Towards Privacy-Preserving Medical Cloud Computing Using Homomorphic Encryption

Övünç Kocabaş
Dept. of Electrical and Computer Engineering, University of Rochester

Tolga Soyata
Dept. of Electrical and Computer Engineering, University of Rochester

ABSTRACT

Personal health monitoring tools, such as commercially available wireless ECG patches, can significantly reduce healthcare costs by allowing patient monitoring outside the healthcare organizations. These tools transmit the acquired medical data into the cloud, which could provide an invaluable diagnosis tool for healthcare professionals. Despite the potential of such systems to revolutionize the medical field, the adoption of medical cloud computing in general has been slow due to the strict privacy regulations on patient health information. We present a novel medical cloud computing approach that eliminates privacy concerns associated with the cloud provider. Our approach capitalizes on Fully Homomorphic Encryption (FHE), which enables computations on private health information without actually observing the underlying data. For a feasibility study, we present a working implementation of a long-term cardiac health monitoring application using a well-established open source FHE library.

Keywords: *Health Information Privacy, Fully Homomorphic Encryption, Remote Health Monitoring.*

INTRODUCTION

The Patient Protection and Affordable Care Act (US Government Printing Office) is one of the most significant government efforts to generalize the use of electronic medical records (EMRs) and to incentivize the development of innovative technologies that can help curb rising US healthcare costs. Cloud computing is a viable option to reduce healthcare costs associated with EMRs by outsourcing the storage of medical data to cloud operators (Amazon Web Services; Google Cloud Platform; Microsoft Windows Azure), however, Personal Health Information (PHI) privacy is strictly mandated by the Health Insurance Portability and Accountability Act (HIPAA) (US Department of Health and Human Services, 2014) and the risks associated with a breach of PHI are steep (up to $1.5M depending on the type of violation). Signing a Business Associate Agreement (BAA) (US-HHS) authorizes cloud storage operators (e.g., (CareCloud, 2013) and (Dr Chrono, 2013)) 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 chapter proposes a novel approach to eliminate privacy concerns. Our proposed Fully Homomorphic Encryption (FHE) based cloud computing 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 (Bos, Lauter, & Naehrig, 2014; Naehrig, Lauter, & Vaikuntanathan, 2011; Kocabas, et al., 2013; Wang, Hu, Chen, Huang, & Sunar, 2013; Dai, Doroz, & Sunar, 2014). In this chapter, one type of computation is shown to be a promising candidate for FHE-based medical applications: long-term patient monitoring.

Contributions of this chapter are: 1) implementation of a well-known ECG algorithm (Couderc, et al., 2011) using an open source FHE library (Halevi & Shoup, 2014), 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 using real data: specifically, the computation of the average heart rate, minimum and maximum heart rate, and the detection of a cardiac hazard called the drug-induced long QT syndrome (LQTS) (Aktas, Shah, & Akiyama, 2007; Brenyo, Huang, & Aktas, 2011), 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 (Couderc J.-P. , 2010), 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 (Hoyert & Xu, 2012), our study is an important and novel step in the development of generalized secure medical cloud computing.

This chapter is organized as follows: We provide 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 the next section and a detailed FHE scheme used for our application development is presented. Our circuit-based computational approach for this development and the details of our implementation are presented. The performance evaluation of our proposed solution details our findings. Conclusions and pointers to future research are provided.

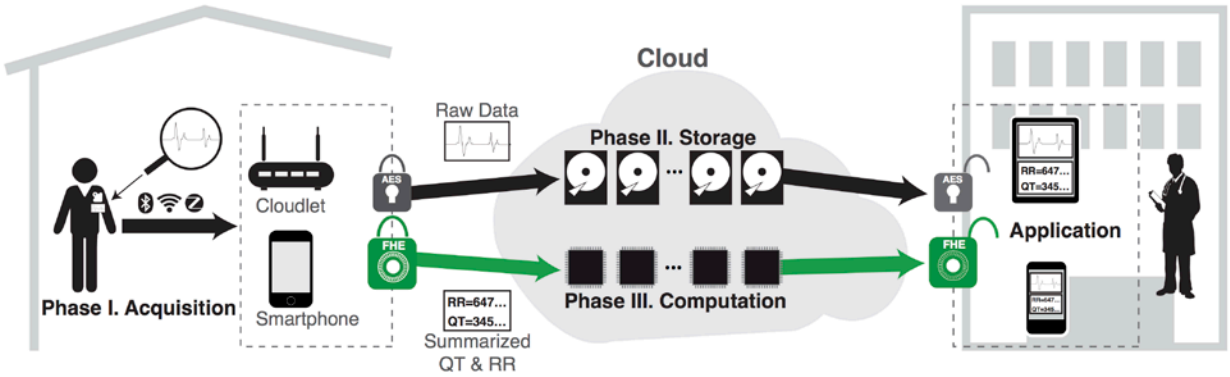

Figure 1: *Proposed Cloud-based secure long-term patient monitoring system. Adapted from* **(Page, Kocabas, Ames, Venkitasubramaniam, & Soyata, 2014)**.

BACKGROUND

Conventional encryption schemes such as AES (NIST-AES, 2001) do not provide a mechanism for computations on encrypted data. When data is encrypted using AES, the only permitted operation on it is *decryption* by using a *secret key*. This implies that AES provides a *secure storage*, but not a *secure computation* mechanism. Cryptographic strength of AES, combined with its ease of its implementation makes AES-based encrypted storage standards the heart of *HIPAA-compliant storage* (Scarfone, Souppaya, & Sexton, 2007), while no current mechanism exists for *HIPAA-compliant computation*. Without this missing component, a system that achieves security in medical data acquisition (Phase I) and storage (Phase II) is possible (Figure 1), but computation (Phase III) is not.

Fully Homomorphic Encryption (FHE) schemes enable the computation of meaningful operations on encrypted data without observing the actual data. Figure 2 illustrates a conceptual example for adding A=23 and B=17 using FHE, where $\boldsymbol{A}$, $\boldsymbol{B}$ are the FHE-encrypted ciphertexts of A and B that are sent to the cloud. Homomorphic Addition (denoted as $+_h$) is performed on $\boldsymbol{A}$ and $\boldsymbol{B}$ to yield the FHE-encrypted result $\boldsymbol{C}$, which can be safely transmitted back out of the cloud, where it is decrypted to obtain the intended computation result C.

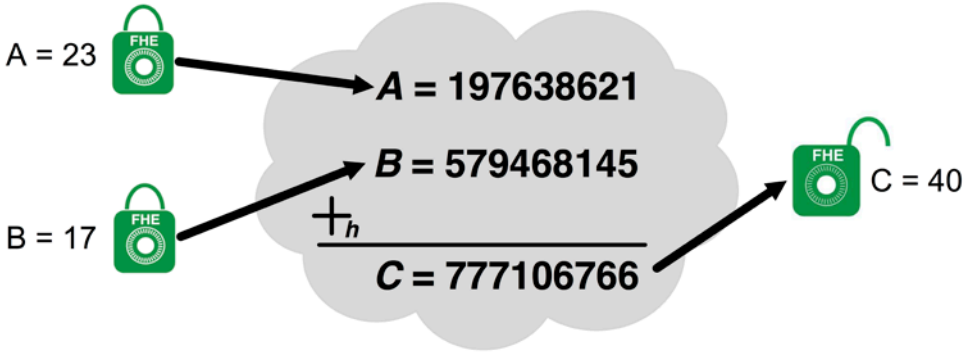

Figure 2: *A conceptual example for Fully Homomorphic Encryption.*

The idea of FHE was first proposed by Rivest et al. in 1978 (Rivest, Adleman, & Dertouzos, 1978) and remained a puzzle to the researchers over the last three decades until Gentry (Gentry, 2009) proposed the first possible mechanism in 2009. Schemes proposed before (Goldwasser & Micali, 1982; El Gamal, 1985; Cohen & Fischer, 1985; Paillier, 1999; Damgard & Jurik, 2001; Boneh, Goh, & Nissim, 2005) could only perform a limited set of operations (i.e., only addition or only multiplication).

Gentry's FHE scheme can perform arbitrary number of additions and multiplications, allowing a set of generic computations on FHE-encrypted data. Gentry's scheme introduces random noise to ciphertexts during encryption. The amount of noise grows with each homomorphic operation and if it exceeds a threshold, decryption produces an incorrect message. Gentry's novel proposal

(Gentry, 2009) uses a method called *bootstrapping* (also known as *recryption*) which resets the noise inside the ciphertexts. However, Gentry's scheme has several inefficiencies related to storage and computation time, which has prevented it from becoming commonplace. Each ciphertext can only encrypt a one-bit message, and to increase the noise threshold, the size of ciphertexts must be large, which results in an expansion of storage space (e.g., the size of a ciphertext encrypting a one-bit message could be multi-million bits). Furthermore, homomorphic operations require a computationally intensive *recryption* operation to periodically reset the noise, making Gentry's FHE scheme impractical (Gentry & Halevi, 2011).

Following Gentry's FHE scheme (Gentry, 2009), several FHE implementations have been proposed to date (Naehrig, Lauter, & Vaikuntanathan, 2011; Dijk, Gentry, Halevi, & Vaikuntanathan, 2010; Brakerski & Vaikuntanathan, 2011; Brakerski & Vaikuntanathan, 2011; Coron, Mandal, Naccache, & Tibouchi, 2011; Gentry & Halevi, 2011; Smart & Vercauteren, 2010) (Stehle & Steinfeld, 2010; Brakerski, Gentry, & Vaikuntanathan, 2012; Gentry, Halevi, & Smart, 2012; Gentry, Halevi, & Smart, 2012). Yet a practical FHE scheme does not exist as of now and improving the performance of FHE remains very active research area (DARPA-PROCEED).

SYSTEM AND APPLICATION MODELING

Figure 2 illustrates our proposed system, which will enable long-term health monitoring of patients securely and automatically. Privacy of the health data is handled in three distinct phases (Kocabas, et al., 2013): Acquisition (Phase I), Storage (Phase II), and Computation (Phase III).

ECG Acquisition Devices

ECG recording technology has advanced to the point where there are personal ECG recording devices that allow patients to record ECG activity at home (e.g., the iPhone attachment from Alivecor (AliveCor, 2014), and the ECG patch from Clearbridge VitalSigns CardioLeaf (CardioLeaf, 2013)). Most of today's acquisition devices provide accurate measurements, but do not support long-term trend analysis, which can play a significant role in disease prevention.

On the left of Figure 1, Phase I (acquisition) is assumed to be done with such devices, where the data is sampled and converted to digital format. Some computational pre-processing can be performed and the acquired (and partially pre-processed) data is transmitted to the cloud after being encrypted using FHE (Brakerski, Gentry, & Vaikuntanathan, 2012). This encryption step is computationally expensive and necessitates the existence of a nearby computationally capable device such as the patient's smartphone or a cloudlet (Soyata, Ba, Heinzelman, Kwon, & Shi, 2013; Soyata, et al., 2012; Soyata, Muraleedharan, Funai, Kwon, & Heinzelman, 2012). The link between the acquisition device and the computationally capable device can be secured with conventional encryption schemes (NIST-AES, 2001; Rivest, Adleman, & Shamir, 1978).

