Cloud-based Secure Health Monitoring: Optimizing Fully-Homomorphic Encryption for Streaming Algorithms

Alex Page*, Ovunc Kocabas*, Scott Ames†, Muthuramakrishnan Venkitasubramaniam †, Tolga Soyata*
*Dept. of Electrical and Computer Engineering
University of Rochester
Rochester, NY 14627
{alex.page, ovunc.kocabas, tolga.soyata}@rochester.edu
†Dept. of Computer Science
University of Rochester
Rochester, NY 14627
{sames,muthuv}@cs.rochester.edu

***Abstract*—There are many incentives for healthcare providers to shift their datacenters to the cloud. However, privacy of patient health information is a major concern when processing medical data off-site. One possible solution is the use of Fully Homomorphic Encryption (FHE), but this solution is too slow for most applications. We present a technique that increases efficiency and parallelism for certain algorithms under FHE. Through simulations, we demonstrate that our method yields about 20x speedup in a sample application. This is a significant step towards practical FHE-based medical remote monitoring.**

I. Introduction

Although a hospital's in-house datacenter is unlikely to match the reliability, efficiency, and scalability benefits of commercial cloud service providers [1]–[3], due to privacy risks and HIPAA regulations, it is often impractical for hospitals make the transition to the cloud. We look for a method to alleviate such privacy concerns, therefore allowing medical centers to take advantage of cloud computing with confidence that their patients' data is safe [4]. Thorough approaches for preventing a data breach in the cloud involve encrypting all data with keys held only by the healthcare provider. These approaches are useful for *storage* [5], [6], but prevent medical applications from *processing* the data. For example, a program may need to look at a patient's ECG recording and report some statistics to their doctor. If conventional encryption was employed, this processing would necessarily expose the data on the system performing the processing.

A novel technique exists which *does* allow encrypted data to be processed securely: *fully homomorphic encryption* (FHE). Unfortunately, current FHE schemes are overwhelming computationally, typically requiring megabytes or gigabytes of space for keys and ciphertexts — even for a relatively small amount of raw data — and requiring several orders of magnitude more CPU instructions per mathematical operation than the equivalent unencrypted operations [7]. However, our initial research with an FHE library called HElib [8] indicates that it may be practical for some basic applications. One such application involves streaming sensor data to the cloud and comparing the values to a threshold. Increasing the complexity of this application, though — i.e., introducing a few more arithmetic operations — quickly makes it unable to complete processing in a reasonable amount of time [9], [10].

HElib is based on the Brakerski-Gentry-Vaikuntanathan (BGV) encryption scheme [11], and requires all algorithms to be modeled in terms of logic gates. The resulting circuits can become quite unwieldy as the complexity of the computer program increases. In order to expand the usefulness of HElib to non-trivial applications, we suggest using an alternate method of computation. Given a boolean algorithm to implement in HElib, we would typically convert it directly to a circuit. (We refer to this as the "naïve" approach.) Instead, we will first transform it into a matrix whose determinant is the algorithm's output (i.e. 0 or 1) by the process described in Section IV. This "matrix" approach will prove to be advantageous in several ways, including efficiency and scalability. Figure 1 provides a system overview for the type of medical application where we intend to apply our matrix technique. Our initial case study involves searching for a specific medical condition in a stream of sensor data, and is described in Section III.

II. Background

Fully Homomorphic Encryption (FHE) is an encryption scheme that allows for data to be stored and processed in an encrypted format. This mechanism gives the cloud provider a method to host and process data without even knowing what the data is. It was not until 2009 that Gentry showed how to construct the first *provably secure* FHE scheme [17]. While the initial work showed how to theoretically achieve such a primitive, it was far from being practical. In the past five years, while there has been tremendous progress in improving the efficiency of FHE schemes [8], [11], [18], there has been no application where it is economically or practically viable to use these schemes. In this paper, we demonstrate the feasibility of achieving secure computation in the cloud for an application where there is a real need to do so.

While our feasibility study will be focused on our particular application, our results should be interpreted more generally as demonstrating the feasibility of securely processing streaming data. *Data streaming* concerns the scenario where input arrives in high volume and only limited space is available to store and process it. Examples include detecting botnets and hackers from the stream of IP addresses that visit a website, or

