliest FHE method known as DGHV scheme which is based on basic modular arithmetic. Craig [13], has proposed FHE using ideal lattices with squashing technique that permits bootstrapping. In [14], the authors have presented FHE which uses shorter public keys. In [15], Brakerski and Vaikantanathan (BV) have realized FHE with re-linearization and dimension reduction techniques to improve the performance of FHE. Because of dimension reduction technique, the decryption process is relatively fast. However, in BV method, the plaintext is a bit (0 or 1). Hedglin phillips and reilley (HPR) [16], have developed FHE directly for integers whereby conversion from integers to binary and vice-versa are avoided. HPR method is computationally expensive. In [17], [18], authors have discussed on FPGA based fully homomorphic encryptions. They have provided solutions for achieving low-complexity homomorphic operations for FHE, converging on the hardware implementation. In [19]-[23], authors have discussed on privacy preserving aggregation techniques for non-homomorphic methods. Whereas in our proposed research work, new privacy preserving method for fully homomorphic aggregation functions are discussed and is realized using software which can be scaled up effortlessly at low computational cost.

2. PROPOSED METHOD

In our proposed work, a novel secure data aggregation scheme based on homomorphic operations is described. The scheme is designated as homomorphic-secure data aggregation (HSDA). The basic layout of HSDA is as shown in Figure 1.

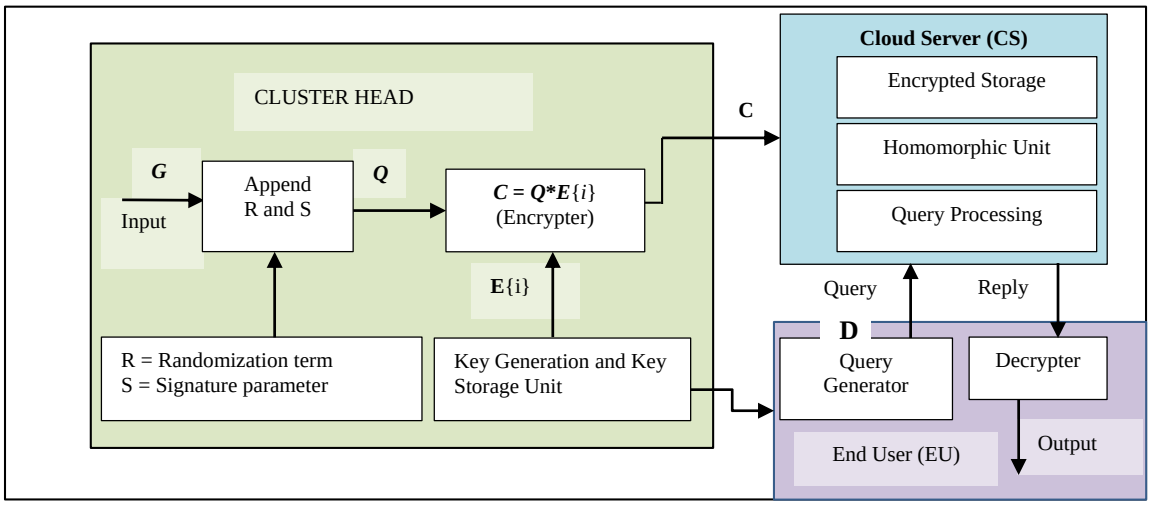

Figure 1. Layout of homomorphic secure data aggregation (HSDA)

2.1. Basic layout of homomorphic secure data aggregation

In Figure 1, the cluster head (CH) collects the vital data from sensors and it is the data owner. It also houses the Encrypter, Key Generation/Storage Unit and other necessary hardware and software. The encrypted data is sent and stored in the CS. The CS block holds the encrypted data in appropriate tables with

suitable labels. The homomorphic unit is implemented in CS using the python software. CS has the query processing unit that accepts queries from EUs and generates the corresponding response after homomorphic operations and then, sends back the correct replies to the EUs. The query and reply are also in the encrypted form. The EUs can be Doctors, Specialists or any authorized entities. The EU unit houses the decrypter which decrypts the replies from homomorphic unit to get the final result in the plaintext format. Additional details about the working of the various units of Figure 1 will be unveiled subsequently.

2.2. Symbols, definitions and notations

Our proposed method HSDA uses modular arithmetic operations involving vectors and matrices with positive and negative decimal numbers. Vectors and matrices are represented by symbols in bold capital fonts. Scalar variables are represented in normal font.

2.2.1. Modular arithmetic for signed integers

A homomorphic encryption system that uses subtraction should be able to handle both positive as well as negative numbers as, the result of subtraction of two numbers can be positive or negative. In conventional modular arithmetic Z_p, all the elements are in the range 0 to (p–1) and hence they are positive. In cryptography, p is a large prime number. In this work, signed finite field (SFF) modular arithmetic is introduced to take care of positive as well as negative integers. Conventional modular arithmetic is used in conventional finite field (CFF) Z_p. In Z_p, the range of integers is from 0 to (p–1). When negative integers are involved, the Signed Finite Field, abbreviated as SFF_p is used. The range of integers in SFF_p is from $-\text{floor}\left(\frac{p-1}{2}\right)$ to $+\text{floor}\left(\frac{p-1}{2}\right)$. Table 1 shows the comparison between CFF and SFF.

Table 1. Comparison of CFF and SFF

	Conventional Finite Field (CFF)	Signed Finite Field (SFF)
Symbol	Z_p	SFF_p
Range	Integers from 0 to (p-1)	Integers from $-\text{floor}\left(\frac{p-1}{2}\right)$ to $+\text{floor}\left(\frac{p-1}{2}\right)$
No. of elements in range	p	p
Representation of an arbitrary integer 'x'	$y = (x \bmod p) = x - \text{floor}\left(\frac{x}{p}\right) * p$	$y = x - \text{round}\left(\frac{x}{p}\right) * p$
Matlab function	$y =$ **mod**(x, p) [built in function]	$y =$ **sign Mod** (x, p) [User defined]

when x is a scalar integer, signMod (x, p) is defined as,

$$\text{signMod}\ (x, p) = x - \text{round}\left(\frac{x}{p}\right) * p \quad (1)$$

The definition is extended for an integer vector or matrix $\boldsymbol{X}$ as, signMod ($\boldsymbol{X}$, p) = $\boldsymbol{X}$ – round ($\boldsymbol{X}$/p) *p;

The signMod operation is applied to all the individual elements of matrix $\boldsymbol{X}$ to get the matrix signMod($\boldsymbol{X}$, p). The sizes of $\boldsymbol{X}$ and signMod($\boldsymbol{X}$, p) are same. Example 1 demonstrates the difference between CFF and SFF values. ***Example* 1**: Here, p = 11. Integer variable x varies from 0 to 22 and the corresponding equivalent values in CFF given by $y =$ mod (x, p) and in SFF given by z = signMod(x, p) are as shown in Table 2.

Table 2. y = mod(x, p) and z = signMod(x, p) values for p = 11 and for x = 0 to 22

x	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
y	0	1	2	3	4	5	6	7	8	9	10	0	1	2	3	4	5	6	7	8	9	10	0
z	0	1	2	3	4	5	–5	–4	–3	–2	–1	0	1	2	3	4	5	–5	–4	–3	–2	–1	0

From (1) and from Table 2, it can be seen that,

$$\text{signMod}(x, p) = \begin{cases} \text{mod}(x, p), & \text{when } 0 \le \text{mod}(x, p) \le \text{floor}(p/2) \\ \text{mod}(x, p) - p. & \text{when } \text{mod}(\text{x}, \text{p}) > \text{floor}(p/2) \end{cases}$$

It can be verified that the following distributive property which holds good for mod(…) also holds good for signMod(…) as,

signMod($a \pm b$, p) = signMod(signMod(a, p) ± signMod(b, p), p)

signMod(a * b, p) = signMod(signMod(a, p)*signMod(b, p), p)

The above identities hold good for both positive and negative integer values of a and b as well as when a and b are integer matrices. An obvious, but interesting property of sigMod(x, p) and mod(x, p), when x = 1 is,

signMod(1, p) = mod(1, p) =1, assuming $p > 1$.

Similarly, when the argument is an identity matrix $\boldsymbol{I}_{n\times n}$,

$$\text{signMod}(\boldsymbol{I}_{n\times n}, p) = \text{mod}(\boldsymbol{I}_{n\times n}, p) = \boldsymbol{I}_{n\times \text{n}} \quad \text{, assuming } p > 1.$$

2.2.2. Decryption matrices

The decryption matrix, designated by $\boldsymbol{D}$ is an integer matrix of size $m\times n$ where $\boldsymbol{D} \in \text{Z}_\text{p}^{m\times n}$. Here m is chosen to be greater than n and the elements of $\boldsymbol{D}$ are chosen such that rank($\boldsymbol{D}$) = n. The value of n depends on the size of the plaintext data element to be encrypted. Here, $\boldsymbol{D}$ is a tall matrix and it has its left modular inverse [24] designated by $\boldsymbol{A}$ such that,

$$\text{mod}(\boldsymbol{A}*\boldsymbol{D}, p) = \boldsymbol{I}_{n\times n} \tag{2}$$

Here $\boldsymbol{A} \in Z_p^{n\times m}$ and is given by,

$$\boldsymbol{A} = \boldsymbol{D}_{\text{left}}^{-\boldsymbol{1}} = (\boldsymbol{D}^{\boldsymbol{T}} * \boldsymbol{D})^{-\boldsymbol{1}} * \boldsymbol{D}^{\text{T}} \qquad \text{(in mod p)} \tag{3}$$

Here, matrix $\boldsymbol{A}$ is the Moore-Penrose inverse [25] of $\boldsymbol{D}$ in Z_p and $\boldsymbol{D}^{\text{T}}$ is the transpose of $\boldsymbol{D}$. For a given full rank $\boldsymbol{D}$, its Moore-Penrose inverse is unique and is given by (3) where $(\boldsymbol{D}^{\boldsymbol{T}} * \boldsymbol{D})^{-\boldsymbol{1}}$ is the modular matrix inverse of $(\boldsymbol{D}^{\boldsymbol{T}} * \boldsymbol{D})$. When multiple inverses are needed, they are generated using the null space of $\boldsymbol{D}$. Since $\boldsymbol{D}$ is a tall matrix, it has left null space [25]. Let matrix $\boldsymbol{F}$ represents the modular left null space of $\boldsymbol{D}$.

