Utilizing Homomorphic Encryption to Implement Secure and Private Medical Cloud Computing

Ovunc Kocabas, Tolga Soyata
Dept. of Electrical and Computer Engineering, University of Rochester, Rochester, NY 14627
{ovunc.kocabas,tolga.soyata}@rochester.edu

***Abstract*—With a large number of commercially-available non-invasive health monitoring sensors today, remote health monitoring of patients in their homes is becoming widespread. In remote health monitoring, acquired sensory data is transferred into a private or public cloud for storage and processing. While simple encryption techniques can assure data privacy in the case of private clouds, ensuring data privacy becomes a lot more challenging when a public cloud (e.g., Amazon EC2) is used to store and process data. We present an approach that eliminates data privacy concerns in the public cloud scenario, by utilizing an emerging encryption technique called Fully Homomorphic Encryption (FHE). The ability of FHE to allow computations without actually observing the data itself makes it an attractive option for certain medical applications. In this paper, we use cardiac health monitoring for our feasibility assessment and demonstrate the advantages and challenges of our approach by utilizing a well-established FHE library called HElib.**

I. Introduction

Cloud computing can reduce healthcare costs by outsourcing the storage and computation of medical data to cloud operators, however, Personal Health Information (PHI) privacy is strictly mandated by the Health Insurance Portability and Accountability Act (HIPAA) [1]. Signing a Business Associate Agreement (BAA) [2] authorizes cloud storage operators (e.g., CareCloud [3] and DrChrono [4]) to store PHI data. These offerings are all based on *encrypted data storage*, however, there is currently no service that offers secure long-term patient *monitoring*, which would imply *computation* on encrypted data. This paper proposes a novel approach to eliminate privacy concerns. Our proposed Fully Homomorphic Encryption (FHE) based remote patient monitoring solution allows the cloud to perform computations on encrypted data, without actually observing the data (i.e., patient private health information). While this method holds the promise to completely eliminate the cloud-based privacy concerns, it comes at a steep price: FHE-based operations are orders of magnitude slower than regular operations, rendering FHE impractical for generic applications [5]–[7].

Contributions of this paper are: 1) implementation of a well-known ECG algorithm using an open source FHE library [8], 2) detailed description of the steps required for such an implementation, which are far from trivial, 3) presentation of a proof-of-concept study on a restricted set of computations for long-term patient health monitoring, 4) demonstration of the potential for FHE-based generalized secure medical cloud computing. Our claims are proven on test data taken from the University of Rochester THEW ECG database [9], and it is shown that such operations can be performed homomorphically, thereby guaranteeing information security. Given that cardiac diseases are the #1 cause for deaths in the United States [10], our study is an important step in the development of generalized secure medical cloud computing.

This paper is organized as follows: Section II provides background information on FHE, followed by a system- and application-level introduction to our proposed solution. A description of the nature of the acquired medical data and the operations performed on this data are described in Section III. We detail the FHE scheme used for our application development in Section IV and our circuit-based computational approach for this development in Section V. The details of our implementation are presented in Section VI and its performance evaluation in Section VII. Conclusions and pointers to future research are provided in Section VIII.

II. Medical Cloud Computing Environment

Figure 1 presents the proposed medical application for long-term patient monitoring. The application is partitioned into three distinct phases [6]: Acquisition (Phase I), Storage (Phase II) and Computation (Phase III). In this section, we introduce the details of each phase and propose methods to protect the privacy of medical data in each phase.

A. Background on Fully Homomorphic Encryption (FHE)

HIPAA [1] enforces rules to protect the privacy of medical data. Conventional symmetric-key cryptosystems such as AES can be used to provide *HIPAA-compliant storage* [11]. However, once the data is encrypted with AES, no computation can be performed on the encrypted data without first decrypting it. In this paper, we propose to use Fully Homomorphic Encryption [12], which is a type of encryption that allows computations on encrypted data without decrypting it, thereby eliminating privacy concerns.

B. ECG Acquisition Devices

Currently a variety of wearable devices [13], [14] perform ECG recording of patients remotely, however, they are not capable of performing long-term trend analysis. In the proposed scheme, Phase I (on the left of Figure 1) consists of such devices. We assume these devices will acquire the medical data and transfer them to the cloud in both AES and FHE encrypted format. While AES encryption can be performed without any significant resource usage, FHE encryption is

2159-6190/15 $31.00 © 2015 IEEE
DOI 10.1109/CLOUD.2015.78

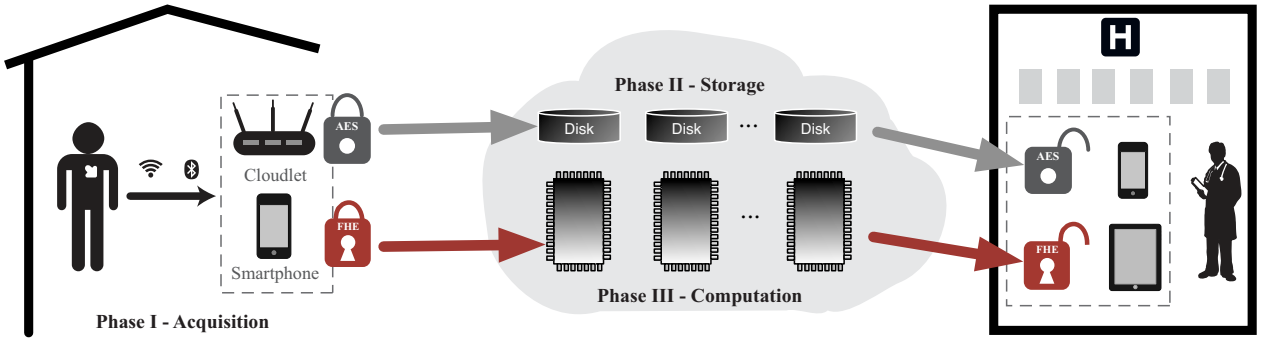

Fig. 1: Proposed Cloud-based secure long-term patient monitoring system.

computationally expensive. Therefore FHE encryption can be done with a nearby devices such as patient's smartphone or a cloudlet [15]–[18]. The communication between acquisition device and nearby device can be secured with conventional encryption schemes [19].