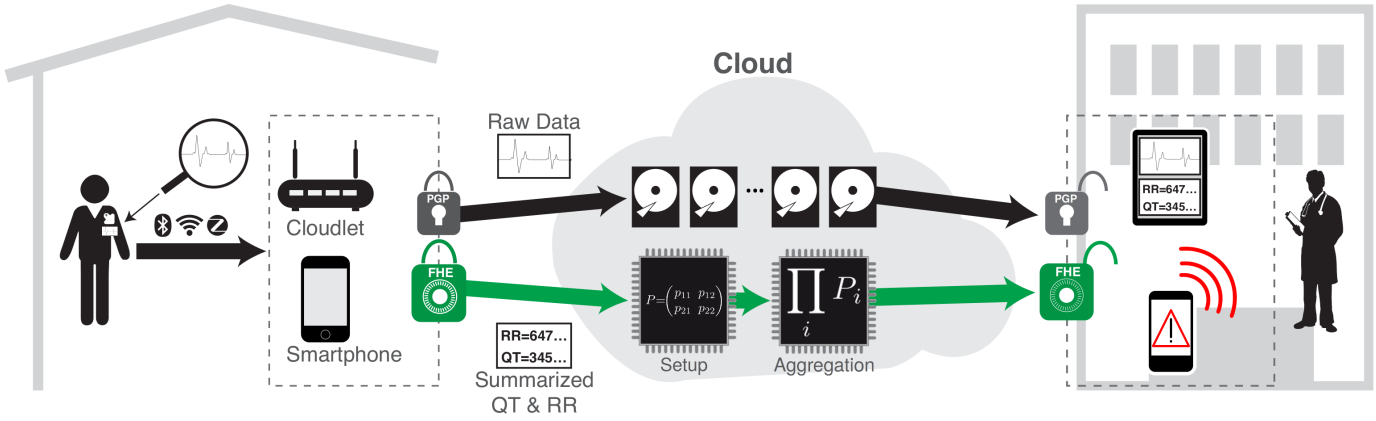

Fig. 1. Overview of an ECG monitoring system. The patch on the patient transmits AES-encrypted ECG data to a nearby Internet-connected device (a smartphone, PC, or a "cloudlet" [12]–[16]. The phone or cloudlet then re-encrypts the data and transmits it to the cloud. The cloud stores the original data for future retrieval, and computes a given function on the FHE-encrypted ECG data. The encrypted result of this function is transmitted to the doctor. Finally, the doctor's phone/tablet/PC decrypts and decodes the result, and alerts the doctor to check the ECG if necessary.

preprocessing massive data sets to collect statistics. The main performance metric of streaming algorithms is the number of passes it makes on the data under memory constraints. Since there are effectively no memory or processing constraints in the cloud, relying on streaming algorithms might look too cautious an approach. However, due to the large overhead of securing the system, these algorithms will provide the right tool to make it feasible.

FHE schemes allow arbitrary computations to be performed on encrypted data. Our approach is to choose a suitable computational model that is reasonably powerful and then rely on existing homomorphic encryption schemes or develop new schemes that allow for both (1) homomorphic computation of all functions in the chosen computational model and (2) aggregation of the results collected over a period of time. Prior to Gentry's work, we already knew how to construct HE schemes that satisfied the first requirement for various (weaker) computational models. The work of Ishai and Paskin [19] and Sander, Young, and Yung [20] also show how to perform NC computations (described in Section IV) over encrypted data. The main challenge is to construct one that will satisfy the second requirement as well.

III. Case Study : Long Term Health Monitoring

There are many medical applications that fit the general form of Figure 1. Since we have advisors in the cardiology department, we chose to implement an ECG-oriented application: detecting prolongation of the QT interval. This application is representative of remote-monitoring scenarios, and our data collection will be simplified by the abundance of hardware sensors (the Clearbridge VitalSigns CardioLeaf [21], for example) in this field. Prolongation of the QT interval (shown in Figure 2) may be genetic (Long QT syndrome, LQTS) or drug-induced. In either case, it increases the risk of torsades de pointes (TdP), an arrhythmia which can lead to serious issues including fibrillation and death [22]. The QT interval (or the "corrected" version, QTc) is therefore an important value for cardiologists to monitor, particularly on patients prescribed with drugs that are known to prolong it. Monitoring is typically conducted in the hospital, but this gives an incomplete picture of the patient's QT interval since they are not participating in their normal daily activities. This makes remote QTc surveillance of an ambulatory patient a good candidate application for our proof-of-concept.

Our goal is to upload live, FHE-encrypted ECG data from a patient to the cloud, and have the cloud server(s) calculate some results and push them to the patient's doctor in relatively real-time. The cloud will be receiving two values, QT and RR, pictured in Figure 2. These values will be generated and homomorphically-encrypted on the patient's end, either on the ECG patch's microprocessor, or more likely, a nearby PC. (At first, rather than using live signals, we use recordings from the THEW database. [23]) The patient's QTc is then computed (homomorphically, in the cloud) as $QT_c = \frac{QT}{\sqrt[3]{RR/sec}}$ (Fridericia's formula [24]). The still-encrypted QTc values then need to be compared to a threshold value, such as 500ms. The function we are interested in, then, is:

$$f(\mathrm{QT}, \mathrm{RR}) = [\mathrm{QT}^3 > (500\mathrm{ms})^3(\mathrm{RR}/\mathrm{sec})], \qquad (1)$$

and this equation will be converted into a matrix by the process described in the next section. Finally, the results must be aggregated over several heartbeats, and transmitted to the doctor.