Storage and Computation of the Patient Data

Two functions of the cloud are outlined in Figure 1: storage (Phase II) and computation (Phase III). Cloud-based monitoring results are transmitted to the doctor's mobile device (on the right) in FHE-encrypted format, where they are decrypted only while the doctor is reviewing the results. Keeping the data in encrypted format from its acquisition point (the ECG patch) to its end point (the doctor's smartphone) enables long-term health monitoring options that didn't exist with storage-only encryption, and will be the focus of this chapter.

Notice that in Figure 1, we propose two alternate paths to transmit the data: Top and bottom paths are storage-only (AES-encrypted) and computation only (FHE-encrypted) data paths, respectively and are synchronized with time-stamp markers. The top path redundantly contains all of the information that the bottom part contains, with one significant distinction: The top (storage-only) version of the data is encrypted with a storage-neutral encryption such as AES (NIST-AES, 2001), whereas, the bottom path is encrypted with a computation-only encryption, such as FHE. While the AES version of the data is meant for long-term storage (e.g., 10 years, mandated by the law), the

bottom part is meant for short-term storage, only during the homomorphic computations. This separation is necessary due to the significant bloating of the data up to a million-fold (Wang, Hu, Chen, Huang, & Sunar, 2013; Halevi & Shoup, 2014; Brakerski, Gentry, & Vaikuntanathan, 2012) during homomorphic encryption. This redundancy allows the application to use the bottom path to calculate the time stamp markers for later retrieval from the permanent AES-based storage.

Operations in the Target Medical Application

We have chosen to detect long QT syndrome (LQTS) as our primary medical application because measuring cardiac safety remains one of the most challenging hurdles in the development of new drugs and biotechnological products. The propensity of the drugs to cause potentially fatal arrhythmia, called torsades des pointes (TdP), is a significant public health issue (Woosley, 2001). An estimated 86% of all of the new drugs that are tested in pharmaceutical development show hERG inhibitory activity leading to TdP (Shah R. R., 2005). hERG is a gene that codes a protein subunit of potassium ion channels, and its contribution to the electrical activity of the heart is well known (Fink, Noble, Virag, Varro, & Giles, 2008). Many drugs potentially prolong the heart's ventricular repolarization process (VR), which in some cases trigger TdP, degenerate into ventricular fibrillation, and can cause sudden cardiac death (SCD) (Shah R. , 2004).

In addition to LQTS Detection, we will also incorporate functionality into our application to provide vital patient health statistics (Page, Kocabas, Soyata, Aktas, & Couderc, 2014). These are the average, minimum, and maximum heart rates. Although more sophisticated operations are feasible with FHE, we will restrict our focus on these fundamental operations, which form a base that allows the implementation of a more generalized set of operations.

COMPUTATIONAL/FUNCTIONAL MODELING

An observation of the applications mentioned in the previous section reveals distinct patterns for the incoming data and the type of functions applied to this data. In this section, we will describe a computational framework for the proposed system described in the previous section.

Operations in the Target Medical Application

The input to the LQTS detection is QT and RR intervals (see Figure 3), computed from raw ECG data by using a set of algorithms validated on a large cohort of subjects (Couderc, et al., 2011). The QT interval represents the ventricular recovery phase of the heart and its prolongation (i.e., long QT) is a marker for a potential Torsades des Pointes (a deadly cardiac hazard, abbreviated as TdP) (Aktas, Shah, & Akiyama, 2007; Brenyo, Huang, & Aktas, 2011; Couderc, et al., 2011; Couderc, et al., 2010; Zareba, et al., 1995). The RR interval is the time between QRS complexes, which is used for determining the heart rate.

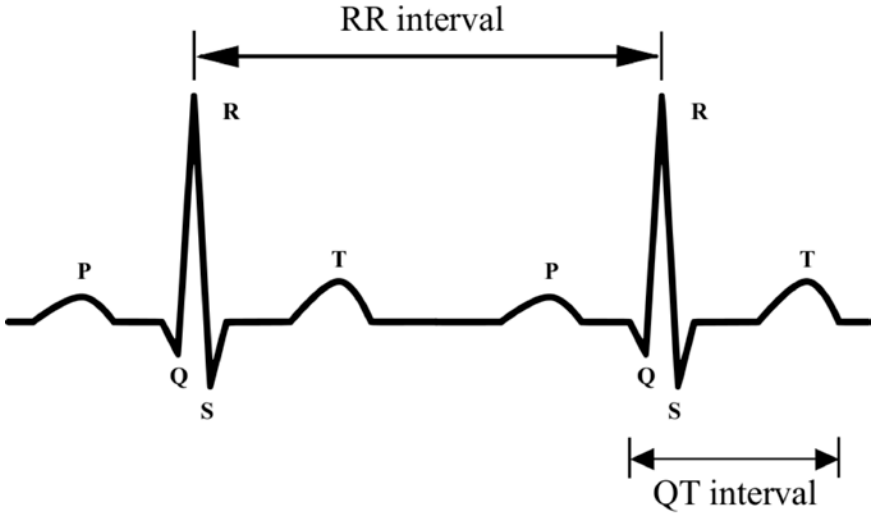

Figure 3: *QT and RR intervals in ECG. (Image based on SinusRhythmLabels.png by Anthony Atkielski).*

QT prolongations are detected by first calculating the QT_C (known as the *corrected QT*) as $QT_c = \frac{QT}{\sqrt{RR}}$ from QT and RR intervals using the widely accepted Bazett's formula (Bazzett, 1997). QT_C values are compared to a clinical threshold (e.g., 500 ms) to detect LQTS as shown below

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

which outlines the criteria to distinguish between normal cardiac operation vs. a potential LQTS hazard. Our case study and preliminary results will be derived by computing Equation 1 on the ECG sample dataset obtained from the THEW repository (Couderc J.-P. , 2010).

Modeling the Data Stream and Operations

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

Computation functions $f_c(.)$: These are the functions that are applied to individual data elements $d[i]$. An example function representing the $f_c(.)$ family is

$$f_c(d[i]) = (d[i] > 500)|_{i=1\ldots\psi} \quad (2)$$

which produces ψ individual Boolean results for each of the data elements, denoting whether each data element $d[i]$ is greater than 500 or not. Note that, this function can be used to detect LQTS as shown in Equation 1.

Aggregation functions $f_a(.)$: These are the functions that aggregate the results obtained by $f_c(.)$ to provide a summarized result. Let $d_c[i]$ be the results calculated by the computation function $f_c(.)$. Then the aggregation result can be described as

$$d_a[j] = f_a(d_c[i])|_{i=1\ldots\psi,\ j=1\ldots\Omega} \quad (3)$$

where $d_a[j]$ are the Ω aggregated results from the ψ element data stream $d[i]$. Note the specific case $\Omega = 1$, which denotes the single result obtained from an entire stream of incoming ψ data elements.

An example application of using data elements $d[i]$ and $f_c(.)$,$f_a(.)$ functions is to detect LQTS within a time interval. More specifically, while $d[i]$ could denote ECG samples 10 ms apart, the aggregated result could be the detection of LQTS hazard within a 20 sec interval (i.e., $\psi = 2000, \Omega = 1$). In this specific example, $f_c(.)$ is chosen to be the comparison function shown in Equation 2 and $f_a(.)$ is chosen to be Logical OR function that aggregates Boolean results generated by the comparison function.

FHE COMPUTATIONAL STRUCTURE

Our proposed system uses one of the most efficient FHE schemes, called the Brakerski-Gentry-Vaikuntanathan (BGV) scheme (Brakerski, Gentry, & Vaikuntanathan, 2012) and its open-source implementation HElib (Halevi & Shoup, 2014). Core of the computations in our implementation resemble Equations 2 and 3 on the incoming FHE-encrypted data stream $d[i]$. Since the operations available in the BGV scheme require an unconventional representation of this data stream $d[i]$ and the evaluation functions $f_c(.)$ and $f_a(.)$, a detailed presentation of the BGV scheme is provided in this section.

Leveled FHE Scheme

Following Gentry (Gentry, 2009), FHE schemes up to date (Dijk, Gentry, Halevi, & Vaikuntanathan, 2010; Brakerski & Vaikuntanathan, 2011; Brakerski & Vaikuntanathan, 2011; Coron, Mandal, Naccache, & Tibouchi, 2011; Gentry & Halevi, 2011; Smart & Vercauteren, 2010; Stehle & Steinfeld, 2010) rely on introducing a small *noise* into a ciphertext during encryption. This noise grows with each operation and can cause decryption errors if it is allowed to exceed a certain threshold. This requires performing the computationally expensive *recrypt* operation to reset the noise after every homomorphic multiplication, which would otherwise increase the noise exponentially.

BGV introduces a *leveled* FHE scheme that avoids the expensive *recrypt* operation. The *leveled* FHE scheme uses a better noise management technique called *modulus-switching* (Brakerski & Vaikuntanathan, 2011), which allows performing cascaded homomorphic multiplications ($\times_h$) without causing decryption errors. A parameter L (the *Level*) is introduced, which must be determined before starting any computation. The level L is predominantly determined by the depth of the $\times_h$operations for the function to be evaluated and for the rest of the chapter we will use *multiplicative depth* and level L interchangeably. Right after encryption, each ciphertext is set to a level L and L is reduced by one after each $\times_h$ until $L=1$, at which point further $\times_h$ operations can cause decryption errors.

Leveled FHE improves traditional FHE performance, but introduces an implementation burden: L must be determined *a priori* (before performing any homomorphic operation). During the implementation of our medical application, we will calculate the multiplicative depth of each computation and set the level L accordingly. For the rest of this chapter, we will use 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 operation.

Plaintext Space

In the BGV scheme, plaintexts are represented as polynomial rings in the $GF(p^d)$ where p is a prime number that defines the range of polynomial coefficients and d is the degree of the polynomials. Homomorphic addition and multiplication of ciphertexts correspond to addition and multiplication of plaintexts in the specified polynomial ring, respectively. We choose the polynomial ring in $GF(2)$ (i.e., $p = 2,\ d = 1$), where homomorphic addition and multiplication of ciphertexts translate to XOR and AND operations on the plaintexts, respectively. This "functionally complete" set (i.e., XOR and AND) will allow the fundamental operations of our medical application to be represented as a binary circuit using a combination of XOR and AND gates in the following sections.

Message Packing

Representing plaintexts as polynomial rings in $GF(p^d)$ allows the "packing" of multiple messages into a plaintext by partitioning it into independent "slots" (Smart & Vercauteren, 2014), thereby permitting the execution of the same homomorphic operation on multiple slots in a Single Instruction Multiple Data (SIMD) fashion. Figure 4 exemplifies a ciphertext $\boldsymbol{X}$, encrypting a plaintext that packs two 4-bit messages (X[0] and X[1]), into 8 plaintext slots.