Then, mod($\boldsymbol{F}$*$\boldsymbol{D}$, p) = $\boldsymbol{0}_{(m-n)xn}$

Here, the size of $\boldsymbol{F}$ is (m–n) *m. When there is no ambiguity, the above equation can be written as,

$$\boldsymbol{F}*\boldsymbol{D} = \boldsymbol{0}_{(m-n)xn} \tag{4}$$

$\boldsymbol{F}$ is obtained using the modular linear algebra.

2.2.3. Encryption matrices

The encryption matrix, designated by $\boldsymbol{E}$ is an integer matrix of size $n\times m$, where matrix $\boldsymbol{E} \in \text{Z}_\text{p}^{\text{n}\times\text{m}}$. Matrix $\boldsymbol{E}$ is constructed such that $\boldsymbol{E} * \boldsymbol{D} = \boldsymbol{I}_{n\times n}$. Matrix $\boldsymbol{E}$ is derived from $\boldsymbol{A}$ and $\boldsymbol{F}$ as follows. Consider (4) and pre-multiply both sides of (4) by an arbitrary random integer matrix $\boldsymbol{W}_{nx(m-n)}$ that belongs to $Z_p^{n\times(m-n)}$. Then,

$$\boldsymbol{W}_{nx(m-n)} * (\boldsymbol{F} * \boldsymbol{D})_{(m-n)xn} = \boldsymbol{W}_{nx(m-n)} * \boldsymbol{0}_{(m-n)xn} = \boldsymbol{0}_{nxn} \tag{5}$$

In (5) can be rewritten as,

$$(\boldsymbol{W} * \boldsymbol{F})_{nxm} * \boldsymbol{D}_{mxn} = \boldsymbol{0}_{nxn} \tag{6}$$

Now, consider (2) which can be expressed as,

$$\boldsymbol{A}_{nxm} * \boldsymbol{D}_{mxn} = \boldsymbol{I}_{n\times n} \tag{7}$$

The size of the LHS of (6) as well as that of (7) is nxn. Therefore, on adding (7) and (6), we get,

$$\boldsymbol{A}_{nxm} * \boldsymbol{D}_{mxn} + (\boldsymbol{W} * \boldsymbol{F})_{nxm} * \boldsymbol{D}_{mxn} = \boldsymbol{I}_{n\times n}$$

This is rewritten as,

$$(\boldsymbol{A} + \boldsymbol{W} * \boldsymbol{F}) * \boldsymbol{D} = \boldsymbol{I}_{n\times n} \tag{8}$$

Let the matrix sum $(\boldsymbol{A} + \boldsymbol{W} * \boldsymbol{F})$ be denoted by matrix $\boldsymbol{E}$ **as**,

$$\boldsymbol{E} = \boldsymbol{A} + \boldsymbol{W} * \boldsymbol{F} \tag{9}$$

From (8) and (9),

$$\boldsymbol{E} * \boldsymbol{D} = \boldsymbol{I}_{n\times n} \tag{10}$$

Matrix $\boldsymbol{E}$ that satisfies (10) is called the generalized inverse [25] of $\boldsymbol{D}$ and is given by (9). Since $\boldsymbol{E}$ depends on $\boldsymbol{W}$ which is random matrix that can take different distinct values, $\boldsymbol{E}$ also can take different values.

The matrix $\boldsymbol{W}$ has $n\times(m-n)$ elements which belong to $\boldsymbol{Z}_p$. Each element can take p distinct values from 0 to (p – 1) and thus theoretically, the number of possible distinct ways in which $\boldsymbol{W}$ can be constructed is $p^{n\times(m-n)}$. Let us represent the i^{th} instance of $\boldsymbol{W}$ by $\boldsymbol{W}\{i\}$ where i can take values in the range 1 to $p^{n\times(m-n)}$. Then from (9), the corresponding $\boldsymbol{E}\{i\}$ can be rewritten as,

$$\boldsymbol{E}\{\boldsymbol{i}\} = \boldsymbol{A} + \boldsymbol{W}\{\boldsymbol{i}\}*\boldsymbol{F} \tag{11}$$

for i = 1 to $p^{n\times(m-n)}$. In terms of the i^{th} version of $\boldsymbol{E}$, in (10) can be rewritten as,

$$\boldsymbol{E}\{i\}*\boldsymbol{D} = \boldsymbol{I}_{n\times n} \tag{12}$$

We use $\boldsymbol{E}\{1\}$, $\boldsymbol{E}\{2\}$,…, $\boldsymbol{E}\{i\}$,… as the encryption matrices which are obtained from (11), by correspondingly choosing $\boldsymbol{W}\{1\}$, $\boldsymbol{W}\{2\}$,…, $\boldsymbol{W}\{i\}$ and so on. For good security we choose $\boldsymbol{W}\{i\}$ such that $\boldsymbol{E}\{i\}$ is non-sparse. There is no rigid rule in selecting the order $\boldsymbol{W}\{1\}$, $\boldsymbol{W}\{2\}$,…, $\boldsymbol{W}\{i\}$. The first randomly selected $\boldsymbol{W}$ is denoted as $\boldsymbol{W}\{1\}$, the second one as $\boldsymbol{W}\{2\}$, the i^{th} one is called $\boldsymbol{W}\{i\}$. The intention of generating different $\boldsymbol{E}\{i\}$'s is to use dissimilar $\boldsymbol{E}\{i\}$'s for successive encryptions to avoid chosen plain text attack. $\boldsymbol{E}\{i\}$'s are the left modular inverses [24] of $\boldsymbol{D}$.

Security of the Encryption Keys: By knowing the decryption key $\boldsymbol{D}$, the encryption key $\boldsymbol{E}\{i\}$ cannot be determined as there are $p^{n\times(m-n)}$ possible values for $\boldsymbol{E}\{i\}$. The probability of correct guessing the present $\boldsymbol{E}\{i\}$ is thus $\frac{1}{p^{n\times(m-n)}}$ which will be a very small fraction when p and (m–n) are large. In our examples (m–n) is taken as 2. Larger values of p and (m–n) can provide higher degree of security for the encryption key. In example 2, two samples of $\boldsymbol{W}\{i\}$'s and the corresponding $\boldsymbol{E}\{i\}$'s are generated for a given $\boldsymbol{D}$ and it is shown that the product of encryption key and the decryption key results in the identity matrix.

***Example* 2**: Let m = 3, n = 2 and the modulus p =11 and D $= \begin{bmatrix} 1 & 2 \\ 3 & 5 \\ 10 & 7 \end{bmatrix}$. *T*he modular null space of $\boldsymbol{D}$ is found to be, $\boldsymbol{F}$ = [7, 9, 1]. Using (3), matrix $\boldsymbol{A}$ is found to be, $\boldsymbol{A} = (\boldsymbol{D}^{\mathbf{T}} * \boldsymbol{D})^{-\mathbf{1}} * \boldsymbol{D}^{\mathbf{T}} = \begin{bmatrix} 8 & 3 & 5 \\ 10 & 8 & 1 \end{bmatrix}$.

Let us take $\boldsymbol{W}\{1\}$ as $\boldsymbol{W}\{1\} = \begin{bmatrix} 4 \\ 7 \end{bmatrix}$.

Then using (11), we get $\boldsymbol{E}\{1\}$ as $\boldsymbol{E}\{1\} = \begin{bmatrix} 8 & 3 & 5 \\ 10 & 8 & 1 \end{bmatrix} + \begin{bmatrix} 4 \\ 7 \end{bmatrix} * [7 \quad 9 \quad 1] = \begin{bmatrix} 36 & 39 & 9 \\ 59 & 71 & 8 \end{bmatrix}$.

After taking the mod with p = 11, matrix $\boldsymbol{E}\{1\} = \begin{bmatrix} 3 & 6 & 9 \\ 4 & 5 & 8 \end{bmatrix}$.

Now, it can be verified that $\boldsymbol{E}\{1\} * \boldsymbol{D} = \begin{bmatrix} 3 & 6 & 9 \\ 4 & 5 & 8 \end{bmatrix} * \begin{bmatrix} 1 & 2 \\ 3 & 5 \\ 10 & 7 \end{bmatrix} = \begin{bmatrix} 111 & 99 \\ 99 & 89 \end{bmatrix} (mod\ 11) = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$.

Similarly, taking $\boldsymbol{W}\{2\} = \begin{bmatrix} 8 \\ 6 \end{bmatrix}$, matrix $\boldsymbol{E}\{2\}$ is found to be, $\boldsymbol{E}\{2\} = \begin{bmatrix} 9 & 9 & 2 \\ 8 & 7 & 7 \end{bmatrix}$.

It can be verified that $\boldsymbol{E}\{2\} * \boldsymbol{D} = \begin{bmatrix} 9 & 9 & 2 \\ 8 & 7 & 7 \end{bmatrix} * \begin{bmatrix} 1 & 2 \\ 3 & 5 \\ 10 & 7 \end{bmatrix} = \begin{bmatrix} 56 & 77 \\ 99 & 100 \end{bmatrix} (mod\ 11) = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$

2.2.4. Representation of data to be encrypted

In HSDA, the data to be encrypted are bio-medical samples like BP, sugar level, pulse rate, body temperature and so on. The actual values may be integers for BP measurements or decimal numbers with fractional parts for body temperature (in Fahrenheit) measurements. In our work, the fixed-point representation for decimal numbers is used. For the integer part, L digits are used and for the fractional part, K digits are used as shown in (13). Consider a decimal number represented by $g(L, K)$ as follows.

$$g(L,K) = \underbrace{g(1)\,g(2)\,\text{----}\,g(L)}_{\text{Integer Part: } L \text{ digits}} \bullet \overbrace{g(L+1)\,g(L+2)\,\text{----}\,g(L+K)}^{\text{Fractional Part: } K \text{ digits}} \tag{13}$$

Here, the weights of the integer part are $[10^{L-1},\ 10^{L-2}, \ldots,\ 10^{0}]$ and the weights of the fractional part are $[10^{-1},\ 10^{-2}, \ldots,\ 10^{-K}]$ in that order. The concatenated weight vector represented by $\boldsymbol{V}(L, K)$ is defined as,

$$\boldsymbol{V}(L,K) = [10^{L-1}, 10^{L-2}, \ldots, 10^{0},\ \ 10^{-1}, 10^{-2}, \ldots, 10^{-K}]^T \tag{14}$$