C. Storage and Computation of the Patient Data

The cloud will provide two essential functionalities: storage (Phase II) and computation (Phase III) as shown in Figure 1. The results of the computations will be transmitted to doctor's medical device in FHE-encrypted format, where it can be decrypted only by doctors' private key [20], [21]. We note that medical data will be always kept in encrypted format between acquisition to decryption by the doctor, which enables long-term health monitoring possibilities that didn't exist before.

We propose two alternate paths to transmit medical data to the doctor as shown in Figure 1. In the top path, the medical data is encrypted with AES, which will also serve as permanent storage of the medical data. In the bottom path, the medical data is encrypted with FHE that enables computations on the encrypted medical data and transmits results to the doctor. Top and bottom paths are synchronized so that based on the results of the bottom path, the doctor can request actual raw data from top path for further analysis. This separation could also save storage space, since the FHE-encrypted data is orders of magnitude larger than AES-encrypted data. FHE-encrypted data can be discarded after computations, while AES-encrypted data will be stored permanently.

D. Operations in the Target Medical Application

The proposed medical application will provide two sets of information to the doctor: vital patient health statistics and long-term trend analysis. Our focus will be on the long-term cardiac trend analysis that remains one the most challenging obstacles for developing new drugs and biotechnological products. We choose detecting long QT syndrome (LQTS) as our long-term cardiac analysis which can cause potential deadly cardiac hazards, abbreviated as TdP [22]–[24]. For the vital patient health statistics, we choose average heart rate (HR), minimum and maximum HR as our set of statistics which are also frequently used by doctors to monitor patient's heart activity. We note that more sophisticated computations can be performed with FHE, yet we choose a set of fundamental operations that could serve as a base to build more sophisticated ones [17], [25].

III. Computational Framework

In this section, we will present the computational framework for the proposed medical application. We model the computations introduced in Section II-D as two set of functions: computation and aggregation. We use LQTS detection to explain the details of our framework.

A. Computational Framework for LQTS Detection

LQTS detection can be done using the widely accepted Bazett's formula [26], which is based on the QT and RR intervals in the ECG waveforms (see Figure 2). QT interval represents the ventricular recovery phase of the heart, while RR interval determines the heart rate. Bazett's formula detects LQTS by first computing corrected value of QT (i.e., QT_c), and then comparing QT_c with a clinical threshold (th). Equation 1 presents Bazett's formula with a common clinical threshold of 500 ms, where $QT_c > 500$ is considered as an LQTS hazard.

$$QT_c = \frac{QT}{\sqrt{RR}} \implies \begin{cases} Normal & QT_c \leq 500 \\ LQTS & QT_c > 500 \end{cases} \quad (1)$$

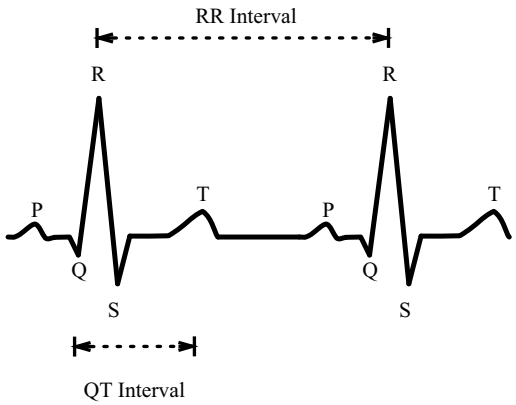

Fig. 2: QT and RR intervals in an ECG

B. Modeling the Data Stream and Functions

Medical data $d[i]$: This is the stream of FHE-encrypted ECG data transmitted from patients' home to the cloud.

Computation functions $f_c(.)$: These are the functions that perform operations on medical data $d[i]$. An example of $f_c(.)$ for the LQTS detection is computing the following equation over λ samples of medical data $d[i]$:

$$d_c[i] = f_c(d[i]) = (d[i] > 500) \mid_{i=1\cdots\lambda} \quad (2)$$

that produces λ Boolean results for each $d[i]$.

Aggregation functions $f_a(.)$: These are the functions that aggregate results computed by $f_c(.)$ and produce a summarized result. An example of $f_a(.)$ for the LQTS detection is detecting if LQTS hazard happened during λ samples of $d[i]$ with Logical OR operation which is presented below:

$$d_a[j] = f_a(d_c[i]) \mid_{j=1\cdots\zeta} = \bigwedge_{j=1\cdots\zeta} \left(\bigwedge_{i=1\cdots\lambda} d_c[i] \right) \quad (3)$$

where $d_a[j]$ are the ζ-aggregated results from the λ-element data stream $d[i]$. Intuitively, $f_a(.)$ computes "the patient is sick if sample 0 says so, OR sample 1 says so OR ..."

IV. FHE Circuit-Based Computational Structure

We will now describe the inner workings of the state-of-the-art FHE scheme called Brakerski-Gentry-Vaikuntanathan (BGV) scheme [27] and its implementation HElib [8].

A. Leveled *FHE scheme*

Gentry's first FHE scheme [12] allowed arbitrary number of additions and multiplications on the encrypted data. In this scheme a small noise is introduced to the encrypted message which affords FHE's security and with each multiplication the noise inside ciphertext increases exponentially. To allow an arbitrary number of operations, a crucial step is the *bootstrapping* method which resets the noise. This step however, incurs severe performance penalties and makes FHE impractical.

Recent FHE schemes proposed different noise management methods such as *modulus-switching* [28] to mitigate performance penalties associated with *bootstrapping*. A variant of BGV is called *leveled* FHE scheme, in which the number of cascaded multiplication operations (termed *multiplicative depth*) can be set beforehand by a variable called level L. With this *leveled* version, a ciphertext starts at level L after encryption and level of the ciphertext decreases by one with each multiplication. Operations can be performed until the ciphertext reaches $L = 1$ and any multiplication after this point will cause a decryption error. The caveat of the *leveled* FHE scheme is the necessity to determine the level L beforehand, as we will do in our proposed computations.