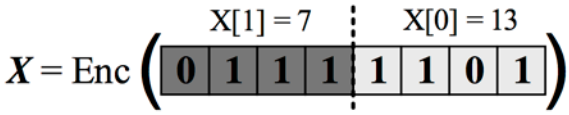

Figure 4: *Two 4-bit messages (X[0], X[1]) packed into 8 plaintext slots*

Equation 4 (left) and ***Table 1*** exemplify the relationship among relevant BGV parameters for message packing. In our implementation, which uses 87,896 ECG samples (i.e., "messages," nMsgs=87, 896) packed into plaintexts, containing 682 slots each (nSlots=682). From Equation 4 (right), 2093 ciphertexts are needed to store these 16-bit ECG samples (i.e., $k = 16,\ N = 2093$).

$$N = \left\lceil \frac{nMsgs}{\left\lfloor \frac{nSlots}{k} \right\rfloor} \right\rceil \Rightarrow \left\lceil \frac{87896}{\left\lfloor \frac{682}{16} \right\rfloor} \right\rceil \Rightarrow N = 2093 \quad (4)$$

Table 1: *Relevant BGV parameters*

Term	BGV Definition	Usage in Application
nSlots	Number of slots in plaintext	-
nMsgs	Number of messages	Number of ECG samples
K	Bit-length of a message	ECG sample bit-length
N	Number of ciphertexts	To store all samples

Primitive Operations in BGV

Packing allows $nSlots$ parallel operations on plaintext slots, but with restricted applicability, as will be detailed in the next section. From a set of existing operations in BGV, we use an orthogonal set of four as shown in Figure 5. Note that, the data contained in a ciphertext (e.g., ciphertext ***A*** in Figure 5a) does not necessarily have a one-on-one correspondence with the data contained in the unencrypted plaintext slots (1010 0101).

Homomorphic Addition ($+_h$): of two ciphertexts corresponds to a slot-wise XOR of the corresponding plaintext in $GF(2)$, as shown in Figure 5a. $+_h$ does not affect the level L of the BGV scheme.

Homomorphic Multiplication ($\times_h$): of two ciphertexts corresponds to a slot-wise AND operation of the corresponding plaintexts, as shown in Figure 5b. $\times_h$ operation adds one to the level L of the ciphertext. Therefore, the depth of multiplications will determine the required level of the BGV scheme.

Rotate ($>>>_h$, $<<<_h$): provides rotation of slots similar to a barrel shifter as shown in Figure 5c. Slots will wrap around based on the rotation direction, thereby potentially garbling the data contained in neighboring slots. This will be corrected using Select operations.

Select (sel_{mask}): chooses between the slots of two plaintexts based on the selection mask as shown in Figure 5d, through an unencrypted binary vector. We will use Select to mask out the bits that are diffused from other messages after a Rotate.

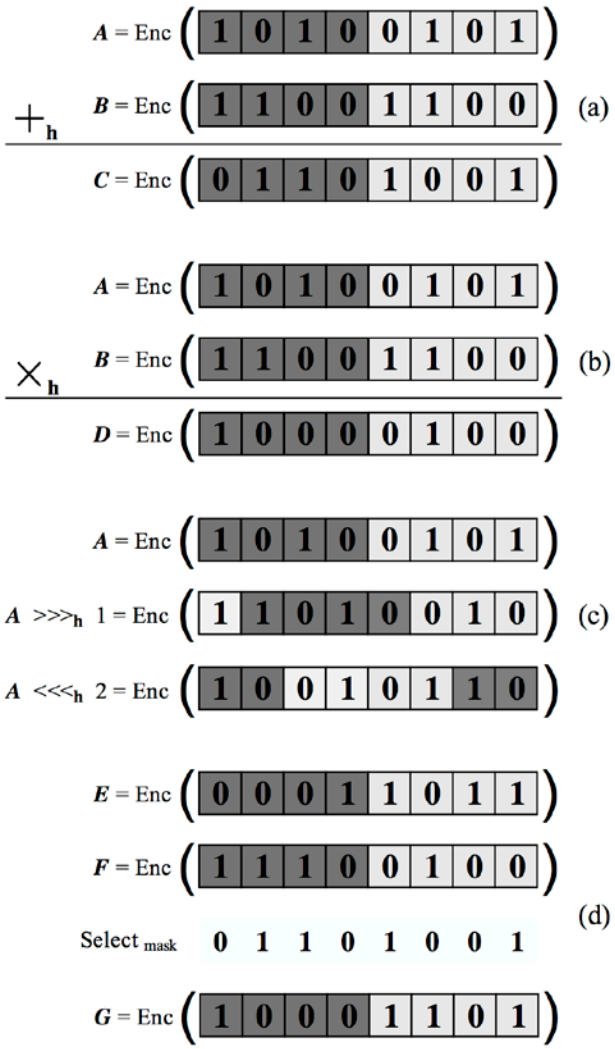

Figure 5: *Computational Primitives in BGV*

Performance Analysis

Leveled FHE implies an untraditional trade-off scheme: choosing L too high slows down the entire chain of homomorphic operations, while small L values prohibit evaluating elaborate functions. Figure 6 shows impact of level L on ciphertext and public key sizes, which grow substantially with the increased L, even to represent the same data. For example, one bit of plaintext might correspond to a 100KB of storage at $L = 10$, but it might grow to 1MB when the level is $L = 20$, requiring 10x more storage space to perform 20 cascaded homomorphic multiplications instead of 10.

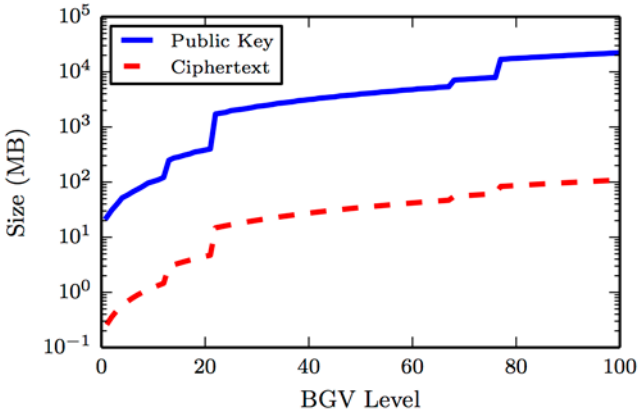

Figure 6: *Public Key and Ciphertext Sizes for different BGV level.*

Figure 7 shows a second disadvantage of increased L: Homomorphic operations execute slower with increased L. For example, while a $\times_h$ operation might take one second at $L = 20$, it takes 10 seconds at $L = 40$. Figure 7 presents the performance of individual FHE operations. While addition operation is almost free, rotation and multiplication operations are expensive and will dominate the execution time. This emphasizes the importance of L in formulating our application. When designing our medical application, optimizing the chain of computations to reduce L, multiplications, and rotations will be our priority.

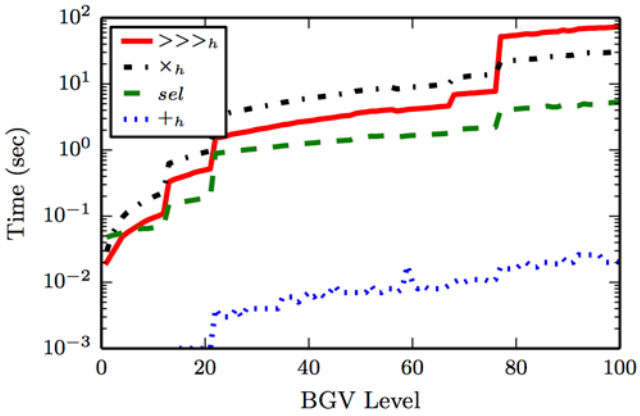

Figure 7: *Level-dependent execution times of BGV primitives.*

FHE IMPLEMENTATION STEPS

To utilize the properties of BGV efficiently, the entire cloud application must be centered around the computational roadmap shown in Figure 8. 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 (Wang, Liu, & Soyata, 2014; Powers, Alling, Gyampoh-Vidogah, & Soyata, 2014) (see Figure 1). The bottom of Figure 8 (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, Circuit-to-SIMD mapping, and Execution using BGV primitives.

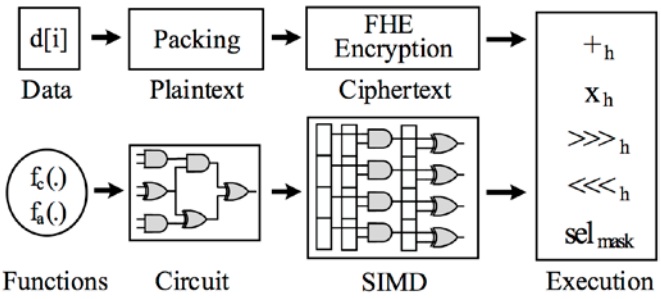

Figure 8: *Roadmap for secure cloud computing FHE*

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 demonstrated on a 4-bit greater-than ($X > Y$) comparator for two numbers X and Y circuit as follows:

$$X > Y = (x_3\bar{y}_3 \oplus x_2\bar{y}_2e_3 \oplus x_1\bar{y}_1e_3e_2 \oplus x_0\bar{y}_0e_3e_2e_1) \quad (5)$$

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

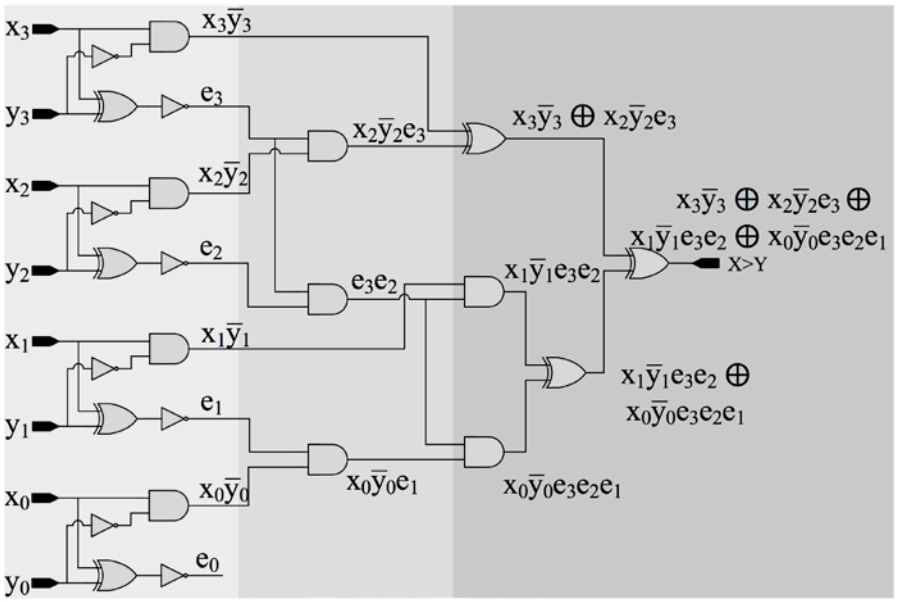

Figure 9: *Depth-3 Comparison Circuit for implementing* $X > Y = (x_3\bar{y}_3 \oplus x_2\bar{y}_2e_3 \oplus x_1\bar{y}_1e_3e_2 \oplus x_0\bar{y}_0e_3e_2e_1)$

The minimum multiplication depth of a 4-bit comparator circuit is 3 as shown in Figure 9. Each multiplication depth is represented by different shades of gray. Note that, the comparator circuit contains only XOR and AND gates, corresponding to homomorphic addition ($+_h$) and multiplication ($\times_h$) in $GF(2)$, respectively. For clarity we depict inverters, which can be implemented by XORs.