In (14) superscript *T* represents transpose operation. The size of column vector $\boldsymbol{V}(L, K)$ is $(L+K)$x1. The equivalent row vector of *g*, represented by $\boldsymbol{G}(L, K)$ be defined as,

$$\boldsymbol{G}(L, K) = [g(1), g(2), \text{----}, g(L), g(L+1), g(L+2), \text{----}, g(L+K)] \quad (15)$$

In (14) and (15), parameter *L* represents the length of the integer part in digits, while the length of the fractional part is represented by *K*. Vector $\boldsymbol{G}(L, K)$ is a vector of size 1x$(L+K)$. Each element of $\boldsymbol{G}(L, K)$ is a decimal digit in the range 0 to 9. In (15) implies that the *j*th element of $\boldsymbol{G}(L, K)$ is obtained as the *j*th decimal digit of *g*, for $j = 1$ to $(L+K)$, counting from left to right, ignoring the decimal point. Thus, vector $\boldsymbol{G}(L, K)$ is the equivalent row vector of decimal number $g(L, K)$. The process of generating vector $\boldsymbol{G}(L, K)$ from $g(L, K)$ can be called as the decimal digit decomposition. From (13), (14) and (15), it can be seen that,

$$g(L, K) = \boldsymbol{G}(L, K) * \boldsymbol{V}(L, K) \quad (16)$$

In (16), the size of *g* is (1x$(L+K)$) x($(L+K)$ x1 = 1 which is a scalar. When there is no ambiguity, *g* is used in place of $g(L, K)$. Example 3 illustrates the representation of a decimal number by its equivalent row vector.

***Example* 3.** Let the given decimal number be, $g(4, 4)$ = 2345•6789 , Here, $L = 4$ and $K = 4$. Then, $\boldsymbol{G}(4, 4)$ = [2, 3, 4, 5, 6, 7, 8, 9]. In this case, $\boldsymbol{V}(4, 4) = [10^3, 10^2, 10^1 10^0, \ \mathbf{10^{-1}}, \mathbf{10^{-2}}, \mathbf{10^{-3}}\ \mathbf{10^{-4}}]^T$. From $\boldsymbol{G}(4, 4)$ and $\boldsymbol{V}(4, 4)$, the decimal equivalent *g* is calculated as,

$g = \boldsymbol{G}(4, 4) * \boldsymbol{V}(4, 4) = [2, 3, 4, 5, 6, 7, 8, 9] * [10^3, 10^2, 10^1, 10^0, \ \mathbf{10^{-1}}, \mathbf{10^{-2}}, \mathbf{10^{-3}}, \mathbf{10^{-4}}]^T$

$= 2 * 10^3 + 3 * 10^2 + 4 * 10^1 + 5 * 10^0 + 6 * 10^{-1} + 6 * 10^{-2} + 8 * 10^{-3} + 9 * \ 10^{-4}$ = 2345•6789.

In our proposed method, HSDA, we use integer row vectors like $\boldsymbol{G}(L, K)$ as the basic plaintext data to be encrypted. Depending on the nature of the problem, the length of the row vector is fixed at $(L+K)$. The values of *L* and *K* are designer's decision and depend on the range of the data values of the problem under consideration.

2.2.5. Representation of a negative decimal number

Let *h* be a negative decimal number as $h = -g$ and let $\boldsymbol{G}(L, K)$ be the row vector which is equivalent of *g*. Then, obviously, $\boldsymbol{H}(L, K)$ = row vector of $h = -\boldsymbol{G}(L,K)$. As an example, let $h = -567.23$. Then, $\boldsymbol{H}(3, 2) = -[5, 6, 7, 2, 3]$. From $\boldsymbol{H}(3, 2)$, the corresponding decimal number *h* is obtained based on (16) as,

$h = \boldsymbol{H}(3, 2) * \boldsymbol{V}(3, 2) = -[5, 6, 7, 2, 3] * [100, 10, 1, 0.1, 0.01]^T = -(500 +60+7+ 0.2+0.03) = -567.23$. Note that, $-[5, 6, 7, 2, 3] = [-5, -6, -7, -2, -3]$.

In general, the range of the elements of a row vector corresponding to a positive or negative decimal number, is –9 to +9. A few numerical examples are shown in Table 3 for $L = 6$.

Table 3. Row vector representation of decimal numbers

Sl. No.	Row Elements → / Decimal weights →	Row Vector $\boldsymbol{G}(L, K)$ of size 1x$(L+K)$ with $L = 6$ and $K = 4$									
		g(1)	g(2)	g(3)	g(4)	g(5)	g(6)	g(7)	g(8)	g(9)	G(10)
		10^5	10^4	10^3	10^2	10^1	10^0	10^{-1}	10^{-2}	10^{-3}	10^{-4}
	Decimal number (g) ↓	Integer part							Fractional part		
1	235.46	0	0	0	2	3	5	4	6	0	0
2	999999.9999 (+ve max)	9	9	9	9	9	9	9	9	9	9
3	–89.0305	0	0	0	0	–8	–9	0	–3	0	–5
4	–999999.9999 (–ve max)	–9	–9	–9	–9	–9	–9	–9	–9	–9	–9
5	468039	4	6	8	0	3	9	0	0	0	0
6	0.38	0	0	0	0	0	0	3	8	0	0

If the number of decimal digits of the integer part is less than *L*, leading zeros are inserted such that the total number decimal digit is equal to *L*. Similarly, if the number of digits in the fractional part is less than *K*, trailing zeros are appended to make it equal to *K*. For example, in Table 3, for Serial No. 1 with data = 235.46, three leading zeros and two trailing zeros are inserted. For Serial No. 3 with data = –89.0305 four leading zeros are inserted.

2.2.6. Encryption of data

Let the data to be encrypted be decimal number *g* which can be positive or negative, which is represented by its row vector equivalent $\boldsymbol{G}(L, K)$ of size 1x$(L+K)$. This $\boldsymbol{G}(L, K)$ is extended by appending

random scalar element R for encryption process and scalar element S representing digital signature for signature verification process to get the Ready to Encrypt Vector $\boldsymbol{Q}$ as,

$$\boldsymbol{Q} = [\boldsymbol{G}(L, K), R, S] = [\boldsymbol{G}, R, S] \quad (17)$$

When there is no ambiguity, $\boldsymbol{G}(L, K)$ is referred as $\boldsymbol{G}$ for easy writing. In (17), $\boldsymbol{Q}$ is the augmented version of $\boldsymbol{G}$ and the size of $\boldsymbol{Q}$ is 1x(L+K+2), because two extra elements are appended to $\boldsymbol{Q}$. Element R is the randomizing element and S is the signature element. Both of them belong to $SFF_p\backslash\{0\}$. The purpose of R and S and the selection of their values will be discussed in section 3A(i).

In HSDA, the size of the encryption vector say $\boldsymbol{E}\{1\}$, is $n \times m$ and in the design of encryption/decryption scheme, n is chosen equal to L+K+2. That is,

$$n = L + K + 2 \quad (18)$$

Then the size of $\boldsymbol{Q}$ is $1 \times n$ {which is same as 1x (L+K+2)}. The encryption of $\boldsymbol{Q}$ is carried out by simply post multiplying $\boldsymbol{Q}$ in signed Finite Field by $\boldsymbol{E}\{i\}$ (say for i = 1) to get the ciphertext $\boldsymbol{C}$ which belongs to SFF_p as,

$$\boldsymbol{C} = \text{signMod}(\boldsymbol{Q} * \boldsymbol{E}\{i\},\ p) \quad (19)$$

The size of $\boldsymbol{C}$ is (1xn)x(nxm) = (1xm) and $\boldsymbol{C}$ belongs to SFF_p.

2.2.7. Decryption of data

The data to be obtained after decryption is the decimal number g which could be positive or negative, and it is represented by its row vector equivalent $\boldsymbol{G}$ (L, K) of size 1x (L+K). Decryption of $\boldsymbol{C}$ is carried out as,

$$\text{dec}(\boldsymbol{C}) = \text{signMod}(\boldsymbol{C} * \boldsymbol{D},\ p) \quad (20)$$

Substituting for $\boldsymbol{C}$ from (19) On the RHS of (20) and simplifying gives,

$$\text{dec}(\boldsymbol{C}) = \text{signMod}(\text{signMod}(\boldsymbol{Q} * \boldsymbol{E}\{\boldsymbol{i}\},\ p) * \boldsymbol{D},\ p) = \text{signMod}(\boldsymbol{Q} * \boldsymbol{E}\{\boldsymbol{i}\} * \boldsymbol{D},\ p) \quad (21)$$

From (12), $\boldsymbol{E}\{i\}*\boldsymbol{D} = \boldsymbol{I}_{n\times n}$. Hence,

$$\text{dec}(\boldsymbol{C}) = \text{signMod}(\boldsymbol{Q},\ p) = \boldsymbol{Q} \quad (22)$$

$\text{signMod}(\boldsymbol{Q},\ p) = \boldsymbol{Q}$, because $\boldsymbol{Q}$ belongs to SFF_p. Thus dec($\boldsymbol{C}$) obtained using (22) recovers the original plaintext vector $\boldsymbol{Q}$. After stripping R and S from $\boldsymbol{Q}$, we get $\boldsymbol{G}$. From $\boldsymbol{G}$, its equivalent g is obtained using (16).

3. HOMOMORPHIC OPERATIONS ON BIOMEDICAL DATA

In HSDA, the sum and average aggregates of biomedical data in cipher domain are obtained using Homomorphic Operations. The ciphertexts are integers in SFF_p as specified by (19). These ciphertexts when decrypted, result in the plaintext. This is possible under certain types of encryptions and subsequent decryptions. Those special types of encryptions which are amenable to homomorphic operations are called homomorphic encryptions (HE). If along with addition, other arithmetic, algebraic operations are homomorphic, then the corresponding encryptions are designated as fully homomorphic encryptions (FHE) [6], [7]. In the following sections, homomorphic addition, subtraction, multiplication, division, and average operations are discussed along with signature verification and data authentication.