B. Plaintext Space

To allow computations over encrypted medical data, first we need to encode data as messages on plaintexts based on characteristics of BGV. BGV uses polynomial rings in the form of $GF(p^d)$ to represent plaintexts, therefore homomorphic addition ($+_h$) and homomorphic multiplication ($\times_h$) correspond to addition and multiplication of plaintexts in the $GF(p^d)$. Polynomial addition and multiplication differs from the integer arithmetic that will be used in the proposed medical application. This requires special handling of encoding the medical data to plaintexts that allows all possible operations in the proposed medical application. To compute all operations for the medical application we will use the polynomial ring $GF(2)$, where $+_h \leftrightarrow$ XOR and $\times_h \leftrightarrow$ AND. XOR and AND operations form a universal set which allows computing any possible function. Therefore we will convert our functions to binary circuits to evaluate the computations with FHE.

C. Message Packing

BGV uses the techniques proposed in [29] to partition plaintext space into multiple slots such that a ciphertext can encrypt multiple messages at once. These slots are called as *plaintext slots* and allows executing homomorphic operation in parallel similar to Single Instruction Multiple Data (SIMD) fashion. Figure 3 exemplifies a ciphertext $\boldsymbol{X}$, encrypting a plaintext that packs N 4-bit messages into plaintext slots.

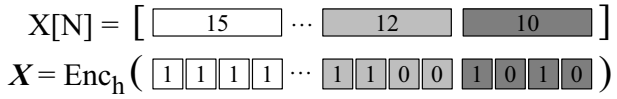

Fig. 3: N 4-bit messages packed into plaintext slots.

For the rest of the paper, we let the lower case $x_7 \cdots x_0$ notation to denote the plaintext slots, and the upper case bold notation $\boldsymbol{X}$ to denote the ciphertext, i.e., the encrypted version of plaintext $X{=}(x_7 \cdots x_0)$. Therefore, $\boldsymbol{X}{=}Enc(X){=}Enc(x_7 \cdots x_0)$, where $Enc()$ is the homomorphic encryption.

D. Primitive Operations in BGV

Message packing allows homomorphic operations to be executed in parallel like SIMD fashion. For example, assume ciphertexts $\boldsymbol{A}$ and $\boldsymbol{B}$ encrypt η plaintext slots. Then $\boldsymbol{A} +_h \boldsymbol{B}$ performs homomorphic addition in parallel to η-slots. Note that result of the homomorphic operations will depend on the selection of plaintext space $GF(p^d)$. We explain four set of BGV primitive operations in $GF(2)$ plaintext space, that will be used extensively during the implementation of the proposed medical application.

Homomorphic Addition ($+_h$): performs slot-wise XOR of the corresponding plaintexts. $+_h$ does not affect the level L of the BGV scheme.

Homomorphic Multiplication ($\times_h$): performs slot-wise AND operation of the corresponding plaintexts. $\times_h$ operation reduces the level L of the ciphertext by one, thereby dominating the level of the BGV scheme.

Rotate ($>>>_h$, $<<<_h$): rotates slots in a barrel shifter fashion that wraps around based on the rotation direction. Rotate could cause garbling of the data which can be corrected using Select operation discussed next.

Select (sel_{mask}): is similar to multiplexer circuit and choses between slots of two ciphertexts. An unencrypted binary vector

is used as a selection mask. For example, if the selection mask entry is 1 for slot-index i, then i^{th} slot from first ciphertext will be selected, otherwise it is selected from second ciphertext. Select operation will be used during our implementation to mask out the bits after rotate operation.

E. Performance Analysis

The performance of FHE implementation with BGV scheme will depend on the level L parameter. Figure 4 shows that the level L affects both ciphertext size and execution times of homomorphic operations. Choosing level L too high both increases the ciphertext size and execution times. However, if level L is too low then desired operations could not be computed. Figure 4 (bottom) also highlights the performance difference between homomorphic operations. For a specific level L, addition is almost *free* while multiplication and rotate operations are computationally expensive.

Therefore, during our implementation we will focus on optimizations that reduce required level L for the proposed application and number of multiplications, and rotations.

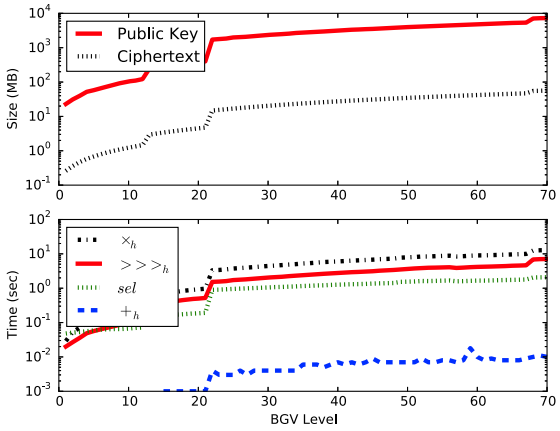

Fig. 4: Effect of BGV Level on performance.

V. FHE Implementation Steps

To utilize BGV efficiently, the entire cloud application must be centered around the computational roadmap shown in Figure 5. The top of this figure (i.e., the *data transformation path*) shows the transformations that the incoming data stream $d[i]$ must go through to produce a properly-formatted ciphertext. This top part is assumed to be performed during data acquisition by a computationally-capable device, such as the patient's smartphone or a cloudlet [15], [30], [31] (see Figure 1). The bottom of Figure 5 (i.e., the *functional transformation path*) will be the focus of this section, which details the steps that must be taken to convert the computation ($f_c(.)$) and aggregation ($f_a(.)$) functions into BGV primitives. These steps include Function-to-Circuit mapping (Section V-A), Circuit-to-SIMD mapping (Section V-B), and Execution using BGV primitives (Section V-C and Section V-D).