Circuit to SIMD Mapping

This step is necessary to execute homomorphic operations in a SIMD fashion. To gain insight into this concept, remember from the previous sections that 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 $(x_{k-1}x_{k-2}\cdots x_1x_0)$. The

homomorphic addition operation $\boldsymbol{X} +_h \boldsymbol{Y}$ adds (i.e., XOR's) each plaintext bit having the *same slot index*. To state alternatively, a single $+_h$ operation on the ciphertext performs bitwise-XOR operations on $nSlots$ 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 previously.

To exemplify these trade-offs, let us focus on the slot index assignments of messages in Equation 5. Computing terms like $x_3\bar{y}_3, x_2\bar{y}_2, x_1\bar{y}_1, x_0\bar{y}_0$ is equal to performing a single $\times_h$ operation on ciphertexts as $\boldsymbol{X} \times_h \boldsymbol{Y} \Leftrightarrow (x_3\ x_2\ x_1\ x_0) \wedge (y_3\ y_2\ y_1\ y_0)$, where $\Leftrightarrow$ denotes the relationship between the ciphertext and plaintext, and $\wedge$ is slotwise AND. Alternatively, computing terms like $x_2\bar{y}_2e_3$ poses a problem since e_3 is in a different slot index than x_2 and $\bar{y}_2$. This means that we need to rotate $\boldsymbol{E}$ right to align it with $\boldsymbol{X} \times_h \boldsymbol{Y}$. After rotating $\boldsymbol{E}$, a selection operation is needed to mask out the bits diffusing from neighboring plaintext slots.

Table 2: *Sequence of FHE operations for 4-bit comparison. $\boldsymbol{X}$, $\boldsymbol{Y}$ are 4-bit messages, $\boldsymbol{x_i}$, $\boldsymbol{y_i}$'s are the bits of the messages at index $\boldsymbol{i}$.*

Step	BGV Operation on Ctxt	Plaintext Slots				Level of Ctxt
		slot 3	slot 2	slot 1	slot 0	
1	$\boldsymbol{E} = \boldsymbol{X}$	x_3	x_2	x_1	x_0	L
2	$\boldsymbol{E} = \boldsymbol{X} +_h \boldsymbol{Y}$	$x_3 \oplus y_3$	$x_2 \oplus y_2$	$x_1 \oplus y_1$	$x_0 \oplus y_0$	L
3	$\boldsymbol{E} = \boldsymbol{X} +_h \boldsymbol{1}$	$e_3 \Leftarrow \overline{x_3 \oplus y_3}$	$e_2 \Leftarrow \overline{x_2 \oplus y_2}$	$e_1 \Leftarrow \overline{x_1 \oplus y_1}$	$e_0 \Leftarrow x_0 \oplus y_0$	L
4	$\boldsymbol{A} = \boldsymbol{E} >>>_h \boldsymbol{1}$	?	e_3	e_2	e_1	L
5	$\boldsymbol{A} = \boldsymbol{E}\ sel_{0111}\ \boldsymbol{1}$	1	e_3	e_2	e_1	L
6	$\boldsymbol{B} = \boldsymbol{E} >>>_h \boldsymbol{2}$	?	?	e_3	e_2	L
7	$\boldsymbol{B} = \boldsymbol{B}\ sel_{0011}\ \boldsymbol{1}$	1	1	e_3	e_2	L
8	$\boldsymbol{C} = \boldsymbol{E} >>>_h \boldsymbol{3}$	?	?	?	e_3	L
9	$\boldsymbol{C} = \boldsymbol{B}\ sel_{0001}\ \boldsymbol{1}$	1	1	1	?	L
10	$\boldsymbol{A} = \boldsymbol{A} \times_h \boldsymbol{B}$	1	e_3	e_3e_2	e_2e_1	$L-1$
11	$\boldsymbol{M} = \boldsymbol{A} \times_h \boldsymbol{C}$	1	e_3	e_3e_2	$e_3e_2e_1$	$L-2$
12	$\boldsymbol{Q} = \boldsymbol{Y}$	y_3	y_2	y_1	y_0	L
13	$\boldsymbol{Q} = \boldsymbol{Q} +_h \boldsymbol{1}$	$\overline{y_3}$	$\overline{y_2}$	$\overline{y_1}$	$\overline{y_0}$	L
14	$\boldsymbol{Q} = \boldsymbol{Q} \times_h \boldsymbol{X}$	$x_3\overline{y_3}$	$x_2\overline{y_2}$	$x_1\overline{y_1}$	$x_0\overline{y_0}$	$L-1$
15	$\boldsymbol{M} = \boldsymbol{M} \times_h \boldsymbol{Q}$	$x_3\overline{y_3}$	$x_2\overline{y_2}e_3$	$x_1\overline{y_1}e_3e_2$	$x_0\overline{y_0}e_3e_2e_1$	$L-3$

Conversion from SIMD to FHE Primitives

To evaluate the circuit in Equation 5 using BGV primitives, we decouple the computation into two separate homomorphic multiplications ($\times_h$) as follows:

$$\begin{gathered}\boldsymbol{X} >_h \boldsymbol{Y} = (\boldsymbol{X} \times_h \overline{\boldsymbol{Y}}) \times_h \boldsymbol{M} \Leftrightarrow \\ \big((x_3\ x_2\ x_1\ x_0) \wedge (\overline{y_3}\ \overline{y_2}\ \overline{y_1}\ \overline{y_0})\big) \wedge (1\ e_3\ e_3e_2\ e_3e_2e_1)\end{gathered} \tag{6}$$

$\boldsymbol{M}$ and $\boldsymbol{E}$ ciphertexts are the encrypted versions of $(1\ e_3\ e_3e_2\ e_3e_2e_1)$ and $(e_3\ e_2\ e_1\ e_0)$. Table 2 lists the steps for evaluating Equation 6 using BGV primitives, by detailing the intermediate ciphertext levels L and the status of the encrypted plaintext slots. First, we compute $\boldsymbol{E}$ in Steps 1-3, which requires an XNOR operation to check if the bits of X and Y are equal as follows:

$$e_i = XNOR(x_i, y_i) = \overline{x_i \oplus y_i} = x_i \oplus y_i \oplus 1 \tag{7}$$

Naive Computation of $\boldsymbol{M}$: Calculating $\boldsymbol{M}$ from $\boldsymbol{E}$ requires storing rotated versions of $\boldsymbol{E}$ within temporary ciphertexts $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}$ which store encrypted values of $(1\ e_3\ e_2\ e_1)$, $(1\ 1\ e_3\ e_2)$ and $(1\ 1\ 1\ e_3)$ in Steps 4-9. Rotation diffuses unwanted bits into $\boldsymbol{E}$ (represented as "?"), which must be replaced with "1"s via a proper selection mask. $\boldsymbol{M}$ (encrypted $(1\ e_3\ e_3e_2\ e_3e_2e_1)$) is computed by multiplying these temporary ciphertexts as $\boldsymbol{M} = \boldsymbol{A} \times_h \boldsymbol{B} \times_h \boldsymbol{C}$ in Steps 10-11. Note that level L of the ciphertexts is reduced by one after each $\times_h$ (as described previously). In general, computing $\boldsymbol{M}$ for k-bit messages requires first generating $k-1$ rotated versions of $\boldsymbol{E}$ and then multiplying them by a $\log_2 k$ depth binary-tree circuit.

Running Products Method: The naive method for computing $\boldsymbol{M}$ requires $O(k)$ of the expensive $\times_h$ and $>>>_h$ operations which dominate the run-time of $>_h$. A close observation of $\boldsymbol{M}$ reveals that plaintext slots store running products of the e_i bits. Therefore, calculation of $\boldsymbol{M}$ can be optimized by computing the running products, as pseudo-coded below:

1: $\boldsymbol{M} \leftarrow \boldsymbol{M}\ >>>_h\ \mathbf{1}$
2: $\boldsymbol{M} \leftarrow \boldsymbol{M}\ sel_{mask}\ \mathbf{1}$, $i \leftarrow 1$
3: **while** $i < k$ **do**
4: $\quad \boldsymbol{T} \leftarrow \boldsymbol{M}$
5: $\quad \boldsymbol{T} \leftarrow \boldsymbol{T}\ >>>_h\ i$
6: $\quad \boldsymbol{T} \leftarrow \boldsymbol{T}\ sel_{mask}\ \mathbf{1}$
7: $\quad \boldsymbol{M} \leftarrow \boldsymbol{M} \times_h \boldsymbol{T}$
8: $i \leftarrow i \cdot 2$
9: **end while**

With the optimization, number of $\times_h$ and $>>>_h$ operations are reduced to $O(\log_2 k)$, allowing a reduced multiplication depth of $\log_2 k$. This results in a speedup of $\approx \frac{O(k)}{O(\log_2 k)}$ compared to the naive method. We will $O(\log_2 k)$ provide detailed results on this later.

Once $\boldsymbol{M}$ is computed, the final result of $>_h$ is determined by first computing $\boldsymbol{Y}$ in Steps 12-13 and then calculating $(\boldsymbol{X} \times_h \bar{\boldsymbol{Y}} \times_h \boldsymbol{M})$ in Steps 14-15. Note that the resulting ciphertext is at level $L-3$ indicating the cost of $>_h$ as 3 levels, which is same as the multiplicative depth of the comparison circuit in Figure 9. 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). Figure 10 shows $>_h$ applied to two 4-bit messages, with a TRUE or FALSE result.

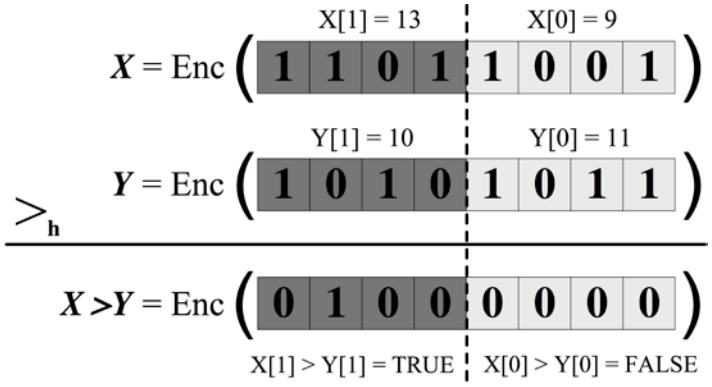

Figure 10: *The result of the comparison in Equation **5** is a **k**-bit integer denoting ZERO (FALSE) or non-zero (TRUE).*

Conversion from SIMD to FHE Primitives

The previous section 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) in the format shown in Figure 10. 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 a have similar format shown in Figure 10, in which even a single "1" in any slot means *LQTS Detected.*

The order of applying $f_a(.)$ affects the depth of the circuit required for aggregation. Since our aggregations are associative, they can be performed in two ways (Savage, 1997): sequential (Figure 11a) or as a binary tree (Figure 11b). While both methods require applying the same number of $f_a(.)$'s for aggregation, binary tree method results in a smaller depth circuit. Specifically, if $f_a(.)$ has a multiplicative depth d, then aggregating N results requires $O(N.d)$-depth using the sequential method, while $O(\lceil \log_2 N \rceil . d)$-depth is sufficient for the binary tree method. Therefore, we will implement aggregation by applying $f_a(.)$ using the binary tree method, which reduces the required level L for BGV.