3.1. Homomorphic addition

The proposed Homomorphic Addition method used in HSDA is designated as HSDA_ADD. Consider two plaintext decimal numbers g and h of length at most L digits. Let their equivalents row vectors be $\boldsymbol{G}$ and $\boldsymbol{H}$ each with size 1x(L+K). Append $R1$, $S1$ and $R2$, $S2$ to $\boldsymbol{G}$ and $\boldsymbol{H}$ respectively to get $\boldsymbol{Q}1$ and $\boldsymbol{Q}2$ as,

$$\left.\begin{aligned} \boldsymbol{Q}1 &= [\boldsymbol{G}, R1, S1] \\ \boldsymbol{Q}2 &= [\boldsymbol{H}, R2, S2] \end{aligned}\right\} \quad (23)$$

The size of $\boldsymbol{Q}1$ as well as $\boldsymbol{Q}2$ is 1x(L+K+2) = 1xn. (Note that n = L+K+2). Let $\boldsymbol{C}1$ and $\boldsymbol{C}2$ be the encrypted ciphertexts obtained from $\boldsymbol{Q}1$ and $\boldsymbol{Q}2$ as,

$$\left.\begin{aligned} \boldsymbol{C}1 &= \text{signMod}(\boldsymbol{Q}1 * \boldsymbol{E}\{1\},\ p) \\ \boldsymbol{C}2 &= \text{signMod}(\mathbf{Q}2 * \mathbf{E}\{2\},\ p) \end{aligned}\right\} \quad (24)$$

Here, $\boldsymbol{E}\{1\}$ and $\boldsymbol{E}\{2\}$ are two different versions of $\boldsymbol{E}$ whose size is nxm. The size of $\boldsymbol{C}1$ or $\boldsymbol{C}2$ is (1xn)x(nxm) = 1xm.When there is no ambiguity, (24) can be simply rewritten as,

$$\left.\begin{aligned} \boldsymbol{C}1 &= \boldsymbol{Q}1 * \mathbf{E}\{1\} \\ \boldsymbol{C}2 &= \mathbf{Q}2 * \mathbf{E}\{2\} \end{aligned}\right\} \quad (25)$$

Here, $\boldsymbol{C}1$ and $\boldsymbol{C}2$ are of size 1xm and belong to SFF_p. Let us add $\boldsymbol{C}1$ and $\boldsymbol{C}2$ in SFF_p to get $\boldsymbol{C}3$ as,

$$\boldsymbol{C}3 = \boldsymbol{C}1 + \boldsymbol{C}2 = \boldsymbol{Q}1 * \boldsymbol{E}\{1\} + \boldsymbol{Q}2 * \boldsymbol{E}\{2\} \quad (26)$$

Now, the resultant Homomorphic addition is ciphertext $\boldsymbol{C}3$, whose size is 1xm is sent to the intended EU who decrypts $\boldsymbol{C}3$ as,

$$\boldsymbol{Q}3 = \text{signMod}(\boldsymbol{C}3 * \boldsymbol{D},\ p) = \boldsymbol{C}3 * \boldsymbol{D} \quad (27)$$

The size of $\boldsymbol{Q}3$ is (1xm)x(mxn) = 1xn. Here, the decrypter has already received the decryption matrix $\boldsymbol{D}$, during initialization of the session. Substituting for $\boldsymbol{C}3$ from (26) in (27) we get the decrypted output as,

$$\boldsymbol{Q}3 = (\boldsymbol{Q}1 * \boldsymbol{E}\{1\} + \boldsymbol{Q}2 * \boldsymbol{E}\{2\}) * \boldsymbol{D} = \boldsymbol{Q}1 * \boldsymbol{E}\{1\} * \boldsymbol{D} + \boldsymbol{Q}2 * \boldsymbol{E}\{2\} * \boldsymbol{D} \quad (28)$$

From (28) and (12), $\boldsymbol{Q}3 = \boldsymbol{Q}1 + \boldsymbol{Q}2$ (29)

Substituting for $\boldsymbol{Q}1$ and $\boldsymbol{Q}2$ from (23) in (29), we have,

$$\boldsymbol{Q}3 = [\boldsymbol{G}, R1, S1] + [\boldsymbol{H}, R2, S2] \quad (30)$$

Splitting $\boldsymbol{Q}3$ into 3 parts, we get,

$$\boldsymbol{Q}3 = [\boldsymbol{B}, R3, S3] \quad (31)$$

In (31), the size of $\boldsymbol{B}$ is 1x(L+K) while $R3$ and $S3$ are scalars. In fact B is the first (L+K) elements of $\boldsymbol{Q}3$. Hence $\boldsymbol{B}$ can be expressed using the colon notation of Matlab as,

$$\boldsymbol{B} = \boldsymbol{Q}3(1 : L+K) \quad (32)$$

From (31) and (30), we see that, $[\boldsymbol{B}, R3, S3] = [\boldsymbol{G}, R1, S1] + [\boldsymbol{H}, R2, S2]$ (33) From (33), the decrypted outputs in SFF_p are,

$$\boldsymbol{B} = \text{signMod}(\boldsymbol{G} + \boldsymbol{H},\ p) \quad (34)$$

$$R3 = \text{signMod}\ (R1 + R2,\ p) \quad (35)$$

$$S3 = \text{signMod}(S1 + S2,\ p) \quad (36)$$

$\boldsymbol{G}$ and $\boldsymbol{H}$ are vectors of decimal digits as in Table 3. Hence the range of the elements of $\boldsymbol{G}$ and $\boldsymbol{H}$ are from [–9 to +9]. Therefore, the range of the elements of their sum $\boldsymbol{B}$ is [–18 to +18]. Since the modulus p used in HSDA is large, the constraint $-\text{floor}\left(\frac{p-1}{2}\right) < -18 < = \text{elements of } \mathbf{B} < = 18 < \text{floor}\left(\frac{p-1}{2}\right)$ is satisfied and hence $\boldsymbol{B}$ belongs to SFF_p and there is no wraparound anomaly in the arithmetic operation $\boldsymbol{B} = \boldsymbol{G} + \boldsymbol{H}$. Therefore, $\boldsymbol{B}$ gives the correct result of addition as in normal algebra. During addition operation $R3$ and $S3$ are ignored.

3.1.1. Role of *R* and *S* in homomorphic encryption/decryption

Consider the case where scalars R and S are not introduced in forming $\boldsymbol{Q}$ from $\boldsymbol{G}$ as in (17). Then $\boldsymbol{Q} = \boldsymbol{G}$. Further, consider the scenario where $\boldsymbol{G}$ is an all zero vector when the corresponding $g = 0$ of size

1x(L+K). Then $\boldsymbol{Q}$ is also an all zero vector and the encrypted value of $\boldsymbol{Q}$ represented by $\boldsymbol{C}$ as given by (19) would also be an all zero vector. Thus, the encrypted ciphertext directly reveals the original plaintext instead of hiding it when the plaintext is zero. To overcome this zero-to-zero mapping, Scalar R is appended to $\boldsymbol{G}$, Then, the encrypted ciphertext of [$\boldsymbol{G}$, R] would be,

$$\boldsymbol{C} = \left[0_{1x(L+k)}, R\right] * \boldsymbol{E}\{i\}_{(L+K+1)xm} \quad (37)$$

In (37), the term, $0_{1x(L+k)}$ is the all zero $\boldsymbol{G}$ vector of size 1x(L+K) and the size of the encrypting matrix $\boldsymbol{E}\{i\}$ is (L+K+1) xm. In this case, $\boldsymbol{C}$ leaks the scaled-up value of the last row of the encryption key matrix $\boldsymbol{E}\{i\}$(L+K+1) xm. Hence the encryption key is compromised. To mitigate this, one more scalar S is appended to [$\boldsymbol{G}$, R] to get $\boldsymbol{Q}$ = [$\boldsymbol{G}$, R, S]. Here, S also serves as the signature verification parameter. Now, when $\boldsymbol{G}$ = $0_{1x(L+k)}$, the ciphertext $\boldsymbol{C}$ is,

$$\boldsymbol{C} = \left[0_{1x(L+k)}, R, S\right] * \boldsymbol{E}\{i\}_{(L+K+2)xm} = R^*[(L+K+1)^{\text{th}} \text{ row of } E\{i\}] + S^*[(L+K+2)^{\text{th}} \text{ row of } E\{i\}] \quad (38)$$

In this case, $\boldsymbol{C}$ is the weighted sum of the last two rows of $\boldsymbol{E}\{i\}$. Therefore, it is hard to recover the exact values of the last two rows of $\boldsymbol{E}\{i\}$. Apart from this, in HSDA_ADD, scalars $S1$ and $S2$ are used for verification of addition operation as well as signatures for authentication and will be explained in section 3A(ii). Scalar R which varies randomly from one encryption to the next encryption provides randomization of the cipher text that prevents plain text attack.

Once the sum vector $\boldsymbol{B}$ is obtained as given by (32), its decimal equivalent b is obtained based on (16) as $b = \boldsymbol{B}^*\boldsymbol{V}(K, L)$. The homomorphic addition has 3 stages as shown in Figure 2. Adder unit in cipher domain is implemented in a cloud server whereas Encryption operation is carried out by the data owner. The decrypter is the EU. During the initialization of the Homomorphic Addition session, the decrypter should have received the decryption matrix $\boldsymbol{D}$ and the scalar sum term designated by $S3_{\text{original}}$ as,