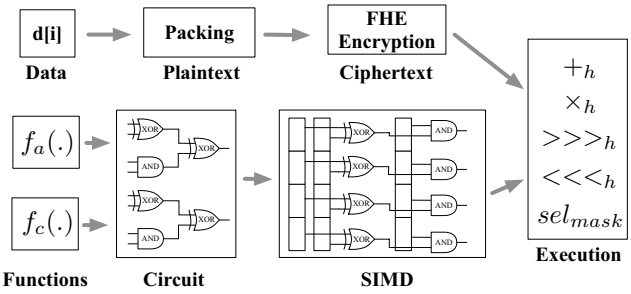

Fig. 5: Roadmap for secure cloud computing with FHE

A. Conversion from Function to Circuit

The first step in functional transformation is the conversion of a function $f(.)$ into a binary circuit. Without loss of generality, this step can be exemplified on a 4-bit greater-than $(X > Y)$ comparator as follows:

$$X > Y = (x_3\overline{y_3} \oplus x_2\overline{y_2}e_3 \oplus x_1\overline{y_1}e_3e_2 \oplus x_0\overline{y_0}e_3e_2e_1) \quad (4)$$

where x_i is the value of bit i of X, $\overline{y_i}$ is the inverse of bit i of Y, and e_i is their bitwise equality $(x_i == y_i)$.

B. Circuit to SIMD Mapping

This step is necessary to execute homomorphic operations in a SIMD fashion. To gain insight into this concept, remember from Section IV-C how each ciphertext encrypts a plaintext that maps the bits of a message into plaintext slots in $GF(2)$. Let $\boldsymbol{X}$ and $\boldsymbol{Y}$ be such ciphertexts that encrypt plaintexts packing k-bit messages X and Y. Further, assume that bit i of message X is mapped to plaintext slot index i (e.g., x_0 is mapped to slot 0) and the plaintext is represented as $(x_{k-1}\ x_{k-2}\ \cdots\ x_1\ x_0)$. The homomorphic addition operation $\boldsymbol{X} +_h \boldsymbol{Y}$ adds (i.e., XORs) each plaintext bit having the *same slot index*, i.e., a single $+_h$ operation on the ciphertext performs bitwise-XOR operations on all plaintext slots in parallel. This has a drawback: No operation can be performed on messages with a *different slot index*, unless proper rotation and selection operations are performed, as detailed in Section IV-D.

To exemplify these trade-offs, let us focus on slot index assignments in Equation 4. Computing terms like $x_0\overline{y_0}$, $x_1\overline{y_1}$, $x_2\overline{y_2}$, $x_3\overline{y_3}$ is equal to performing a single $\times_h$ operation $\boldsymbol{X} \times_h \overline{\boldsymbol{Y}} \iff (x_3\ x_2\ x_1\ x_0) \wedge (\overline{y_3}\ \overline{y_2}\ \overline{y_1}\ \overline{y_0})$ on ciphertexts, where $\iff$ denotes the relationship between the ciphertext and plaintext, and $\wedge$ is slotwise AND. Alternatively, computing terms like $x_2\overline{y_2}e_3$ poses a problem since e_3 is in a different slot index than x_2 and $\overline{y_2}$. So, we need to rotate $\boldsymbol{E}$ right to align it with $\boldsymbol{X} \times_h \overline{\boldsymbol{Y}}$, followed by a selection operation to mask out the bits diffusing from neighboring plaintext slots.

C. Conversion from SIMD to FHE Primitives

To evaluate the circuit in Equation 4 using BGV primitives, we decouple the computation into two separate homomorphic multiplications $(\times_h)$ as follows:

$$(\boldsymbol{X} >_h \boldsymbol{Y}) \ = \ (\boldsymbol{X} \times_h \overline{\boldsymbol{Y}}) \times_h \boldsymbol{M} \iff \quad (5)$$
$$\Big((x_3\ x_2\ x_1\ x_0) \ \wedge \ (\overline{y_3}\ \overline{y_2}\ \overline{y_1}\ \overline{y_0})\Big) \ \wedge \ (1\ e_3\ e_3e_2\ e_3e_2e_1)$$

$$e_i = XNOR(x_i, y_i) = \overline{x_i \oplus y_i} = x_i \oplus y_i \oplus 1 \quad (6)$$

where $\boldsymbol{M}$ and $\boldsymbol{E}$ are the ciphertexts that are the encrypted versions of $(1\ e_3\ e_3e_2\ e_3e_2e_1)$ and $(e_3\ e_2\ e_1\ e_0)$. Calculating $\boldsymbol{M}$ from $\boldsymbol{E}$ requires storing rotated versions of $\boldsymbol{E}$ in temporary ciphertexts that store encrypted values of $(1\ e_3\ e_2\ e_1)$, $(1\ 1\ e_3\ e_2)$ and $(1\ 1\ 1\ e_3)$. Rotation diffuses unwanted bits of $\boldsymbol{E}$ into temporary ciphertexts, which must be replaced with "1"s via a proper selection mask. $\boldsymbol{M}$ is then computed by multiplying these temporary ciphertexts, which reduces the level of $\boldsymbol{M}$ by two. Once $\boldsymbol{M}$ is computed, the final result of $>_h$ is determined by first computing $\overline{\boldsymbol{Y}}$, and then calculating $(\boldsymbol{X} \times_h \overline{\boldsymbol{Y}} \times_h \boldsymbol{M})$. Note that the resulting ciphertext is at level $L-3$ indicating the cost of $>_h$ as 3 levels. In general, comparison of k-bit messages requires $\log_2 k+1$ levels ($\log_2 k$ levels for computing $\boldsymbol{M}$ and 1 level for $\times_h$ at the end).

D. Aggregation of Results

Section V-C detailed the implementation of homomorphic comparison ($>_h$) which detects LQTS based on Equation 2. The result of this comparison is TRUE or FALSE (i.e., LQTS Detected / Not Detected). To detect LQTS in *any sample* within a given interval, a Logical OR aggregation must be performed over multiple comparison results as shown in Equation 3. Logical OR function can be expressed as a depth-1 circuit using XOR, AND gates as $OR(x_i, y_i) = x_i \oplus y_i \oplus x_i y_i$. The aggregated result will be checked for even a single "1" in any slot which means *LQTS Detected*, otherwise it will be interpreted as *Normal*. Aggregation can be performed in binary tree fashion which results in $O(\lceil \log_2 N \rceil)$-depth.

VI. Implementing Medical Applications

In this section, we provide details on the FHE-based implementation of medical applications using the computational structure described in Section IV, and the implementation steps described in Section V. Performance of FHE-based applications depend on two factors: 1) the level L of the FHE scheme, and 2) the number of compute-intensive multiplication and rotation operations. We propose several optimizations to reduce both the level L and the number of expensive FHE operations. We calculate the required level L for each application that operates on N ciphertexts encrypting a vector of k-bit ECG data. Without loss of generality, we specifically focus on three fundamental operations : 1) average heart rate, 2) LQTS detection (Section III-A), and 3) minimum and maximum heart rate calculation.