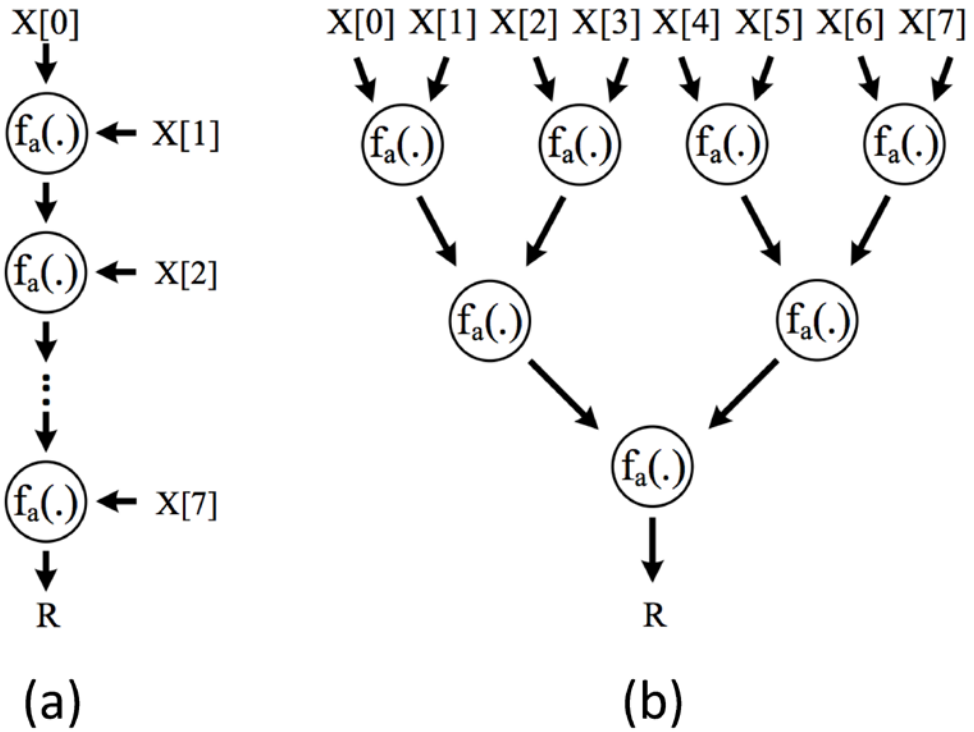

Figure 11: *Aggregation Types: a) Sequential, b)Binary Tree*

IMPLEMENTING MEDICAL APPLICATIONS

In this section, we provide details on the FHE-based implementation of medical applications using the computational structure and the implementation steps described in the previous sections. 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, and 3) minimum and maximum heart rate calculation. These operations will form a fundamental basis for allowing more sophisticated medical applications.

Conversion from SIMD to FHE Primitives

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 (Wallace, 1964) and Dadda (Dadda, 1965) 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 that we described in the previous section. We will now analyze the implementation details of both CSA and parallel-prefix adders:

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

$$\begin{aligned} S &= X \oplus Y \oplus Z \\ C &= (XY \vee XZ \vee XZ) << 1 \end{aligned} \tag{8}$$

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 $\vee$).

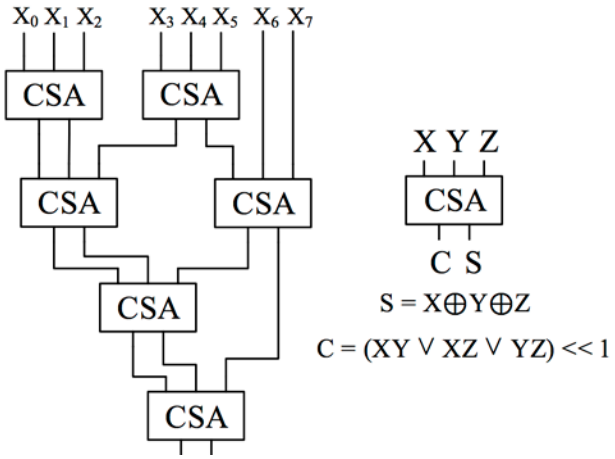

Figure 12: 8:2 compression of the operands using tree of CSAs.

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. Figure 12 exemplifies the reduction of 8 operands down to 2 by using four levels of CSAs. The number of CSA stages ($nCSAStages$) for reducing N operands is lower-bounded by Equation 9 (Savage, 1997) as shown below. The overall multiplication depth required by the CSA compression is equal to $nCSAStages \times 3$

$$\left\lceil \frac{\log_2(N/2)}{\log_2(3/2)} \right\rceil + 1 \leq nCSAStages \tag{9}$$

Further optimizations to CSA: The depth of CSA depends on computing C which requires OR operations over products of inputs. We show in Table 3 that OR operations can be replaced with XORs for computing C, yielding an equivalent result. This wouldn't be a meaningful substitution in a VLSI implementation, since XOR gates typically have a higher delay than OR gates (NCSU, 2014).

However, it reduces the depth of CSA from 3 to 1, which results in a $\approx 3\times$ performance improvement in an FHE implementation. With this optimization, CSA compression requires only $nCSAStages$ depth.

Table 3: *Replacing OR with XOR in CSA.*

X	Y	Z	XY	XZ	YZ	$XY \perp XY \perp XY$	
						$\perp = \vee$	$\perp = \oplus$
0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0
0	1	0	0	0	0	0	0
0	1	1	0	0	1	1	1
1	0	0	0	0	0	0	0
1	0	1	0	1	0	1	1
1	1	0	1	0	0	1	1
1	1	1	1	1	1	1	1

Parallel-Prefix Adder: We use the Kogge-Stone adder (Kogge & Stone, 1973) as our parallel-prefix adder, which has a minimum possible multiplication depth, and its implementation is amenable to SIMD as exemplified in Figure 13.

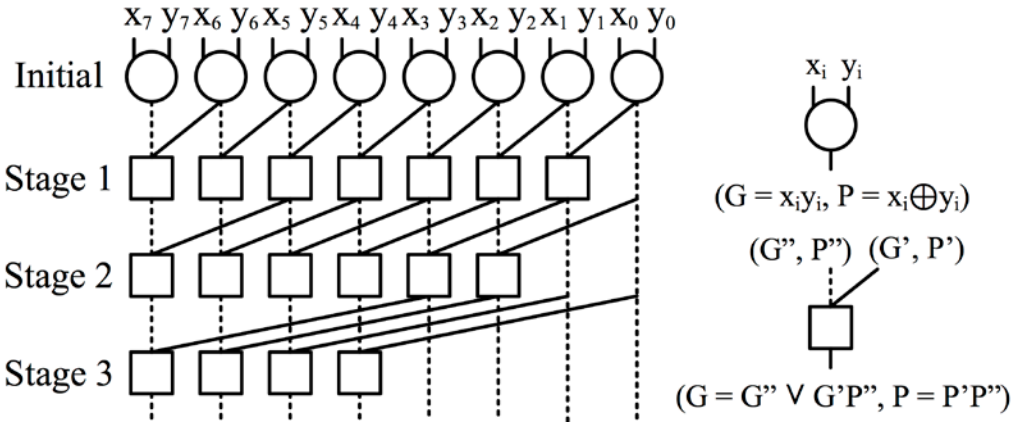

Figure 13: *Kogge-Stone Parallel Prefix Adder*

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 $2\log_2 k + 1$.

To compute average heart rate of N ciphertexts encrypting k-bit messages, first we use tree of CSA adders to compress N ciphertexts down to 2 ciphertexts. Then, we add the remaining 2 ciphertexts using a Kogge-Stone adder. We showed that the multiplication depths for tree of CSA and

Kogge-Stone adder are $\left\lceil \frac{\log_2(N/2)}{\log_2(3/2)} \right\rceil + 1$ and $2\log_2 k + 1$, respectively. Therefore, to compute the average heart rate, the initial level L of BGV should be chosen as $L > \left(\left\lceil \frac{\log_2(N/2)}{\log_2(3/2)} \right\rceil\right) + (2\log_2 k + 1)$.

LQTS Detection

LQTS detection requires evaluating Bazett's formula (Bazzett, 1997) homomorphically as described previously. To avoid the expensive square root and division operations, we re-formulate Equation 1 as follows:

$$\frac{QT}{\sqrt{RR}} > 500\,ms \implies QT^2 > RR \times 250{,}000 \implies QT_H > RR_H \tag{10}$$

where $QT_H = QT^2$ and $RR_H = RR \times 250{,}000$ are pre-computed using front-end devices (left side of Figure 1), which transmit the FHE-encrypted versions of QT_H and RR_H into the cloud for LQTS detection. The cloud can perform LQTS detection as outlined in the previous section, by aggregating the result of the individual comparisons using OR operations, as detailed before. To check if an LQTS occurred within a given interval, the back-end device requests the result from the cloud and decrypts it (right side of Figure 1). The back-end 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 plus the OR-reduction depth, which were shown to have individual depths $(\log_2 k + 1)$ and $\log_2 N$, respectively in the previous section. Therefore, the initial FHE level $L > (\log_2 k + 1 + \log_2 N)$ must be chosen. Figure 14 shows speed-ups obtained with the optimized method for different BGV levels. The parameters are same as that are used in evaluation later in the next section. For each level L, ciphertexts pack 16-bit messages and we observe an average speed-up of $\approx 3\times$ which close to the best-case theoretical improvement ratio of $\approx \frac{O(k)}{O(\log_2 k)}$.

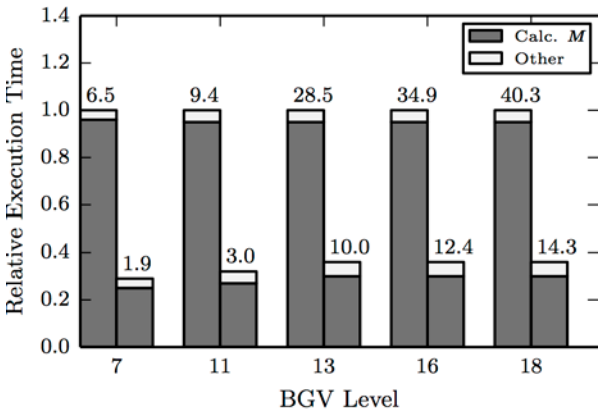

Figure 14: *Normalized run-times (sec) for $>_{\boldsymbol{h}}$ using naive(left bars) and running-products (right bars) method for $\boldsymbol{k}$=16 (16-bit messages).*

Minimum & Maximum Heart Rate

Minimum and Maximum Heart Rate computations are based on selecting between the same indexed messages packed inside two ciphertexts. **Figure 15** presents an example of max $f_c(.)$ function applied to two ciphertexts packing two 4-bit messages.