$$S3_{\text{original}} = \text{signMod}(S1+S2, p) \quad (39)$$

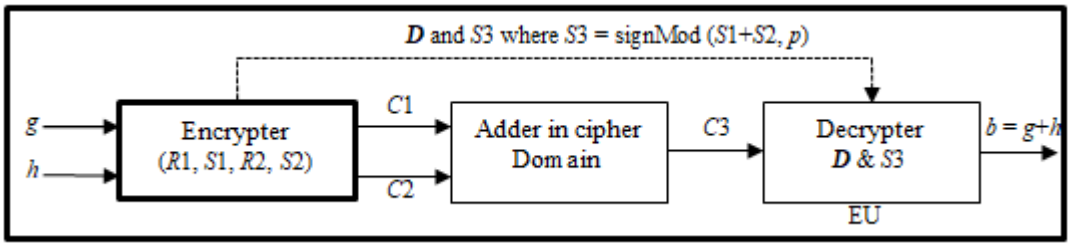

Figure 2. Homomorphic addition

3.1.2. Signature verification and authentication during addition

The EU, after decryption of $\boldsymbol{C}$3, gets $\boldsymbol{Q}$3 from which, its last element designated as $S3_{\text{dec}}$ is obtained as indicated in (36). Then the EU checks whether $S3_{\text{dec}}$ is exactly equal to $S3_{\text{original}}$. If there were no errors, $S3_{\text{dec}}$ would be equal $S3_{\text{original}}$. If $S3_{\text{dec}} \neq S3_{\text{original}}$, it indicates the presence of some computational error or that the input $\boldsymbol{C}$3 is altered or $\boldsymbol{C}$3 is not from an authentic source. The homomorphic addition algorithm of HSDA involves encryption, decryption and signature verification as given:

```
-------------------------------------------------------
Algorithm HSDA_ADD
-------------------------------------------------------
Inputs: Integers g and h to be added using homomorphic encryption.
Output: Homomorphically added ciphertext and its decrypted result, b = g + h
        //Encryption stage
 1.  Get vectors G and H from g and h as in (15)
 2.  Formulate Q1 and Q2 by appending suitable R1, S1 and R2, S2 as in (23)
 3.  Obtain C1 and C2 by encrypting Q1 and Q2 as in (24)
        //Encryption over
        //Addition at Homomorphic Adder in Cloud
 4.  Get sum C3 as, C3 = signMod(C1+C2, p)
        //Addition over. C3 is sent to the decrypter
        //Decryption
 5.  Get Q3 using the decryption key D as, Q3 = signMod(C3*D, p)
```

```
6.  Get S3dec as the last element of Q3
7.  If S3rec ≠ S3original
              Display "ERROR"
              Discard Q3 (and take any remedial action like 'request repeat' etc.)
              Goto step 10
       Else
8.  Get B by taking first (L+K) terms of Q3
9.  Get b using b = B*V(L, K)              //based on (16)
10. End
-----------------------------------------------------
```

The Encryption, homomorphic addition, and decryption of two numbers are illustrated in example 4.
Example 4: Here, L = 2, K =2, n = 6, m = 8 and p = 97. Decryption matrix $\boldsymbol{D}$ is created randomly. From $\boldsymbol{D}$, two encryption matrices $\boldsymbol{E}\{1\}$ and $\boldsymbol{E}\{2\}$ are generated as in (11).

$$Matrix\ D = \begin{bmatrix} 20 & 27 & 38 & 30 & 63 & 24 \\ 79 & 17 & 27 & 13 & 71 & 73 \\ 19 & 08 & 57 & 15 & 3 & 79 \\ 83 & 67 & 53 & 87 & 52 & 84 \\ 44 & 40 & 54 & 87 & 1 & 1 \\ 56 & 15 & 24 & 17 & 47 & 43 \\ 69 & 80 & 26 & 3 & 58 & 89 \\ 47 & 25 & 23 & 19 & 89 & 36 \end{bmatrix}$$

$$Matri\ E\{1\} = \begin{bmatrix} 81 & 89 & 3 & 11 & 31 & 32 & 55 & 54 \\ 72 & 53 & 30 & 67 & 59 & 68 & 58 & 32 \\ 1 & 57 & 95 & 20 & 86 & 50 & 32 & 64 \\ 30 & 54 & 66 & 81 & 23 & 67 & 26 & 33 \\ 41 & 65 & 6 & 75 & 75 & 91 & 18 & 39 \\ 71 & 37 & 83 & 23 & 88 & 42 & 46 & 71 \end{bmatrix} \quad MatriX\ E\{2\} = \begin{bmatrix} 74 & 11 & 70 & 23 & 93 & 4 & 46 & 85 \\ 88 & 26 & 75 & 38 & 61 & 7 & 73 & 43 \\ 34 & 24 & 92 & 12 & 87 & 51 & 35 & 42 \\ 22 & 87 & 83 & 27 & 61 & 63 & 49 & 42 \\ 28 & 20 & 95 & 94 & 85 & 51 & 69 & 12 \\ 96 & 23 & 96 & 45 & 2 & 19 & 70 & 65 \end{bmatrix}$$

It can be verified signMod ($\boldsymbol{E}\{1\}$ *$\boldsymbol{D}$, p) = signMod ($\boldsymbol{E}\{2\}$ *$\boldsymbol{D}$, p) = I_{6x6}. The two addends are taken as g = 63•79 and h = 89•65. Then $\boldsymbol{G}$ = [6, 3, 7, 9] and $\boldsymbol{H}$ = [8, 9, 6, 5]. Taking [$R1$, $S1$] = [23, 17] and [$R2$, $S2$] = [12, 19] we get,
$\boldsymbol{Q}1$ = [6, 3, 7, 9, 23, 17] and $\boldsymbol{Q}2$ = [8, 9, 6, 5, 12, 19]. From (24), $\boldsymbol{C}1$ = signMod($\boldsymbol{Q}1 * \boldsymbol{E}\{1\}$, p) and $\boldsymbol{C}2$ = signMod($\boldsymbol{Q}2 * \boldsymbol{E}\{2\}$, p). The ciphertexts $\boldsymbol{C}1$ and $\boldsymbol{C}2$ are found as,
$\boldsymbol{C}1$ = [25 16 6 -46 28 -15 24 -29] and $\boldsymbol{C}2$ = [-22 26 25 0 -23 40 -48 -2] Now, $\boldsymbol{C}3$ = signMod ($\boldsymbol{C}1$+$\boldsymbol{C}2$, p) gives, $\boldsymbol{C}3$ = [3 42 31 -46 5 25 -24 -31]. Decryption of $\boldsymbol{C}3$ using (27) gives, $\boldsymbol{Q}3$ = signMod ($\boldsymbol{C}3$*$\boldsymbol{D}$, p) = [14 12 13 14 35 36]. Vector $\boldsymbol{B}$ is obtained from $\boldsymbol{Q}3$, by taking the first 4 (here, L+K = 4) elements of $\boldsymbol{Q}3$ as, $\boldsymbol{B}$ = [14 12 13 14]. It can be observed from normal algebra that, $\boldsymbol{Q}3$ = $\boldsymbol{Q}1$+$\boldsymbol{Q}2$. Now the decimal integer b is obtained based on (16) as, b = $\boldsymbol{B}$*$\boldsymbol{V}$ (2, 2) = [14, 12, 13, 14] *[10, 1, 0.1, 0.01]^T = 153.44 which is same as g+h.

3.1.3. Addition with multiple addends

Consider the addition $b = u(1) + u(2) + \ldots + u(j) + \ldots + u(J)$. Let $\boldsymbol{U}(j)$ be the row vector corresponding to $u(j)$. Let the corresponding cipher value be $\boldsymbol{C}(j)$ = signMod($\boldsymbol{U}(j)$*$\boldsymbol{E}\{j\}$, p) for j = 1 to J. Then addition $\boldsymbol{C}3 = \boldsymbol{C}(1) + \boldsymbol{C}(2) + \ldots + \boldsymbol{C}(j) + \ldots + \boldsymbol{C}(J)$ is carried out cumulatively as,

$\boldsymbol{C}3$ = 0;
for j = 1: J
$\boldsymbol{C}3$ = signMod($\boldsymbol{C}3$+$\boldsymbol{C}(j)$, p);
End

Then, $\boldsymbol{Q}3$ = signMod ($\boldsymbol{C}3$*$\boldsymbol{D}$, p) = [$\boldsymbol{B}$, R, S]. Here, $\boldsymbol{B} = \boldsymbol{U}(1) + \boldsymbol{U}(2) + \ldots + \boldsymbol{U}(j) + \ldots + \boldsymbol{U}(J)$. The range of element $\boldsymbol{U}(j)$ is from –9 to +9 for j = 1 to J. When all such elements are added the range would be –9*J to +9*J and this range has to be within the SFF_p range, $-\text{floor}\left(\frac{p-1}{2}\right)\ to + \text{floor}\left(\frac{p-1}{2}\right)$ for correct result. Therefore, the value of p should be chosen such that,

$$-\text{floor}\left(\frac{p-1}{2}\right) \leq -9*J < 9*J \leq \text{floor}\left(\frac{p-1}{2}\right) \quad (40)$$

3.2. Homomorphic subtraction

Homomorphic Subtraction is similar to addition except that $\boldsymbol{C}3$ is taken as $\boldsymbol{C}3 = \boldsymbol{C}1 - \boldsymbol{C}2$ in step 4 of Algorithm HSDA_ADD to get $b = \boldsymbol{g} - h$. Another approach is to treat $g - h$ as $g + (-h)$ which is the addition operation that can be carried out by HSDA_ADD.

3.3. Multiplication (non-homomorphic) of decimal numbers using their equivalent vectors

The basic non homomorphic multiplication of two decimal numbers is explained in Example 5. Here equivalent vectors of two decimal numbers are used.

Example 5: Consider two decimal numbers g and h as, g = [23•4] and h = [45•6]. Let g be the multiplicand and h be the multiplier. Then g and h can be represented in terms of powers of 10 as,

$g = (10^1 * 2 + 10^0 * 3 + 10^{-1} * 4)$ and $h = (4 * 10^1 + 5 * 10^0 + 6 * 10^{-1})$

The weighted addition forms of g and h can be further expanded in terms of the vector product as,

$$g = [10 \quad 1 \quad 0.1] * \begin{bmatrix} 2 \\ 3 \\ 4 \end{bmatrix} \text{ and } h = [4 \quad 5 \quad 6] * \begin{bmatrix} 10 \\ 1 \\ 0.1 \end{bmatrix}$$

Then, their product b is,

$$b = g * h = (23 \bullet 4) * (45 \bullet 6) = [10 \quad 1 \quad 0.1] * \begin{bmatrix} 2 \\ 3 \\ 4 \end{bmatrix} * [4 \quad 5 \quad 6] * \begin{bmatrix} 10 \\ 1 \\ 0.1 \end{bmatrix} \tag{41}$$

$$\text{Let } \boldsymbol{V}(2,1) = \begin{bmatrix} 10 \\ 1 \\ 0.1 \end{bmatrix}. \text{ Then, (41) can be expressed as, } b = \boldsymbol{V}(2,1)^T * \begin{bmatrix} 2 \\ 3 \\ 4 \end{bmatrix} * [4 \quad 5 \quad 6] * \boldsymbol{V}(2,1) \tag{42}$$

Let $\boldsymbol{G}$ and $\boldsymbol{H}$ be the vector formats of g and h as, $\boldsymbol{G}$ = [2, 3, 4] and $\boldsymbol{H}$ = [4, 5, 6]. Then, (42) can be rewritten as,

$$b = \boldsymbol{V}(2,1)^T * \boldsymbol{G}^T * \boldsymbol{H} * \boldsymbol{V}(2,1) = \boldsymbol{V}(2,1)^T * (\boldsymbol{G}^T * \boldsymbol{H}) * \boldsymbol{V}(2,1) \tag{43}$$

In this example, $(\boldsymbol{G}^T * \boldsymbol{H}) = \begin{bmatrix} 2 \\ 3 \\ 4 \end{bmatrix} * [4 \quad 5 \quad 6] = \begin{bmatrix} 8 & 10 & 12 \\ 12 & 15 & 18 \\ 16 & 20 & 24 \end{bmatrix}$

According to (43), It can be verified that,

$$b = [10 \quad 1 \quad 0.1] * \begin{bmatrix} 8 & 10 & 12 \\ 12 & 15 & 18 \\ 16 & 20 & 24 \end{bmatrix} * \begin{bmatrix} 10 \\ 1 \\ 0.1 \end{bmatrix} = [10 \quad 1 \quad 0.1] * \begin{bmatrix} 91.2 \\ 136.8 \\ 182.4 \end{bmatrix} = [912 + 136.8 + 18.24] = 1067.04$$

The value of b is same as the direct multiplication of 23•4*45•6.