A. Average Heart Rate

Finding the average heart rate involves accumulating encrypted values in N ciphertexts. For an efficient implementation of FHE-based accumulation, we use conventional VLSI design techniques similar to Wallace [32] and Dadda [33] multipliers that perform high-speed multi-operand additions by reducing both the depth and the number of carry operations. This design approach benefits our FHE-based accumulation in two ways: 1) reducing the number of carry operations avoids compute-intensive $\times_h$ operations, and 2) reducing the depth of computations translates to a reduced L in FHE.

We implement multi-operand additions by using a tree of Carry Save Adders (CSA), which reduces N operands down to 2. Remaining operands are added using a fast parallel-prefix adder with a low-depth carry calculation/propagation to compute the final sum. Both CSA and parallel-prefix adders are amenable to SIMD, which perfectly fits the FHE implementation described in Section V.

Carry Save Adder (CSA): compresses 3 k-bit inputs (X, Y, Z) to 2 outputs (S: sum, C: carry) as follows:

$$\begin{aligned} S &= X \oplus Y \oplus Z \\ C &= (XY \vee XZ \vee YZ) << 1 \end{aligned} \quad (7)$$

where $\oplus$ and $\vee$ are SIMD operations performed on all k bits of the input in parallel. Multiplication depth of the CSA adder is determined by the computation of C which requires a depth-3 circuit (1 for multiplications and 2 for combining the results of multiplications via $\vee$). To reduce depth of computing C, we replace OR operation with XOR which yields an equivalent result in this specific case. This reduces the depth of CSA from 3 to 1. To reduce N k-bit operands, multiple stages of CSAs can be arranged as a tree by connecting S and C as inputs to other CSAs. The number of CSA stages (`nCSAStages`) for reducing N operands is lower-bounded by Equation 8 [34] as shown below. The overall multiplication depth required by the CSA compression is therefore equal to `nCSAStages`.

$$\left\lceil \frac{\log_2(N/2)}{\log_2(3/2)} \right\rceil + 1 \leq \texttt{nCSAStages} \quad (8)$$

Parallel-Prefix Adder: We use the Kogge-Stone adder [35] as our parallel-prefix adder, which has a minimum possible multiplication depth. To add two k-bit numbers, Kogge-Stone adder first computes the initial Generate (G) and Propagate (P), which require a single multiplication depth to compute $G = XY$. Then G and P are updated in $\log_2 k$ stages, where each stage requires a depth-2 multiplication for updating G as $G = G'' \vee G'P'$. The final sum (S) is computed as $S = P \oplus (G << 1)$. Therefore, overall multiplication depth of the Kogge-Stone adder is equal to $2\log_2 k + 1$.

The required level for computing average heart rate of N ciphertexts encrypting k-bit messages is therefore equal to:

$$L > \left(\left\lceil \frac{\log_2(N/2)}{\log_2(3/2)} \right\rceil + 1\right) + (2\log_2 k + 1)$$

B. LQTS Detection

LQTS detection requires evaluating Equation 1 which can be re-written without the square root operation as

$$\frac{QT}{\sqrt{RR}} > 500\ ms \quad \Longrightarrow \quad QT_H > RR_H \quad (9)$$

where $QT_H = QT^2$ and $RR_H = RR \times 250{,}000$ are pre-computed using front-end devices (Figure 1 left), which transmit the FHE-encrypted versions of QT_H and RR_H into the cloud. The cloud can perform LQTS detection as outlined

in Section V, by aggregating the result of the individual comparisons using OR operations, as detailed in Section III-B. To check if an LQTS occurred within a given interval, the doctor's device requests the result from the cloud and decrypts it (Figure 1 right). The doctor's device only needs to check presence of a "1" in the decrypted plaintext. If even a single "1" is present, this indicates that during that interval, QT_H was greater than RR_H at least once, i.e., *LQTS condition detected*.

The required depth for LQTS detection is the comparison depth ($(\log_2 k + 1)$) plus the OR-reduction depth ($\log_2 N$), (individual depths are provided in Section V-C). Therefore, the initial FHE level $L > (\log_2 k + 1 + \log_2 N)$ must be chosen.

C. Minimum & Maximum Heart Rate

Minimum and Maximum Heart Rate computations are based on selecting between the same indexed messages packed inside two ciphertexts. We compute maximum operation as a $f_c(.)$ function applied to two ciphertexts packing a vector of k-bit messages, which is a multiplexer (MUX) circuit as follows:

$$\boldsymbol{R} = (\boldsymbol{X} \times_h \boldsymbol{S}) +_h (\boldsymbol{Y} \times_h \overline{\boldsymbol{S}}) \tag{10}$$

where $\boldsymbol{S}$ is the selector of MUX, computed from the $>_h$ result.

In Section V-C, we showed that comparing two k-bit numbers will produce a k-bit result. If the first number is greater, result will have a single "1" and $(k$–$1)$ "0"s. Otherwise the result will contain k "0"s. Generating $\boldsymbol{S}$ requires diffusing the single "1" of the greater than case to all plaintext slots for the corresponding message. We use a combination of rotate and select operations to route the "1" and add the rotated result to generate k "1"s. Generating $\boldsymbol{S}$ does not involve multiplication. Therefore, the required level is $\log_2 k+1$ (same as $>_h$). Once $\boldsymbol{S}$ is computed, Minimum, Maximum operations require multiplications in Equation 10 which adds one more level, totaling $\log_2 k + 2$. The Minimum operation requires an additional step: inverting $\boldsymbol{S}$, which can be formulated as $\overline{\boldsymbol{S}} = \boldsymbol{S} +_h \boldsymbol{1}$. Using $\overline{\boldsymbol{S}}$ (i.e., inverted $\boldsymbol{S}$) as the selector in Equation 10 will yield the intended Minimum result.