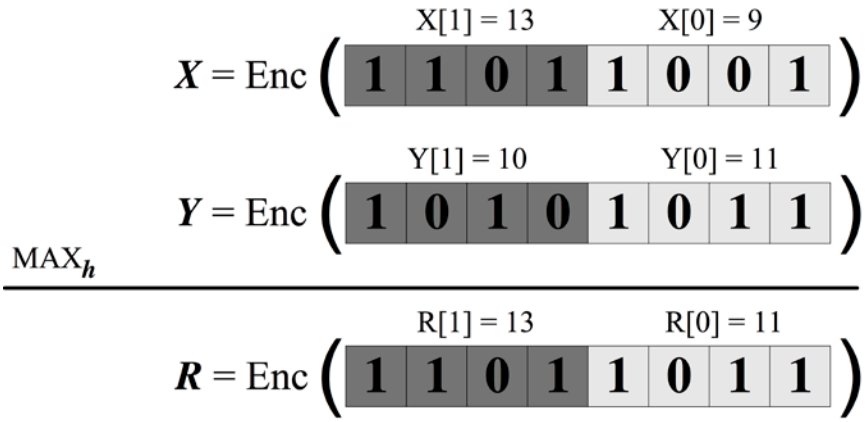

Figure 15: *Maximum Computation*

We model max $f_c(.)$ as a multiplexer circuit as follows:

$$\boldsymbol{R} = (\boldsymbol{X} \times_h \boldsymbol{S}) +_h (\boldsymbol{Y} \times_h \overline{\boldsymbol{S}}) \tag{11}$$

where $\boldsymbol{S}$ acts as the selector of the multiplexer. We use the result of $>_h$ to compute $\boldsymbol{S}$ as demonstrated in Figure 16.

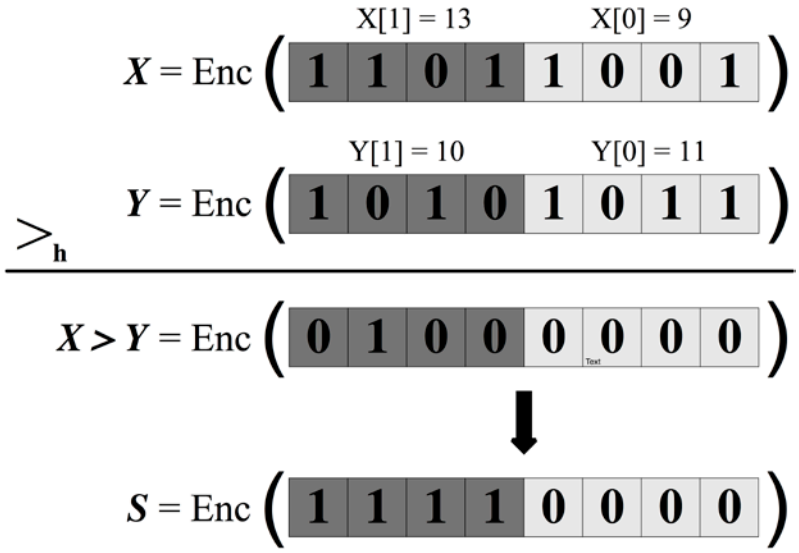

Figure 16: *Generating selector of the multiplexer from comparison.*

Naive Computation of $\boldsymbol{S}$: In the previous section, 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. Pseudo-code for generating $\boldsymbol{S}$ from the comparison result is shown in Figure 17 (left).

Naïve Method

1. $\boldsymbol{C} \leftarrow \boldsymbol{X} >_h \boldsymbol{Y}$
2. $\boldsymbol{S} \leftarrow \boldsymbol{C}$
3. **for** $i = 1$ to k **do**
4. $\quad \boldsymbol{T} \leftarrow \boldsymbol{C} >>>_h i$
5. $\quad \boldsymbol{T} \leftarrow \boldsymbol{T}\ sel_{mask}\ \boldsymbol{0}$
6. $\quad \boldsymbol{S} \leftarrow \boldsymbol{S} +_h \boldsymbol{T}$
7. **end for**
8. **for** $i = 1$ to k **do**
9. $\quad \boldsymbol{T} \leftarrow \boldsymbol{C} <<<_h i$
10. $\quad \boldsymbol{T} \leftarrow \boldsymbol{T}\ sel_{mask}\ \boldsymbol{0}$
11. $\quad \boldsymbol{S} \leftarrow \boldsymbol{S} +_h \boldsymbol{T}$
12. **end for**

Total Sums Method

1. $\boldsymbol{C} \leftarrow \boldsymbol{X} >_h \boldsymbol{Y}$
2. $\boldsymbol{R} \leftarrow \boldsymbol{C}, i \leftarrow 1$
3. **while** $i < k$ **do**
4. $\quad \boldsymbol{T} \leftarrow \boldsymbol{R}$
5. $\quad \boldsymbol{T} \leftarrow \boldsymbol{T} >>>_h i$
6. $\quad \boldsymbol{T} \leftarrow \boldsymbol{T}\ sel_{mask}\ \boldsymbol{0}$
7. $\quad \boldsymbol{R} \leftarrow \boldsymbol{R} +_h \boldsymbol{T}$
8. $\quad i \leftarrow i \cdot 2$
9. **end while**
10. $\boldsymbol{L} \leftarrow \boldsymbol{C}, i \leftarrow 1$
11. **while** $i < k$ **do**
12. $\quad \boldsymbol{T} \leftarrow \boldsymbol{L} <<<_h i$
13. $\quad \boldsymbol{T} \leftarrow \boldsymbol{T}\ sel_{mask}\ \boldsymbol{0}$
14. $\quad \boldsymbol{L} \leftarrow \boldsymbol{L} +_h \boldsymbol{T}$
15. $\quad i \leftarrow i \cdot 2$
16. **end while**
17. $\boldsymbol{S} \leftarrow \boldsymbol{L}$
18. $\boldsymbol{S} \leftarrow \boldsymbol{S} +_h \boldsymbol{R}$

Figure 17: *Pseudo-code for computing S using Naive(left) and Total Sums(right) methods*

Total Sums Method: Computing $\boldsymbol{S}$ with the naive method requires $2k$ rotations, selections and additions. Since we would like to perform a sum operation ($+_h$) on each slot entry, we can use the *Total Sums Algorithm* for vector summation to reduce the number of required rotations to $2\log_2 k$ as shown in Figure 17(right). Speed-ups obtained with this optimized method are plotted in Figure 18. Average speed-up is $\approx 3.37\times$, which is close to the theoretical best-case speed-up of $\approx \frac{O(k)}{O(\log_2 k)}$ compared to the naive method.

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 11, 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 $\boldsymbol{S} = \boldsymbol{S} +_h \boldsymbol{1}$. Using $\boldsymbol{S}$ (i.e., inverted $\boldsymbol{S}$) as the selector in Equation 11 will yield the intended Minimum result.

To find the minimum and maximum of N ciphertexts encrypting a vector of k-bit messages, we keep applying min and max $f_c(.)$ in $\log_2 N$ stage binary tree shown in Figure 11b which has a multiplication depth of $(\log_2 k + 2) \times \log_2 N$. Therefore, the initial level L of BGV should be chosen as $L > (\log_2 k + 2) \times \log_2 N$.

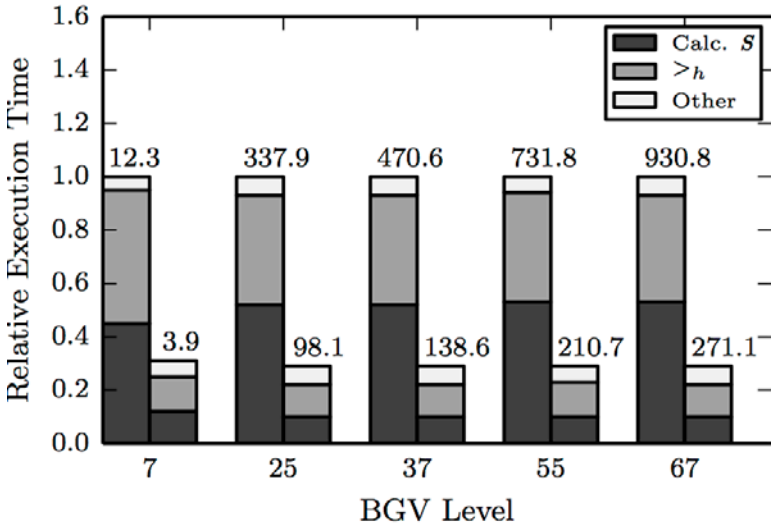

Figure 18: *Normalized run-times (sec) for Maximum using naive(left bars) and Total-Sums (right bars) methods for k=16 (16-bit messages).*

PERFORMANCE EVALUATION

In this section, we will provide the implementation results of the three fundamental operations described in the previous section: 1) average heart rate, 2) LQTS detection, and 3) minimum/maximum heart rate computation.

Experimental Setup

In the previous section, we analyzed the required minimum BGV level (L) for these three fundamental operations, and showed that L depends on two parameters: bit-length of each message inside the plaintext (k) and the total number of ciphertexts (N). Table 4 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 before an overflow occurs (Kocabas & Soyata, 2014).

Table 4: *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 (Couderc J.-P. , 2010), which contains raw ECG data captured from a patient via a 12-lead Holter monitor at a 1000 Hz sampling rate. A 24-hour time period is processed to extract the heart beat information (i.e., the RR interval in Figure 3) in an ISHNE format annotation file (Badilini, 1998) 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 in this annotation file using FHE, 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., $nMsgs = 87896$ and $k = 16$). 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.

Implementation

For implementation, we used the HElib library (Halevi & Shoup, 2014). We ran our simulations on a workstation within the University of Rochester Bluehive cluster (University of Rochester, CIRC), which includes two Intel Xeon E5-2695 Processors and 252GB RAM. Since the HElib library is not thread-safe, we report only single-thread run-time results. We set the parameters of the BGV scheme based on the analysis provided in (Gentry, Halevi, & Smart, 2012). For the fastest execution time, we set the level L of the BGV scheme to the lowest possible value that allows the execution of all homomorphic operations without exceeding the noise threshold.

Table 5: *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)
$1 \leq L < 12$	630	39	19
$12 \leq L < 22$	682	42	21
$22 \leq L < 68$	1285	80	40
$68 \leq L < 77$	1650	103	51
$77 \leq L < 100$	2048	128	64

Table 5 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 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 16-bit message sizes ($k = 16$) on 87,896 messages ($nMsgs = 87{,}896$). Therefore, combining Table 4 and Equation 4, we derive:

$$L > (\log_2 k + 1 + \lceil \log_2 N \rceil) \Longrightarrow L > (5 + \lceil \log_2 N \rceil) \Longrightarrow L > \left(5 + \left\lceil \log_2 \left\lceil \frac{87{,}896}{\left\lceil \frac{nSlots}{16} \right\rceil} \right\rceil \right\rceil\right) \Longrightarrow L = 18 \qquad (12)$$

L value can be iteratively computed from Equation 12 as $L = 18$, which implies packing 42 messages in each ciphertext containing 682 slots. Although $\left(\frac{682}{16}\right) = 42.625$ messages can be packed into 682 slots, partial messages are not utilized in our implementation for simplicity, causing a wasted space of 10 slots in each plaintext. Since each plaintext can hold 42 messages, our 24-hour ECG data requires $N = \left\lceil \frac{87{,}896}{42} \right\rceil = 2093$ ciphertexts.

Evaluation Criteria

We evaluate the performance of the proposed system based on three relevant metrics, each quantifying a different operational cost for cloud outsourcing (Amazon Web Services):

Computation Rate (Γ): We define Γ as:

$$\Gamma = \frac{\Gamma_{out}}{\Gamma_{in}} \qquad (13)$$

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 (which is not a trivial amount, as we will shortly see).

Storage Expansion (Λ): As shown in Figure 6, FHE significantly expands the amount of storage required to store the encrypted incoming patient data as well as 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 (Kwon, et al., 2014). We define a third metric, Υ, which determines how much data is being transmitted across the WAN during computations. Some cloud operators (e.g., AWS (Amazon Web Services)) 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.