An important feature of matrix $(\mathbf{G}^T * \mathbf{H})$ is that all its elements are the product of two single decimal digits. Hence the range of each element is from – (9*9) to + (9*9). That is, from –81 to +81. This factor will be utilized later in homomorphic multiplication operation. Example 6 shows that in general when g and h are (L+K) digit decimal numbers with L digit integer part and K digit fractional part, the product b = g*h can be obtained as,

$$b = g * h = \boldsymbol{V}(L,K)^T * \boldsymbol{G}^T * \boldsymbol{H} * \boldsymbol{V}(L,K) = \boldsymbol{V}(L,K)^T * (\boldsymbol{G}^T * H) * \boldsymbol{V}(L,K)$$

when there is no ambiguity, $\boldsymbol{V}(L, K)$ can be simply written as $\boldsymbol{V}$. Then the above relation reduces to,

$$b = g * h = \boldsymbol{V}(L,K)^T * (\boldsymbol{G}^T * \boldsymbol{H}) * \boldsymbol{V}(L,K) \tag{44}$$

where $\boldsymbol{G}$ and $\boldsymbol{H}$ are the row vectors of g and h respectively and $\boldsymbol{V}$ (L, K) is the decimal weight vector of size (L+K) x1 as given by (14). And the homomorphic multiplication is based on (44).

3.4. Fully homomorphic multiplication

In this case, both multiplicand and the multiplier are ciphertexts at Homo-Multiplier unit in cloud server. Let g and h be the multiplicand and multiplier in plain text as in section 3C. Let $\boldsymbol{G}$ and $\boldsymbol{H}$ be the row vectors of g and h respectively, with size 1x(L+K) based on (15).

Encryption: the encrypter generates two ciphertexts $\boldsymbol{C}1$ and $\boldsymbol{C}2$ as,

$$\boldsymbol{C}1 = \text{signMod}(\boldsymbol{G} * \boldsymbol{E}\{i\},\ p) \tag{45}$$

$$\boldsymbol{C}2 = \text{signMod}(\boldsymbol{H} * \boldsymbol{E}\{j\},\ p) \tag{46}$$

The size of $C1$ as well as $C2$ is (1x(L+K)) x((L+K) xm) = 1xm. Vectors $\boldsymbol{C}1$ and $\boldsymbol{C}2$ are sent to the homomorphic multiplier unit at Cloud.

Multiplication: Fully Homomorphic Multiplication is carried out by the Homo-Multiplier unit at cloud server as,

$$\boldsymbol{Y} = \text{signMod}(\boldsymbol{C}1^{T} * \boldsymbol{C}2,\ p) \quad (47)$$

The size of $\boldsymbol{Y}$ is (mx1) x(1xm) = mxm. Matrix $\boldsymbol{Y}$ is sent to the decrypter at EU

Decryption: Decryption of $\boldsymbol{Y}$ is carried out at EU as,

$$\boldsymbol{B} = \text{signMod}(\boldsymbol{D}^{\mathrm{T}} * \boldsymbol{Y} * \boldsymbol{D},\ p) \quad (48)$$

The size of $\boldsymbol{B}$ is (nxm)x(mxm)x(mxn) = nxn. On substituting for $\boldsymbol{Y}$ from (47) in (48), we get,

$$\boldsymbol{B} = \text{signMod}(\boldsymbol{D}^{\mathrm{T}} * \boldsymbol{C}1^{T} * \boldsymbol{C}2 * \boldsymbol{D},\ p) \quad (49)$$

Substituting for $\boldsymbol{C}1$ and $\boldsymbol{C}2$ from (45) and (46) in (49), we get,

$$\boldsymbol{B} = \text{signMod}(\boldsymbol{D}^{\mathrm{T}} * (\boldsymbol{G} * \boldsymbol{E}[i\})^{T} * \boldsymbol{H} * \boldsymbol{E}\{j\}\boldsymbol{D}, p) \quad (50)$$

On replacing $(\boldsymbol{G} * \boldsymbol{E}[i\})^{T}$ by $\boldsymbol{E}[i\}^{T} * \boldsymbol{G}^{T}$, we have,

$$\boldsymbol{B} = \text{signMod}(\boldsymbol{D}^{\mathbf{T}} * \boldsymbol{E}[i\}^{T} * \boldsymbol{G}^{T} * \boldsymbol{H} * \boldsymbol{E}\{j\} * \boldsymbol{D}, p) \quad (51)$$

From (12), $\boldsymbol{E}\{j\} * \boldsymbol{D} = \boldsymbol{I}_{nxn}$ and $\boldsymbol{D}^{\mathbf{T}} * \boldsymbol{E}[\boldsymbol{i}\}^{\boldsymbol{T}} = \boldsymbol{I}_{nxn}$. Therefore, (51) gets simplified as,

$$\boldsymbol{B} = \text{signMod}(\boldsymbol{G}^{T} * \boldsymbol{H}, p) \quad (52)$$

Since the elements of matrix $(\boldsymbol{G}^{\mathrm{T}} * \boldsymbol{H})$ are in the range -81 to +81 as explained in 2.2H(vii) and p is a large number, $\text{signMod}(\boldsymbol{G}^{T} * \boldsymbol{H}, p) = \boldsymbol{G}^{\boldsymbol{T}} * \boldsymbol{H}$ itself. Therefore by (52) reduces to,

$$\boldsymbol{B} = (\boldsymbol{G}^{T} * \boldsymbol{H}) \quad (53)$$

From (44) and (53), $\mathrm{b} = \mathrm{g} * \mathrm{h} = \boldsymbol{V}^{\mathrm{T}} * (\boldsymbol{G}^{\mathrm{T}} * \boldsymbol{H}) * \boldsymbol{V} = \boldsymbol{V}^{\mathrm{T}} * \boldsymbol{B} * \boldsymbol{V}$ (54)

In (54), note that $\boldsymbol{H}$*$\boldsymbol{V}$ = h and $\boldsymbol{V}^{T} * G^{T} = g$ based on (16). Fully homomorphic multiplication is given in the following algorithm.

```
-------------------------------------------
Algorithm HOMO_MULT_FULL
-------------------------------------------
Inputs: Plaintext decimal numbers, g and h (selected by the sender) with integer part
length = L and fractional             part length = K, Output: b = g*h
//Encryption stage
 1.  Sender gets row vectors G and H from g and h based on (15)
 2.  Sender encrypts G and H to get C1 and C2 according to (45) and (46)
     //Encryption over
     //Multiplication at Homomorphic unit in Cloud sever
 3.  Homomorphic unit in Cloud server calculates matrix Y as given by (47)
     //Multiplication over. Y is sent to the decrypter (End User)
     //Decryption at End User
 4.  EU gets B using (48)
 5.  EU calculates b from B according to (53) as b = V^T * B * V = g * h    //uses non-modular
     algebra
 6.  End
```

In homomorphic multiplication also, the row vectors $\boldsymbol{G}$ and $\boldsymbol{H}$ can be augmented with randomizing parameter R and the signature verification parameter S to provide better security and authentication. Example 6 shows the numerical values of 4 digit Fully Homomorphic Multiplication without R and S.

Example 6: The modulus p of SFF_{p} is p = 997. The decryption matrix $\boldsymbol{D}$ of size mxn where m = 6 and n = 4 is generated. The corresponding inverse matrices $\boldsymbol{E}\{1\}$ and $\boldsymbol{E}\{2\}$ are calculated using (11). The plain text multiplier and multiplicand g and h are taken as g = 63•79 and h = 89•65 respectively. Here the length of integer part and fractional part of g and h are, L = 2 and K =2 respectively.

$$Matrix\ \boldsymbol{D} = \begin{bmatrix} 20 & 69 & 40 & 57 \\ 79 & 47 & 15 & 53 \\ 19 & 27 & 80 & 54 \\ 83 & 17 & 25 & 24 \\ 44 & 08 & 38 & 26 \\ 56 & 67 & 27 & 23 \end{bmatrix}$$

$$Matrix\ \boldsymbol{E}\{1\} = \begin{bmatrix} 191 & 432 & 379 & 194 & 679 & 616 \\ 666 & 294 & 780 & 840 & 408 & 615 \\ 595 & 520 & 824 & 018 & 762 & 321 \\ 191 & 141 & 671 & 941 & 191 & 720 \end{bmatrix} \qquad Matrix\ \boldsymbol{E}\{2\} = \begin{bmatrix} 576 & 47 & 426 & 697 & 705 & 578 \\ 655 & 608 & 732 & 248 & 450 & 645 \\ 506 & 944 & 662 & 511 & 778 & 366 \\ 471 & 386 & 260 & 848 & 170 & 764 \end{bmatrix}$$

$$Matrix\ \boldsymbol{Y} = \begin{bmatrix} 199 & -464 & 266 & -288 & -43 & 196 \\ 322 & 355 & -255 & -92 & 280 & 254 \\ -283 & 267 & 238 & -204 & 014 & -87 \\ -324 & -29 & -108 & 235 & -65 & -237 \\ 485 & -292 & 494 & -281 & 205 & 364 \\ -34 & -440 & 321 & 437 & -333 & 289 \end{bmatrix} \qquad \text{Matrix } \mathbf{B} = \begin{bmatrix} 48 & 54 & 36 & 30 \\ 24 & 27 & 18 & 15 \\ 56 & 63 & 42 & 35 \\ 72 & 81 & 54 & 45 \\ 48 & 54 & 36 & 30 \\ 24 & 27 & 18 & 15 \end{bmatrix}$$

The corresponding $\boldsymbol{G}$ and $\boldsymbol{H}$ vectors are $\boldsymbol{G}$ =[6, 3, 7, 9] and $\boldsymbol{H}$ = [8, 9, 6, 5]. The ciphertexts generated after encryption using (45) and (46) are, $\boldsymbol{C}1$ =[55, 407, 469, 315, 387, 310] and $\boldsymbol{C}2$ = [–58, 481, 313, 159, 253, 493]. The product matrix $\boldsymbol{Y}$, in cipher domain is calculated using (47). Decryption of $\boldsymbol{Y}$ is carried out by the EU according to (48) and after decryption the matrix B obtained. It can be seen that the elements of $\boldsymbol{B}$ are within the range -81 to +81. We can also see that $\boldsymbol{B} = \boldsymbol{G}^T * \boldsymbol{H}$. Now b, the product of g and h is obtained using (54) and found to be 5718.7735.