To find the min/max of N ciphertexts encrypting a vector of k-bit messages, we keep applying min and max $f_c(.)$ using a $\log_2 N$ stage binary tree, which has a multiplication depth of $(\log_2 k+2) \times (\lceil \log_2 N \rceil)$. Therefore, the initial level L for the min/max operation should be $L > (\log_2 k + 2) \times (\lceil \log_2 N \rceil)$.

VII. Performance Evaluation

We provide the implementation results for 1) average heart rate, 2) LQTS detection, and 3) min/max heart rate.

A. Experimental Setup

In Section VI, we showed that the level L depends on the bit-length of each message inside the plaintext (k) and the total number of ciphertexts (N). Table I summarizes the minimum required BGV level L for each operation for a given pair of k and N values. While 16-bit messages ($k=16$) were used for the LQTS detection and the min/max heart rate computations, 32-bit messages ($k = 32$) were used for the average heart rate computation to provide sufficient space for up to 2^{16} accumulations of 16-bit individual values.

TABLE I: BGV Level required for each operation.

Operation Type	Required BGV Level L
Average HR	$\left(\left\lceil \frac{\log_2(N/2)}{\log_2(3/2)} \right\rceil + 1\right) + (2\log_2 k + 1)$
LQTS	$\log_2 k + 1 + \lceil \log_2 N \rceil$
Min, Max HR	$(\log_2 k + 2) \times (\lceil \log_2 N \rceil)$

To simulate the acquired ECG samples (Phase I in Figure 1), we used THEW ECG database [9], which contains raw ECG data. A 24-hour time period is processed to extract the heart beat information and can be readily used to simulate our Phase I, where our acquisition devices capture raw ECG data and then pre-process them to extract RR and QT intervals before sending them to cloud in FHE-encrypted format. We encrypted the 87,896 values, each of which is the temporal distance between two heart beats in number-of-samples acquired from Holter monitor during the RR interval (termed toc). Each toc value is encoded as a k-bit *message* (i.e., k=16). We perform operations over encrypted toc values, so our results will be in terms of toc, which can be trivially converted to "time" values by multiplying them with the sampling rate (1000 Hz) at the doctor's phone/tablet after decrypting the final result.

B. Implementation

We use the HElib library [8] for the implementation and run our simulations on a workstation with dual Xeon E5-2695 Processors and 252GB RAM. We report only single-threaded run-time results since HElib is not thread-safe. We set the parameters of the BGV scheme based on the analysis provided in [36] and the optimum level L based on the previously described noise threshold criteria. Table II lists the number of messages that one plaintext can pack at different BGV levels (L). From this table, we can calculate L for performing the three fundamental operations on our 24-hour ECG data, containing 87,896 toc entries, where we encode each toc entry as a "message" (`nMsgs`=$87,896$), using two different message sizes, 16-bit ($k=16$) and 32-bit ($k=32$), depending on the operation. For example, for LQTS detection, we use k=16 on `nMsgs`=$87,896$ messages.

TABLE II: Number of packed messages in a plaintext at various BGV levels for different message bit lengths.

BGV Level (L)	`nSlots`	16-bit messages ($k=16$)	32-bit messages ($k=32$)
$1 \le L < 12$	630	39	19
$12 \le L < 22$	682	42	21
$22 \le L < 68$	1285	80	40
$68 \le L < 77$	1650	103	51
$77 \le L < 100$	2048	128	64

C. Evaluation Criteria

We evaluate the performance of the proposed system based on three relevant billable cloud cost metrics [37]:

Computation Rate (Γ): We define Γ as:

$$\Gamma = \frac{\Gamma_{out}}{\Gamma_{in}} \tag{11}$$

where Γ_{in} is the time interval for the data being transmitted from the patient's house into the cloud, and Γ_{out} is the computation time in the cloud for this data. This "relative" definition allows us to determine whether FHE computations in the cloud can *catch up* with the rate of the incoming data ($\Gamma \leq 1$), or lag behind ($\Gamma > 1$). The significance of the Γ metric is its ability to signal the necessity of additional storage space, and the added computational latency in providing the final result to the doctor. For example, $\Gamma = 2$ implies that, 1 hour patient data takes 2 hours to compute, causing a one hour delay in providing the results to the doctor, and additional storage space to buffer the incoming encrypted data.

Storage Expansion (Λ): FHE significantly expands the storage space required for the encrypted incoming patient data and the FHE public keys. This *storage expansion* becomes worse for increased BGV levels, L. In our definition, $\Lambda = 10,000$ implies that, to store one byte of plain data in encrypted format, 10,000 bytes of cloud storage is required.

Network Throughput (Υ): FHE-based computations strain the network bandwidth, since large amount of encrypted data must be transmitted over WAN connections. We define a third metric, Υ, that determines how much data is being transmitted across the WAN during computations [38], [39]. Some cloud operators (e.g., AWS [37]) only charge for outgoing traffic, not for the incoming traffic. Therefore, we break Υ into two separate parts: $\Upsilon_{patient}$ is the amount of data transferred from front-end devices used by the patient, and Υ_{doctor} is the data transferred from cloud to back-end device used by the doctor.

D. Experimental Results

We present our results in Table III for the aforementioned three fundamental operations over a 24 hours of ECG data, containing 87,896 toc values. Although every row in Table III relates to the same 24-hour ECG data described in detail in Section VII-A, the *partitioning* of the data differs among different rows. From Table III, we see that, Γ=0.36, translating to a computation time of $24 \times 0.36 = 8.64$ hours, or, 31,485 seconds, as indicated in the "Run-time" column. Since $\Gamma < 1$, we deduce that, the FHE computations can be performed faster than their arrival rate, thereby eliminating the necessity to buffer data in the cloud. However, the 24-hour ECG data still takes up $52,700\times$ more space than the original raw data, as indicated in the Λ column. Computing LQTS requires shuttling 8.28GB of encrypted data from the patient's house to the cloud (the $\Upsilon_{patient}$ column) and requires transferring 4.1MB of resulting ciphertexts (the Υ_{doctor} column). The significant disparity is due to the substantial amount of aggregation performed during LQTS detection, leaving only the highly-summarized results that need to be transferred to the doctor's smartphone. This will reduce the application cost since most cloud operators only charge for outgoing data.