Experimental Results

We present our experimental results in Table 6 for the aforementioned three fundamental operations over 24 hours of ECG data, containing 87,896 *toc* values. Although every row in Table 6 relates to the same 24-hour ECG data described in detail in the previous section, the *partitioning* of the data differs among different rows. For example, for the LQTS detection, the last row indicates $N = 2093$, $L = 18$, and $\Gamma = 0.36$. These are the results we derived in Equation 12 at the end of this section. From Table 6, we see that, $\Gamma = 0.36$, 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 done faster than the rate of data arrival, thereby eliminating the necessity to buffer large amounts of 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 is good news in the sense that, cloud operators, such as AWS, only charge for outgoing data, which is orders of magnitude lower in this case.

Table 6: *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)	Ptxt (MB)	Run-time (sec)	Γ	Λ (K)	$\Upsilon_{patient}$ (GB)	Υ_{doctor} (GB)
	1 min	3	14	0.30	0.24	3.17	274.95	31.2	0.52	41.2	6.5	4564.8
	5 min	15	18	0.41	0.31	4.05	352.23	81.6	0.27	52.6	8.3	1166.4
	15 min	44	21	0.45	0.39	4.72	399.12	215.9	0.24	61.3	9.6	453.1
Average	30 min	46	22	1.88	0.77	14.81	1721.55	677.7	0.38	107.4	15.9	710.9
Heart	60 min	92	23	1.96	0.80	15.49	1783.98	1317.4	0.37	112.2	16.6	371.7
Rate	3 hr	275	26	2.21	0.90	17.55	2038.95	4272.7	0.40	127.2	18.8	140.4
(k=32)	6 hr	550	27	2.30	0.94	18.24	2093.75	8679.8	0.40	132.1	19.6	72.9
	12 hr	1099	29	2.46	1.05	19.63	2247.22	18803.9	0.44	142.1	21.1	39.3
	24 hr	2198	31	2.69	1.15	21.03	2414.25	40997.4	0.47	152.3	22.6	21.1
	1 min	2	7	0.08	0.03	0.90	74.32	3.9	0.07	12.6	1.98	1296.1
	5 min	8	9	0.11	0.05	1.12	96.39	21.6	0.07	15.6	2.47	322.6
	15 min	24	11	0.11	0.05	1.34	110.27	99.9	0.11	18.7	2.95	128.6
LQTS	30 min	44	12	0.13	0.06	1.46	121.73	216.9	0.12	18.9	2.98	70.1
	60 min	88	13	0.28	0.22	2.96	249.03	1362.4	0.38	38.4	6.05	71.1
(k=16)	3 hr	262	15	0.33	0.26	3.39	287.73	3215.2	0.30	44.1	6.93	27.2
	6 hr	524	16	0.39	0.27	3.61	308.25	6741.8	0.31	46.9	7.38	14.5
	12 hr	1047	17	0.40	0.28	3.83	323.87	14393.3	0.33	49.7	7.83	7.7
	24 hr	2093	18	0.41	0.31	4.05	352.23	31485.1	0.36	52.7	8.28	4.1
	1 min	2	7	0.08	0.04	0.90	74.32	3.9	0.07	12.6	1.98	1296.1
	5 min	8	19	0.41	0.32	4.27	36.07	198.1	0.66	15.6	2.47	1229.7
Min	15 min	12	25	2.12	0.86	16.86	1984.95	1185.4	1.32	18.7	2.95	1618.6
Max	30 min	23	31	2.58	1.06	21.03	2414.25	2852.8	1.58	18.9	2.98	1009.5
	60 min	46	37	3.09	1.27	25.27	2880.37	6946.8	1.93	38.4	6.05	66.5
HR	3 hr	138	49	4.28	1.73	33.86	3850.37	28459.1	2.64	44.1	6.93	270.9
	6 hr	275	55	4.75	1.88	38.21	4372.2	70849	3.28	46.9	7.38	152.8
(K=16)	12 hr	550	61	4.99	2.05	42.57	4918.28	148189.6	3.43	49.7	7.83	85.2
	24 hr	1099	67	5.04	2.29	46.94	5362.7	325331.9	3.77	52.7	8.28	46.9

Let us now focus on the remaining rows of Table 6. 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 (indicated as "1 min") displays $N = 2, L = 7$ and $\Gamma = 0.07$. This can be interpreted as: When the entire 24-hour data is transmitted 1 minute at a time, 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 $\Gamma = \frac{3.9s}{60s} = 0.07$. Although this row looks computationally-advantageous based on the computation metric (Γ), we see that, 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. To state alternatively, the smaller the granularity of the results, the worse the Υ_{doctor} becomes, and the better the Γ is. The storage required for these results improves when the chunk size is smaller due to the reduced level, L, to encrypt these chunks. For example, while $\Lambda = 12{,}600$ for 1 minute chunks, it increases to $\Lambda = 52{,}700$ for a single large 24-hour chunk. The trade-offs involving these three parameters become less obvious when the cloud operators' billing is based on availability of the resources. For example, if the application has the flexibility to wait for the results longer, different rows in Table 6 might provide different cost alternatives for the healthcare organization, which is the rationale for our detailed analysis.

CONCLUSIONS AND FUTURE WORK

In this chapter, 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 operations: 1) calculation of the average heart rate, 2) calculation of the Minimum and Maximum 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 demonstrated that, the afore- mentioned three fundamental computations could be performed at the same rate of data arrival, eliminating the need to store excessive amounts of data. 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. Our results show that, these operations can be performed with reasonable resources, available within cloud computing platforms today. Due to the continuously improving pricing schemes for cloud services and the algorithmic improvements on FHE, the practicality of FHE-based medical cloud computing will improve, and may eventually become commonplace. Therefore, our study is a good step towards making FHE-based medical cloud computing a reality.

ACKNOWLEDGMENTS

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

REFERENCES

Aktas, M., Shah, A. H., & Akiyama, T. (2007). Dofetilide-Induced Long QT and Torsades de Pointes. *Annals of Noninvasive Electrocardiology, 12*, 197-202.

AliveCor. (2014). *AliveCor Heart Monitor*. From http://www.alivecor.com/home

Amazon Web Services. (n.d.). Retrieved from http://aws.amazon.com

Badilini, F. (1998). The ISHNE Holter Standard Output File Format. *Annals of Noninvasive Cardiology*, 263-266.

Bazzett, H. (1997). An analysis of the time-relations of electrocardiograms. *Annals of Noninvasive Electrocardiology*, 177-194.

Boneh, D., Goh, E.-J., & Nissim, K. (2005). Evaluating 2-DNF Formulas on Ciphertexts. *Proceedings of the 2nd International Conference on Theory of Cryptography*, (pp. 325-341).

Bos, J. W., Lauter, K., & Naehrig, M. (2014). Private predictive analysis on encrypted medical data. *Journal of Biomedical Informatics*.

Brakerski, Z., & Vaikuntanathan, V. (2011). Efficient Fully Homomorphic Encryption from Standard LWE. *Foundations of Computer Science*, 97-106.

Brakerski, Z., & Vaikuntanathan, V. (2011). Fully homomorphic encryption from ring-LWE and security for key dependent messages. *CRYPTO*, (pp. 505-524).

Brakerski, Z., Gentry, C., & Vaikuntanathan, V. (2012). Leveled Fully Homomorphic Encryption without Bootstrapping. *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, (pp. 309-325).

Brenyo, A. J., Huang, D. T., & Aktas, M. (2011). Congenital long and short qt syndromes . *Cardiology, 122*, 237-247.

CardioLeaf. (2013). *Clearbridge VitalSigns CardioLeaf PRO*. From http://www.clearbridgevitalsigns.com/pro.html

CareCloud. (2013). From http://www.carecloud.com/

Cohen, J. D., & Fischer, M. J. (1985). A robust and verifiable cryptographically secure election scheme. *Symposium on Foundations of Computer Science* (pp. 372-382). IEEE.

Coron, J.-S., Mandal, A., Naccache, D., & Tibouchi, M. (2011). Fully homomorphic encryption over the integers with shorter public keys. *CRYPTO*, (pp. 487-504).

Couderc, J., Xia, J., McGrath, M., Zareba, W., Slaton, B., Kakulavaram, A., . . . Gray, D. (2011). Increased repolarization heterogeneity is associated with increased mortality in hemodialysis patients. *Computing in Cardiology*, (pp. 845-848).

Couderc, J., Xia, J., Xu, X., Kaab, S., Hinteeser, M., & Zareba, W. (2010). Static and dynamic electrocardiographic patterns preceding torsades de pointes in the acquired and congenital long QT syndrome. *Computing in Cardiology*, (pp. 357-360).

Couderc, J.-P. (2010). The Telemetric and Holter ECG Warehouse initiative (THEW): A data repository for the design, implementation and validation of ECG related technologies. *IEEE Engineering in Medicine and Biology Society (EMBC)*, 6252-6255.

Couderc, J.-P., Garnett, C., Li, M., Handzel, R., McNitt, S., Xia, X., . . . Zareba, W. (2011). Highly automated QT measurement techniques in 7 thorough QT studies implemented under ICH E14 guidelines. *Annals of Noninvasive Electrocardiology, 16*, 13-24.

Dadda, L. (1965). Some schemes for parallel multipliers. *Alta Frequenza*, 349-356.

Dai, W., Doroz, Y., & Sunar, B. (2014). Accelerating ntru based homomorphic encryption using gpus. *High Performance Extreme Computing Conference.*

Damgard, I., & Jurik, M. (2001). A generalisation, a simplification and some applications of paillier's probabilistic public-key system. *Public Key Cryptography*, (pp. 119-136).

DARPA-PROCEED. (n.d.). From DARPA-PROCEED: http://www.darpa.mil/Our_Work/I2O/Programs/PROgramming_Computation_on_EncryptEd_Data_(PROCEED).aspx

Dijk, M., Gentry, C., Halevi, S., & Vaikuntanathan, V. (2010). Fully Homomorphic Encryption over the Integers. *Advances in Cryptology (EUROCRYPT)*, 24-43.

Dr Chrono. (2013). Retrieved from https://drchrono.com

El Gamal, T. (1985). A Public Key Cryptosystem and a Signature Scheme based on Discrete Logarithms. *IEEE Transactions on Information Theory*, 469-472.

Fink, M., Noble, D., Virag, L., Varro, A., & Giles, W. R. (2008). Contributions of HERG K+ current to repolarization of the human ventricular action potential. *Prog Biophys Mol Biol, 96*, 357-376.

Gentry, C. (2009). *A Fully Homomorphic Encryption Scheme.* Stanford University.

Gentry, C., & Halevi, S. (2011). Fully homomorphic encryption without squashing using depth-3 arithmetic circuits. *FOCS*, (pp. 107-109).

Gentry, C., & Halevi, S. (2011). Implementing Gentry's fully-homomorphic encryption scheme. *EUROCRYPT*, (pp. 129-148).

Gentry, C., Halevi, S., & Smart, N. (2012). Better bootstrapping in fully homomorphic encryption. *PKC*, (pp. 1-16).

Gentry, C., Halevi, S., & Smart, N. (2012). Fully homomorphic encryption with polylog overhead . *EUROCRYPT*, (pp. 465-482).