3.5. Homomorphic division

Homomorphic Division (HOMO_DIVIDE) is achieved by multiplying the dividend by the reciprocal of the divisor. Let u and h be the plaintext divisor and dividend. Let $g = \left(\frac{1}{u}\right)$ be the reciprocal of u. Then $\left(\frac{h}{u}\right) = h*g$. In practice, $g = \left(\frac{1}{u}\right)$ is calculated to the required level of accuracy of decimal places depending on the nature of the problem. This process of division via multiplication is used to calculate the average of a given set of data homomorphically in the cipher domain.

3.6. Homomorphic average

Let the data items whose average has to be calculated are stored as cipher texts in Aggregator within the CH. Let u be the number of elements in the dataset. Then the homomorphic average (HA) is determined in the cipher domain by the homomorphic unit (HU) using HOMO_ADD and HOMO_DIVIDE operations with u as the divisor, as explained in section 3A& E. Example 7 explains the calculations of Homomorphic sum and average of biomedical data.

Example 7: Consider the plaintext dataset shown in Table 4 consisting of BP systolic, diastolic, body temperature and Pulse rate. In this example, the row vectors corresponding to body temperature, designated as $\boldsymbol{VT}\{j\}$'s are shown in Table 5 for j =1 to 8. The temperature data has 3 decimal digits for its integer part and one decimal place for the fractional part. Therefore for this data, L = 3 and K = 1. The values of *p=997, m=6, n=4, and* $\boldsymbol{D}$, $\boldsymbol{E}\{i\}$'s are same as in Example 6.

Table 4. Plaintext data

Sl No	BP-Sys	BP-Dia	Temp in ºF	Pulse Rate
1	106	75	98.4	50
2	120	80	97.2	68
3	115	73	95	60
4	123	85	96	65
5	150	90	99.7	78
6	155	95	100.3	68
7	103	69	101	100
8	110	75	96.5	110

Table 5. Temp $\boldsymbol{VT}\{j\}$'s

Symbol	Temp in ºF (L =3, K =1)
$VT\{1\}$	[0, 9, 8, 4]
$VT\{2\}$	[0, 9, 7, 2]
$VT\{3\}$	[0, 9, 5, 0]
$VT\{4\}$	[0, 9, 6, 0]
$VT\{5\}$	[0, 9, 9, 7]
$VT\{6\}$	[1, 0, 0, 3]
$VT\{7\}$	[1, 0, 1, 0]
$VT\{8\}$	[0, 9, 6, 5]

Table 6. Ciphertext Data

j	$\boldsymbol{EVT}\{j\}$'s
1	[-446 391 344 -496 -435 16]
2	[-426 -411 172 -402 415 249]
3	[-4 261 173 -326 -494 161]
4	[-406 -216 0 -308 268 482]
5	[-275 337 190 351 -97 -494]
6	[-233 -142 398 26 255 -215]
7	[-211 -45 206 212 444 -60]
8	[-448 489 364 409 226 94]

The encrypted values of $VT\{j\}$'s denoted by $EVT\{j\}$'s is obtained based on (45) as,

$$\boldsymbol{EVT}\{j\} = \text{signMod}\,(\boldsymbol{VT}\{j\}*\boldsymbol{E}\{i\}, p) \quad (55)$$

For j = 1 to 8. In each case $\boldsymbol{E}\{i\}$ can take different values as explained in 2.2C. The resulting $\boldsymbol{EVT}\{j\}$'s is shown in Table 6. The sum of these 8 vectors, $\boldsymbol{EVT}\{j\}$'s is calculated using the cumulative add procedure as given in 2.2H(v). The Signed Finite field (SFF_p) sum, in the cipher domain, represented by ***EST*** (Encrypted SUM of Temperatures), is found to be, ***EST*** = [-455 -333 -147 463 -415 233]. On decryption of ***EST***, we get ***DST*** (decrypted Sum of Temperature) in row vector format as, ***DST*** = signMod(***EST*****D***, *p*) = [2 54 42 21]
The corresponding decimal value is found based on (16) as,
dst = ***DST*** ***V*** (*L*, *K*) = [2 54 42 21] *[100, 10, 1, 0.1] ^{T} = 784.1

This result can be verified by taking the sum of the elements of column, 'Temp' in ^0F' in Table 4.
Calculation of homomorphic average in cipher domain: The un-decrypted ***EST,*** which represents the sum in cipher domain, is divided by divisor u = 8, where u is the total number of temperature readings. The division by 8 is same as multiplication by g = 1/8 = 0.1250. Now the matching decimal weight vector is,
V (0, 4) = [10^ (-1), 10^ (-2), 10^ (-3), 10^ (-4)] ^{T}. The row vector for g is ***G*** = [1, 2, 5, 0]. Encryption of ***G***, represented by ***C***, is calculated as, ***C*** = signMod (***G********E***{*i*}, *p*) = [428 001 215 -240 -487 -290]. The product of ***C*** and ***EST*** is carried out in cipher domain as, ***Y*** = signMod (***C*** ^{T}****EST***, *p*). The ***Y*** matrix *is* calculated and tabulated as shown below. This ***Y*** is sent to the *EU* to decrypt ***Y*** according to (48) to get ***B*** = signMod (***D*** ^{T}****Y*D***, *p*). Thus, the matrix **B** is obtained as shown below.

$$\text{Matrix } \boldsymbol{Y} = \begin{bmatrix} -325 & 47 & -105 & -239 & -154 & 24 \\ -455 & -333 & -149 & 463 & -415 & 233 \\ -119 & 189 & 299 & -155 & -492 & 245 \\ -470 & 160 & 385 & -453 & -100 & -88 \\ 251 & -340 & -195 & -159 & -286 & 187 \\ 346 & -139 & -241 & 325 & -287 & 226 \end{bmatrix} \qquad \text{Matrix } \boldsymbol{B} = \begin{bmatrix} 2 & 54 & 42 & 21 \\ 4 & 108 & 84 & 42 \\ 10 & 270 & 210 & 105 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

The average value b is obtained as, $b = \boldsymbol{V}(0, 4)^{\mathrm{T}} * \boldsymbol{B} * \boldsymbol{V}(3, 1) = 98.0125$ (56)

In (56), ***V*** (0,4) corresponds to the left product term g and ***V*** (3, 1) corresponds to the right product term *dst*.

4. RESULTS AND DISCUSSION

In HSDA, the size of encryption matrix $\boldsymbol{E}\{i\}$ is nxm, that of decryption key ***D*** is mxn and the size of the ciphertext is 1xm. In general $m = n$+2. Hence the sizes of the keys depend on n which is the length of the plaintext data vector. Therefore, the execution time of encryption, decryption and homo-multiplication increases with n. In the following sections, comparison of HSDA with other two methods are described.

4.1. Comparison with other methods

HSDA homo-multiplication is compared with other two existing methods by, Brakerski & Vaikuntanathan (BV) [15] and Hedglin, Phillips and Reilley (GSW_HPR) [16]. These two methods also use matrix keys for encryption and decryption. In BV method, the plaintext message space is binary {0, 1}. Hence, decimal numbers are converted to binary equivalents, at the input side of BV method and consequently, converted back to decimal format at the output side. Thus, the BV method has additional computational overhead. In GSW_HPR method, a large sized generator matrix is used for both encryption and decryption. This incurs a higher computational cost compared to HSDA and BV methods. In the following section, we consider the computational cost of homo-multiplication and the subsequent decryption.

4.2. Time complexities of homomorphic multiplication and subsequent decryption in HSDA, BV and GSW_HPR

Homomorphic multiplication in cipher domain is moderately more complex than addition in all the three cases. The execution time of homomorphic multiplication of the three methods is plotted in Figure 3 The plaintext multiplier as well as multiplicand is taken as n digit decimal number which is varied from 3 to 10. From Figure 3 it can be seen that, HSDA has higher execution speed compared to the other two methods. It is found that the percentage saving in execution time for homomorphic multiplication of HSDA is 22.87% with respect to BV method. Decryption process after Homomorphic multiplication has substantially higher time complexity than that of decryption of a single ciphertext. The Execution time for decryption in HSDA, BV and GSW_HPR methods is shown in Figure 4. It can be seen that, HSDA decryption time is lower compared to GSW_HPR method even though it is higher than that of BV method. This is expected as BV decryption process recovers the binary digits as its output and later it is converted into its equivalent integer.