The specific row we just described was based on 24 hours of accumulated patient data, acquired, transmitted, and computed as a whole (indicated as "24 hr" in the "ECG Data Interval" column). This row assumes that, the LQTS detection does not start until the entire 24-hour dataset arrives. Alternatively, the very first row of the LQTS detection ("1 min") displays $N = 2$, $L = 7$ and $\Gamma = 0.07$, where the entire 24-hour data is transmitted 1 minute at a time, and the amount of each chunk is significantly smaller: 1 minute chunks require only 2 ciphertexts ($N = 2$), each of which is encoded at an FHE level of L=7. The computation time for each chunk is only 3.9 seconds, corresponding to $\Gamma = \frac{3.9}{60}\frac{s}{s} = 0.07$. This row is computationally-advantageous based on the computation metric (Γ), but it hurts the Υ_{doctor} metric, since the total amount of data to transfer 24 hours of data in 1 minute chunks ends up being 1296.1 MB. The smaller the granularity of the results, the worse the Υ_{doctor} becomes, and the better the Γ is. Required storage for these results improves when the chunk size is smaller due to the reduced level, L, to encrypt these chunks. As a summary, different rows in Table III

TABLE III: Operational cost of medical applications based-on: Computation Rate (Γ), Storage Expansion (Λ) and Network Throughput (Υ).

Operation	ECG Data Interval	N	L	Enc. (sec)	Dec. (sec)	Ctxt (MB)	Pkey (MB)	Run-time (sec)	Γ	Λ	$\Upsilon_{patient}$ (GB)	Υ_{doctor} (MB)
Average	1 min	3	14	0.30	0.24	3.17	274.95	31.2	0.52	41.2K	6.5	4564.8
Heart Rate	60 min	92	23	1.96	0.80	15.49	1783.98	1317.4	0.37	112.2K	16.6	371.7
(k=32)	12 hr	1099	29	2.46	1.05	19.63	2247.22	18803.9	0.44	142.1K	21.1	39.3
	24 hr	2198	31	2.69	1.15	21.03	2414.25	40997.4	0.47	152.3K	22.6	21.1
Long-QT	1 min	2	7	0.08	0.03	0.90	74.32	3.9	0.07	12.6K	1.98	1296.1
Syndrome	60 min	88	13	0.28	0.22	2.96	249.03	1362.4	0.38	38.4K	6.05	71.1
Detection	12 hr	1047	17	0.40	0.28	3.83	323.87	14393.3	0.33	49.7K	7.83	7.7
(k=16)	24 hr	2093	18	0.41	0.31	4.05	352.23	31485.1	0.36	52.7K	8.28	4.1
Min, Max	1 min	2	7	0.08	0.04	0.90	74.32	3.9	0.07	12.5K	1.98	1296.1
Heart Rate	60 min	46	37	3.09	1.27	25.27	2880.37	6946.8	1.93	182.8K	27.12	606.5
(k=16)	12 hr	550	61	4.99	2.05	42.57	4918.28	148189.6	3.43	308.4K	45.69	85.2
	24 hr	1099	67	5.04	2.29	46.94	5362.67	325331.9	3.77	339.7K	50.38	46.9

might provide different cost alternatives for the healthcare organization based on the specific cloud application.

VIII. Conclusions and Future Work

We proposed a novel method for privacy-preserving medical cloud computing using Fully Homomorphic Encryption (FHE). Due to the computational complexity of FHE, we provided a detailed analysis of our approach on three fundamental computations: 1) the average heart rate, 2) min/max heart rate, and 3) automated detection of the long-QT Syndrome (LQTS). We demonstrated our results on an FHE-driven program by using a 24-hour set of ECG samples from the THEW database. Our results show that, a healthcare organization can utilize this program as of today, despite its performance disadvantage. We defined three performance metrics related to the operation of FHE: Γ, Λ, and Υ quantify the computational, storage, and bandwidth requirements of these three fundamental operations. We demonstrated that, the aforementioned three fundamental computations can be performed at the same rate of data arrival, eliminating the need to store excessive amounts of data. Our results show that, these operations can be performed with reasonable cloud resources available today.

Acknowledgments

This work is supported in part by the National Science Foundation grant CNS-1239423 and by Nvidia Corp.

References

[1] US Department of Health and Human Services, "Health Insurance Portability and Accountability Act," http://www.hhs.gov/ocr/privacy/.

[2] US Department of Health and Human Services, "Business Associate Agreement," http://www.hhs.gov/ocr/privacy/hipaa/understanding/coveredentities/contractprov.html.

[3] Care Cloud, ," http://www.carecloud.com/, 2013.

[4] Dr Chrono, ," https://drchrono.com/, 2013.

[5] Michael Naehrig, Kristin Lauter, and Vinod Vaikuntanathan, "Can homomorphic encryption be practical?," in *CCSW*, 2011, pp. 113–124.

[6] Ovunc Kocabas, Tolga Soyata, Jean-Philippe Couderc, Mehmet Aktas, Jean Xia, and Michael Huang, "Assessment of cloud-based health monitoring using homomorphic encryption," in *Proceedings of the 31st IEEE International Conference on Computer Design (ICCD)*, Ashville, VA, USA, Oct 2013, pp. 443–446.

[7] Scott Ames, Muthuramakrishnan Venkitasubramaniam, Alex Page, Ovunc Kocabas, and Tolga Soyata, "Secure health monitoring in the cloud using homomorphic encryption, a branching-program formulation," in *Enabling Real-Time Mobile Cloud Computing through Emerging Technologies*, Tolga Soyata, Ed., chapter 4. IGI Global, 2015.

[8] Shai Halevi and Victor Shoup, ," https://github.com/shaih/HElib.

[9] J Couderc, "The telemetric and holter ecg warehouse initiative (thew): A data repository for the design, implementation and validation of ecg-related technologies," in *EMBC*. IEEE, 2010, pp. 6252–6255.

[10] Donna L Hoyert, Jiaquan Xu, et al., "Deaths: preliminary data for 2011," *National vital statistics reports*, vol. 61, no. 6, pp. 1–51, 2012.