Gentry, C., Halevi, S., & Smart, N. P. (2012). Homomorphic evaluation of the AES circuit. *CRYPTO*, pp. 850-867.

Goldwasser, S., & Micali, S. (1982). Probabilistic Encryption - How to Play Mental Poker Keeping Secret all Partial Information. *Proceedings of the 14th Annual ACM Symposium on Theory of Computing*, (pp. 365-377).

Google Cloud Platform. (n.d.). Retrieved from https://cloud.google.com

Halevi, S., & Shoup, V. (2014). *HElib Fully Homomorphic Encryption Library*. From http://www.github.com/shaih/HElib

Hoyert, D. L., & Xu, J. (2012). Deaths: preliminary data for 2011. *National vital statistics reports, 61*, 1-51.

Kocabas, O., & Soyata, T. (2014). Medical Data Analytics in the cloud using Homomorphic Encryption. In *Handbook of Research on Cloud Infrastructures for Big Data Analytics* (pp. 471-488). US: IGI Global.

Kocabas, O., Soyata, T., Couderc, J.-P., Aktas, M., Xia, J., & Huang, M. (2013). Assessment of Cloud-based Health Monitoring using Homomorphic Encryption. *Proceedings of the 31th IEEE Conference on Computer Design*, (pp. 443-446). Asheville, NC, USA.

Kogge, P. M., & Stone, H. S. (1973). A parallel algorithm for the efficient solution of a general class of recurrence equations. *Transactions on Computers*, 786-793.

Kwon, M., Dou, Z., Heinzelman, W., Soyata, T., Ba, H., & Shi, J. (2014). Use of Network Latency Profiling and Redundancy for Cloud Server Selection. *Proceedings of the 7th IEEE International Conference on Cloud Computing (IEEE CLOUD 2014)*, (pp. 826-832). Alaska. doi:10.1109/CLOUD.2014.114

Microsoft Windows Azure. (n.d.). Retrieved from http://www.microsoft.com/windowsazure

Naehrig, M., Lauter, K., & Vaikuntanathan, V. (2011). Can homomorphic encryption be practical? *Proceedings of the 3rd ACM workshop on Cloud computing security workshop* . ACM.

NCSU. (2014). *Free PDK 45nm Standard Cell Library* . Retrieved from http://www.eda.ncsu.edu/wiki/FreePDK45

NIST-AES. (2001). *Advanced encryption standard (AES).*

Page, A., Kocabas, O., Ames, S., Venkitasubramaniam, M., & Soyata, T. (2014). Cloud-based Secure Health Monitoring: Optimizing Fully-Homomorphic Encryption for Streaming Algorithms. *IEEE Globecom 2014 Workshop on Cloud Computing Systems, Networks, and Applications (CCSNA).* Austin, TX.

Page, A., Kocabas, O., Soyata, T., Aktas, M., & Couderc, J.-P. (2014). Cloud-Based Privacy-Preserving Remote ECG Monitoring and Surveillance. *Annals of Noninvasive Electrocardiology*.

Paillier, P. (1999). Public Key Cryptosystems Based on Composite Degree Residuosity Classes. *Advances in Cryptology*, (pp. 223-238).

Powers, N., Alling, A., Gyampoh-Vidogah, R., & Soyata, T. (2014). AXaaS: Case for Acceleration as a Service. *IEEE Globecom 2014 Workshop on Cloud Computing Systems, Networks, and Applications.*

Rivest, R. L., Adleman, L., & Dertouzos, M. L. (1978). On data banks and privacy homomorphisms . *Foundations of secure computation*, 169-179.

Rivest, R., Adleman, L., & Shamir, A. (1978). A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM, 21*(2), 120-126.

Savage, J. E. (1997). *Models of Computation: Exploring the Power of Computing.*

Scarfone, K., Souppaya, M., & Sexton, M. (2007). Guide to storage encryption technologies for end user devices. *NIST Special Publication*, 111.

Shah, R. (2004). Drug-induced QT interval prolongation: regulatory perspectives and drug development. *Annals of medicine, 36*, 47-52.

Shah, R. R. (2005). Drug-induced QT interval prolongation regulatory guidance and perspectives on hERG channel studies. *Novartis Foundation Symposium*, (pp. 251-280).

Smart, N. P., & Vercauteren, F. (2010). Fully homomorphic encryption with relatively small key and ciphertext sizes. *PKC*, (pp. 420-443).

Smart, N. P., & Vercauteren, F. (2014). Fully homomorphic {SIMD} operations. *Des. Codes Cryptography, 71*(1), 57-81.

Soyata, T., Ba, H., Heinzelman, W., Kwon, M., & Shi, J. (2013). Accelerating mobile cloud computing: A survey. In *Communication Infrastructures for Cloud Computing* (pp. 175-197). IGI Global.

Soyata, T., Muraleedharan, R., Ames, S., Langdon, J., Funai, C., Kwon, M., & Heinzelman, W. (2012). COMBAT: mobile Cloud-based cOmpute/coMmunications infrastructure for BATtlefield applications. *Proceedings of SPIE*, *8403*, pp. 84030K-84030K. Baltimore, MD.

Soyata, T., Muraleedharan, R., Funai, C., Kwon, M., & Heinzelman, W. (2012, Jul). {Cloud-Vision:} {Real-Time} Face Recognition Using a {Mobile-Cloudlet-Cloud} Acceleration Architecture. *Proceedings of the 17th IEEE Symposium on Computers and Communications (IEEE ISCC 2012)*, (pp. 59-66). Cappadocia, Turkey.

Stehle, D., & Steinfeld, R. (2010). Faster fully homomorphic encryption. *ASIACRYPT*, (pp. 377-394).

University of Rochester, CIRC. (n.d.). *Bluehive Cluster*. Retrieved from http://www.circ.rochester.edu/wiki/ index.php/BlueHive Cluster

US Department of Health and Human Services. (2014). From HIPAA: http://www.hhs.gov/ocr/privacy/

US Government Printing Office. (n.d.). *Patient Protection and Affordable Care Act*. Retrieved from http://www.gpo.gov/fdsys/pkg/BILLS-111hr3590enr/pdf/BILLS-111hr3590enr.pdf

US-HHS. (n.d.). *Business Associate Agreement*. From http://www.hhs.gov/ocr/privacy/hipaa/understanding/coveredentities/contractprov.html

Wallace, C. S. (1964). A suggestion for a fast multiplier . *Transactions on Electronic Computers*, 14-17.

Wang, H., Liu, W., & Soyata, T. (2014). Accessing Big Data in the Cloud Using Mobile Devices. In P. R. Dek, *Handbook of Research on Cloud Infrastructures for Big Data Analytics* (pp. 444-470). Hershey, PA, USA: IGI Global. doi:10.4018/978-1-4666-5864-6.ch018

Wang, W., Hu, Y., Chen, L., Huang, X., & Sunar, B. (2013). Exploring the feasibility of fully homomorphic encryption. *Transactions on Computers*.

Woosley, R. L. (2001, October). *{Drugs That Prolong the QT Interval and/or Induce Torsades de Pointes}.* Tech. rep., {Torsades.org}.

Zareba, W., Moss, A. J., le Cessie, S., Locati, E. H., Robinson, J. L., Hall, W., & Andrews, M. (1995). Risk of cardiac events in family members of patients with long QT syndrome. *Journal of the American College of Cardiology*, 1685-1691.

KEY TERMS AND DEFINITIONS

Cloud Computing: A computational paradigm, in which computational, storage, networking and potentially other resources are rented from a Cloud Service provider to avoid excessive infrastructural investments.

Protected Health Information (PHI): Information that is linked to an individual's medical condition and contains private content that must be protected. This information includes health records, names, addresses, social security numbers to name a few.

Health Insurance Portability and Accountability Act (HIPAA): A set of policies to regulate the protection of the privacy of the medical information (PHI) of individuals.

Healthcare Organization (HCO): Organizations that provide health care that are subject to HIPAA rules. These organizations include not only hospitals, but, a chain of their third party vendors that provide services to the hospitals. A breach in this chain could cause any entity inside the chain to involuntarily violate HIPAA.

Business Associate Agreement (BAA): An agreement signed by third party service providers to ensure the protection of private health information according to HIPAA rules. In other words, any third party vendor of an entity that is subject to HIPAA rules (e.g., hospitals) must have their third party vendors sign a BAA.

Symmetric-key Cryptography: A type of cryptography that uses only a single key for both encrypting and decrypting messages.

Public-key Cryptography: A type of cryptography that requires a key pair: public and private key. While the public key is used for encryption and made available to the public, private key is used for decryption and kept secret for the parties that are privately communicating over a public channel.

Advanced Encryption Standard (AES): A widely used symmetric-key cryptography method based on block ciphers that contain 128-bit message blocks for encryption and decryption.

Homomorphic Encryption (HE): A special class of public-key cryptography that enables computations on ciphertexts without decrypting them. Computations on ciphertexts translate to meaningful operations on the underlying plaintexts.

Additive Homomorphic Encryption: There are two common HE schemes based on the computation that they can perform on ciphertexts: Additive HE and Multiplicative HE. Additive HE schemes can only perform addition of the underlying plaintexts.

Multiplicative Homomorphic Encryption: Multiplicative HE schemes can perform only multiplications of underlying plaintexts.

Fully Homomorphic Encryption (FHE): A homomorphic encryption scheme that can perform addition and multiplication operations on ciphertexts. These operations translate to additions and multiplications of the corresponding plaintexts and enable the computation of any function over the ciphertexts.

Bootstrapping: Method that was proposed by Gentry's first FHE scheme, which allows resetting the noise inside ciphertexts without decrypting them. Bootstrapping is achieved by evaluating decryption

function homomorphically. This method is also known as recryption and it is widely used in current FHE schemes.

Leveled Fully Homomorphic Encryption: A sub-class of fully homomorphic encryption that allows limited number of multiplications on FHE encrypted ciphertexts. The limit of multiplications to be performed is adjusted beforehand and based on multiplication depth, which is also called the *Level*.

Very Large Scale Integration (VLSI): Design/process technique that embeds millions or even billions of transistors into a single integrated circuit (IC) chip. This allows the VLSI IC to perform highly complex computational tasks.

Single Instruction Multiple Data (SIMD): A parallel computation method that relies on executing the same instruction over a set of data. This method is based on parallelism at the data-level.

Electrocardiogram (ECG): Recording of the electrical activity of the heart over a period of time. The recording provides useful information to diagnose possible abnormalities related to the heart.

Holter monitor: A portable monitoring device used by the patients for continuous monitoring of ECG activities outside the hospital while engaging their daily activities.

ECG Patch: A patch consisting of sensors which are attached to a patient to monitor the electrical activities of patients' heart and record them as ECG signals.

RR Interval: The interval between the two R waves of an ECG, which is usually used to compute average heart rate of the individuals.

QT Interval: The interval between Q and T waves of an ECG, which is usually used to diagnose possible problems such as ventricular arrhythmia that can cause sudden deaths.

Long QT syndrome (LQTS): A syndrome that causes prolonged QT intervals. It could be inherited through the genes or acquired later and can cause sudden deaths.

Torsades des Pointes (TdP): A heart condition that can cause sudden deaths. It can be diagnosed by measuring QT intervals.