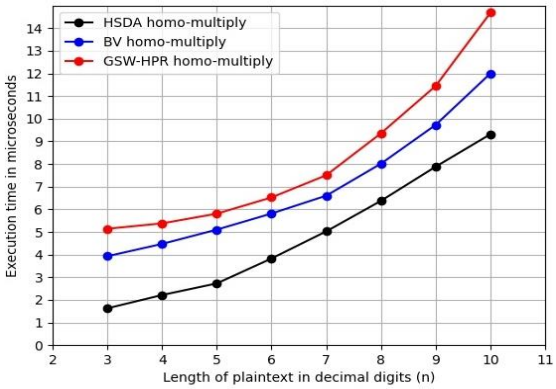

Figure 3. Execution time for homo-multiplication in HSDA, BV and GSW

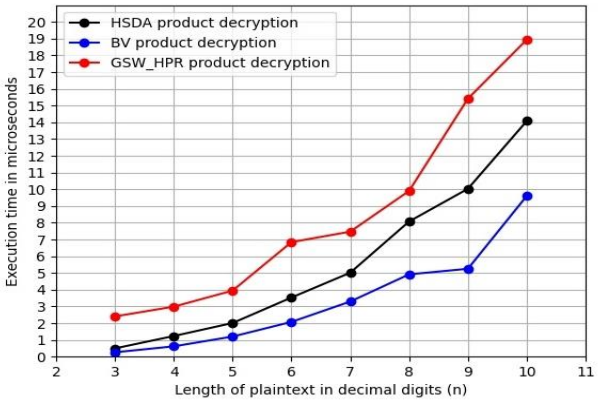

Figure 4. Execution time for decryption of homo-product in HSDA, BV and GSW

5. CONCLUSION

A novel, fully homomorphic-secure data aggregation (HSDA) scheme has been presented. It takes care of addition, subtraction, multiplication and division of decimal numbers with integer as well as fractional part, in fixed point format in the cipher domain. The division operation is accomplished via 'multiplication by the reciprocal of the divisor.' The accuracy can be increased to any desirable number of decimal places by increasing the ciphertext size. The novelty of HSDA is, the representation of data in Signed Finite Field format which enables homomorphic operations to work correctly for both positive and negative decimal numbers having integer and fractional part. Simulation results show that the computational cost of homomorphic multiplication algorithm developed in HSDA is substantially lower compared to other existing methods. Since homomorphic division and multiplications are realised, averages, MSE's, Euclidean distances, and matrix norms, can be easily calculated. These complex computations can be delegated to Cloud Servers without revealing the true data. In future, the research work can be extended for fast and efficient Homomorphic operations using high speed Application Specific Integrated Circuits. In WSN, these ASIC's can be fine tuned for the given application to achieve higher speed with minimum power consumption.

REFERENCES

[1] L. Krishnamachari, D. Estrin, and S. Wicker, "The impact of data aggregation in wireless sensor networks," *Proceedings 22nd International Conference on Distributed Computing Systems Workshops*, 2002, pp. 575-578, doi: 10.1109/ICDCSW.2002.1030829.

[2] S. Randhawa and S. Jain, "Data Aggregation in Wireless Sensor Networks: Previous Research, Current Status and Future Directions," *Wireless Pers Commun*, vol. 97, pp. 3355-3425, 2017, doi: 10.1007/s11277-017-4674-5.

[3] X. Liu, J. Yu, F. Li, W. Lv, Y. Wang, and X. Cheng, "Data Aggregation in Wireless Sensor Networks: From the Perspective of Security," in *IEEE Internet of Things Journal*, vol. 7, no. 7, pp. 6495-6513, July 2020, doi: 10.1109/JIOT.2019.2957396.

[4] J. Guo, J. Fang, and X. Chen, "Survey on secure data aggregation for wireless sensor networks," *Proceedings of 2011 IEEE International Conference on Service Operations, Logistics and Informatics*, 2011, pp. 138-143, doi: 10.1109/SOLI.2011.5986543.

[5] J. Xu, G. Yang, Z. Chen, and Q. Wang, "A survey on the privacy-preserving data aggregation in wireless sensor networks," in *China Communications*, vol. 12, no. 5, pp. 162-180, May 2015, doi: 10.1109/CC.2015.7112038.

[6] R. B. Romdhane, H. Hammami, M. Hamdi, and T. Kim, "At the cross roads of lattice-based and homomorphic encryption to secure data aggregation in smart grid," 2019 *15th International Wireless Communications & Mobile Computing Conference (IWCMC)*, Tangier, Morocco, 2019, pp. 1067-1072, doi: 10.1109/iwcmc.2019.87663588.

[7] L. Wang, L. Li, J. Li, J. Li, B. B. Gupta, and X. Liu, "Compressive Sensing of Medical Images With Confidentially Homomorphic Aggregations," in *IEEE Internet of Things Journal*, vol. 6, no. 2, pp. 1402-1409, April 2019, doi: 10.1109/JIOT.2018.2844727.

[8] X. Li, D. Chen, C. Li, and L. Wang, "Secure Data Aggregation with Fully Homomorphic Encryption in Large-Scale Wireless Sensor Networks," *Sensors*, vol. 15, no. 7, pp. 15952–15973, 2015, doi: 10.3390/s150715952.

[9] A. El-Yahyaoui and M. Dafir ElkettanI, "Fully Homomorphic Encryption: State of Art and Comparison," *International Journal of Computer Science and Information Security (IJCSIS)*, vol. 14, no. 4, pp. 159-167, April 2016, doi: 10.6084/M9.FIGSHARE.3362338.V1.

[10] A. Acar, H. Aksu, A. Selcuk Uluagac, and M. Conti, "A Survey on Homomorphic Encryption Schemes: Theory and Implementation," *ACM Comput. Surv*, vol. 51, no. 4, pp. 1-35, 2018, doi: 10.1145/3214303.

[11] P. Martins, L.l Sousa, and A. Mariano, "A Survey on Fully Homomorphic Encryption: An Engineering Perspective", *ACM Comput. Surv*., vol. 50, no. 6, pp. 1-33, 2018, doi: 10.1145/3124441.
[12] M. Van Dijk, C. Gentry, S. Halevi, and V. Vaikuntanathan, "Fully Homomorphic Encryption over the Integers. In: Gilbert H. (eds) Advances in Cryptology," *EUROCRYPT, Lecture Notes in Computer Science*, Springer, Berlin, Heidelberg, vol. 6110, pp 24-43, 2010, doi: 10.1007/978-3-642-13190-5_2.
[13] G. Craig, "Fully Homomorphic Encryption Using Ideal Lattices," *Proceedings of the Annual ACM Symposium on Theory of Computing*, vol. 9, pp. 169-178, 2009, doi: 10.1145/1536414.1536440.
[14] C. Jean-Sébastien, A. Mandal, D. Naccache, and M. Tibouchi, "Fully homomorphic encryption over the integers with shorter public keys," in *Advances in Cryptology-CRYPTO, Lecture Notes in Computer Science*, Santa Barbara, CA, USA: Springer, vol. 6841, pp. 487-504, 2011, doi: 10.1007/978-3-642-22792-9_28.
[15] Z. Brakerski and V. Vaikantanathan, "Efficient Fully Homomorphic Encryption from (Standard) LWE," *SIAM Journal on Computing*, vol. 43, no. 2, pp. 831-871, 2014.
[16] Hedglin, Nolan, K. Phillips and A. Reilley, "Building a Fully Homomorphic Encryption Scheme in Python," *Conference Proceedings*, 2019.
[17] J. Ye and M. Shieh, "Low-Complexity VLSI Design of Large Integer Multipliers for Fully Homomorphic Encryption," in *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 26, no. 9, pp. 1727-1736, Sept. 2018, doi: 10.1109/TVLSI.2018.2829539.
[18] S. S. Roy, F. Vercauteren, J. Vliegen, and I. Verbauwhede, "Hardware Assisted Fully Homomorphic Function Evaluation and Encrypted Search," *in IEEE Transactions on Computers*, vol. 66, no. 9, pp. 1562-1572, Sept. 2017, doi: 10.1109/TC.2017.2686385.
[19] Y. Geng, A. Q. Wang, Z. Y. Chen, J. Xu, and H. Y. Wang, "An energy-saving privacy-preserving data aggregation algorithm," *Chinese Journal of Computers*, vol. 34, no. 5, pp. 792-800, 2011, doi: 10.3724/SP.J.1016.2011.00792.
[20] J. Sen, "Secure and Privacy-Preserving Data Aggregation Protocols for Wireless Sensor Networks," *Cryptography and Security in Computing*, vol. 3, pp. 133-164, 2012, doi: 10.5772/38615.
[21] R. Bista, H. Yoo, and J. Chang, "A New Sensitive Data Aggregation Scheme for Protecting Integrity in Wireless Sensor Networks," 2010 *10th IEEE International Conference on Computer and Information Technology*, 2010, pp. 2463-2470, doi: 10.1109/CIT.2010.422.
[22] S. 1. Huang, S. Shieh, and 1 D. Tygar, "Secure encrypted-data aggregation for wireless sensor networks," *Wireless Networks*, vol. 16, no. 4, pp. 915-927, 2010, doi: 10.1109/CIS.2007.207.
[23] J. Jose, M. Princy, and J. Jose, "Integrity Protecting and Privacy Preserving Data Aggregation Protocols in Wireless Sensor Networks: A Survey," *International Journal of Computer Network and Information Security*, vol. 7, pp. 66-74, 2013, doi: 10.5815/ijcnis.2013.07.08.
[24] Lecture 33. "Left and right inverses; pseudoinverse-MIT," Available: https://ocw.mit.edu›courses›positive-definite-matrices-and-applications
[25] R. Mac Ausland, "The moore-penrose inverse and least squares," *Math 420: Advanced Topics in Linear Algebra*, pp. 1-10, 2014.

BIOGRAPHIES OF AUTHORS

Chethana G. received her B. E and MTech degree in Electronics and Communication Engineering from Kuvempu (2001) and VTU university (2007) Karnataka respectively, she is currently working towards the Ph.D. degree with electronics and computer networks, at RV College of Engineering (RVCE), Bengaluru. 560059.INDIA. Her research interests include WSN, Cryptography, Embedded systems.

Dr. Padmaja K. V., received her B. E and MTech degree in Electronics and communication Engineering from Gulbarga and Pondicherry Universities respectively. She joined RV College of Engineering in the year 1991. She has been teaching in the areas of wireless Communication, Computer Comn, Net. Sensors and Measurements nearly 20 years. She has guided 31 UG and 18 PG projects and 4 Ph. D scholars. She has to her credit of 15 Journals, and 20 conference publications. She is a member of IEEE society, IETE and CSI. Currently she is working as Professor and Associate Dean Department of Electronics and Instrumentation Engineering, RVCE, Bengaluru, India.