[11] Karen Scarfone, Murugiah Souppaya, and Matt Sexton, "Guide to storage encryption technologies for end user devices," Tech. Rep., NIST, November 2007, NIST Special Publication 800-111.

[12] Craig Gentry, "Fully homomorphic encryption using ideal lattices," 2009, STOC, pp. 169–178.

[13] Alivecor, "ECG screening made easy," http://www.alivecor.com/, 2013.

[14] "Clearbridge VitalSigns CardioLeaf PRO," 2013.

[15] Tolga Soyata, Rajani Muraleedharan, Colin Funai, Minseok Kwon, and Wendi Heinzelman, "Cloud-Vision: Real-Time face recognition using a Mobile-Cloudlet-Cloud acceleration architecture," in *Proceedings of the 17th IEEE Symposium on Computers and Communications (IEEE ISCC 2012)*, Cappadocia, Turkey, Jul 2012, pp. 59–66.

[16] Nathaniel Powers, Alexander Alling, Regina Gyampoh-Vidogah, and Tolga Soyata, "Axaas: Case for acceleration as a service," in *Globecom Workshops (GC Wkshps)*, Austin, TX, Dec 2014, pp. 117–121.

[17] Ovunc Kocabas and Tolga Soyata, "Medical data analytics in the cloud using homomorphic encryption," in *Handbook of Research on Cloud Infrastructures for Big Data Analytics*, P. R. Chelliah and G. Deka, Eds., chapter 19, pp. 471–488. IGI Global, Hershey, PA, USA, Mar 2014.

[18] Haoliang Wang, Wei Liu, and Tolga Soyata, "Accessing big data in the cloud using mobile devices," in *Handbook of Research on Cloud Infrastructures for Big Data Analytics*, P. R. Chelliah and G. Deka, Eds., chapter 18, pp. 444–470. IGI Global, Hershey, PA, USA, Mar 2014.

[19] National Institute of Standards and Technology, "Advanced encryption standard (AES)," Nov. 2001, FIPS-197.

[20] Alex Page, Ovunc Kocabas, Scott Ames, Muthuramakrishnan Venkitasubramaniam, and Tolga Soyata, "Cloud-based secure health monitoring: Optimizing fully-homomorphic encryption for streaming algorithms," in *Globecom Workshops (GC Wkshps)*, Austin, TX, Dec 2014, pp. 48–52.

[21] Alex Page, Ovunc Kocabas, Tolga Soyata, Mehmet Aktas, and Jean-Philippe Couderc, "Cloud-Based Privacy-Preserving Remote ECG Monitoring and Surveillance," *Annals of Noninvasive Electrocardiology (ANEC)*, 2014.

[22] Mehmet K Aktas, Abrar H Shah, and Toshio Akiyama, "Dofetilide-induced long qt and torsades de pointes," *Annals of Noninvasive Electrocardiology*, vol. 12, no. 3, pp. 197–202, 2007.

[23] J.P. Couderc, J. Xia, X. Xu, S. Kaab, M. Hinteeser, and W. Zareba, "Static and dynamic electrocardiographic patterns preceding torsades de pointes in the acquired and congenital long qt syndrome," in *Computing in Cardiology, 2010*, 2010, pp. 357–360.

[24] Rashmi Shah, "Drug-induced qt interval prolongation: regulatory perspectives and drug development," *Annals of medicine*, vol. 36, no. S1, pp. 47–52, 2004.

[25] Ovunc Kocabas and Tolga Soyata, "Towards privacy-preserving medical cloud computing using homomorphic encryption," IGI Global, 2015.

[26] H. C. Bazett, "An analysis of the time-relations of electrocardiograms.," *Annals of Noninvasive Electrocardiology*, vol. 2, no. 2, pp. 177–194, 1997.

[27] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan, "(leveled) fully homomorphic encryption without bootstrapping," in *ITCS*, 2012, pp. 309–325.

[28] Zvika Brakerski and Vinod Vaikuntanathan, "Efficient fully homomorphic encryption from (standard) lwe," in *FOCS*, 2011, pp. 97–106.

[29] Nigel P. Smart and Frederik Vercauteren, "Fully homomorphic SIMD operations," Manuscript at http://eprint.iacr.org/2011/133, 2011.

[30] Tolga Soyata, He Ba, Wendi Heinzelman, Minseok Kwon, and Jiye Shi, "Accelerating mobile cloud computing: A survey," in *Communication Infrastructures for Cloud Computing*, H. T. Mouftah and B. Kantarci, Eds., chapter 8, pp. 175–197. IGI Global, Hershey, PA, USA, Sep 2013.

[31] Tolga Soyata, R. Muraleedharan, S. Ames, J. H. Langdon, C. Funai, M. Kwon, and W. B. Heinzelman, "Combat: mobile cloud-based compute/communications infrastructure for battlefield applications," in *Proceedings of SPIE*, May 2012, vol. 8403, pp. 84030K–84030K.

[32] Christopher S Wallace, "A suggestion for a fast multiplier," *Electronic Computers, IEEE Transactions on*, , no. 1, pp. 14–17, 1964.

[33] Luigi Dadda, "Some schemes for parallel multipliers," *Alta frequenza*, vol. 34, no. 5, pp. 349–356, 1965.

[34] John E. Savage, *Models of Computation: Exploring the Power of Computing*, 1st edition, 1997.

[35] Peter M. Kogge and Harold S. Stone, "A parallel algorithm for the efficient solution of a general class of recurrence equations," *IEEE Trans. Comput.*, vol. 22, no. 8, pp. 786–793, 1973.

[36] Craig Gentry, Shai Halevi, and Nigel P. Smart, "Homomorphic evaluation of the AES circuit," in *CRYPTO*, 2012, pp. 850–867.

[37] "Amazon Web Services," http://aws.amazon.com.

[38] Minseok Kwon, "A tutorial on network latency and its measurements," IGI Global, 2015.

[39] Minseok Kwon, Zuochao Dou, Wendi Heinzelman, Tolga Soyata, He Ba, and Jiye Shi, "Use of network latency profiling and redundancy for cloud server selection," in *Proceedings of the 7th IEEE International Conference on Cloud Computing (IEEE CLOUD 2014)*, Alaska, USA, Jun 2014, pp. 826–832.