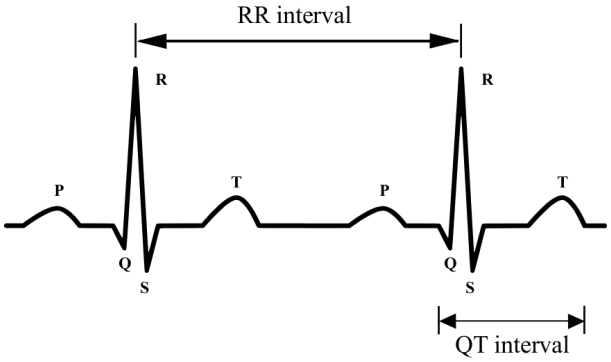

Fig. 2. Normal sinus rhythm. The QT interval represents the time for the ventricular recovery phase of the heart. Prolongation of QT (relative to the RR interval) indicates an increased risk for life-threatening events.

IV. Proposed Solution

In this work we are interested in performing simple computational tasksthat are very useful using FHE schemes. In the context of cloud computing, there are many operations that involve performing the same simple operation on many data points, then somehow aggregating the results and returning them to the client, such as searching a database, or collecting statistics.

A. Our Approach

In a streaming algorithm, we have a stream of data $x_1, x_2, x_3, \ldots$ arriving at the processing center and the goal is to compute a function of the stream. In our motivating case study, we want to detect if there exists an element x_i such that $f(x_i) = 1$ for the simple function f described in Equation 1, where $x_i = (\text{QT}_i, \text{RR}_i)$ (encrypted under an FHE scheme). Concisely, we wish to compute $\bigvee_{i=1}^{N} f(x_i)$.

Since FHE allows for computing over encrypted data, the obvious approach is to send encryptions of the data elements to the cloud, homomorphically evaluate f on each element in the incoming stream, and then compute the "OR" of the result of the computations (again, homomorphically). As we show in our experimental results, this solution is computationally costly. This is because homomorphic operations are inherently incredibly expensive. In fact, the known FHE schemes have a different cost model where performing a multiplication operation homomorphically is typically far more expensive than an addition operation and the cost of multiplication grows exponentially with the multiplication-depth (i.e., a cascaded set of multiplications).

A simple calculation will show that in order to do this following the naive approach we need a depth $d = \text{DEPTH}_f + \log n$ to process n data elements, where DEPTH_f is the multiplication-depth of f. The main contribution of our work is to show how we can significantly improve the computational efficiency by relying on an alternative representation of the computation that will significantly reduce the depth of the computation. In addition, our method will be inherently parallelizable and have small input locality.

Most FHE implementations show how to compute any circuit C over encrypted data. Our starting point deviates from this by first representing the function f as a *branching program* instead of a circuit. A branching program is a directed acyclic graph with a special start node s and final node t where each edge is labeled with either an input bit or its negation. The result of the computation is true if there is a path from the start to the final node traversing only edges for which the assignment sets the value on the edge true.

Next, we show how to use an FHE scheme to evaluate a branching program and aggregate the results of the computation over streaming data. Using elementary linear algebra we can show that evaluating a branching program is equivalent to evaluating the determinant of a particular matrix. More precisely, the detereminant will be $f(x)$ for the matrix corresponding to input x. Given the matrix representation of two inputs x_1 and x_2, computing the "AND" of $f(x_1)$ and $f(x_2)$ now reduces to simply multiplying the matrices corresponding to the inputs, since $det(AB) = det(A)det(B)$.

On a high level, our idea is to compute the elements of the matrix for each element homomorphically and then multiply the matrices corresponding to all elements in the data stream. We can already see the benefit of our approach from observing that matrix multiplication is inherently parallelizable. The main benefit, however, will result from the low multiplicative depth of our computation. In fact, the depth of our computation will be $\log n$. An overview of the process we've just described is shown in Figure 3. We will now explain our approach in detail. We first describe the matrix representation of computing a branching program, and then describe our system.

B. Path Counting in Directed Acyclic Graphs

Let P be a branching program of size $n + 1$. Let G be the corresponding directed acyclic graph and $A(G)$ denote the adjacency matrix corresponding to G. By the definition of a branching program, each entry in A is either $0, 1, x_i$ or $\neg x_i$ for some input bit x_i. Given a matrix M, denote by $M_{i,j}$ the matrix obtained by removing the i^{th} column and j^{th} row (i.e. the cofactor matrix corresponding to element (i, j)). The required matrix representation is $(I - A(G))_{s,t}$ where s and t are the indices corresponding to the start and terminal node in G, and I is the identity matrix of size $(n - 1) \times (n - 1)$.

To see why this is true we start with the following simple observation. If there is an unique path from s to t in G, then the $(s, t)^{th}$ entry of $\sum_{j=1}^{\infty} A^j$ when performed in GF(2) will be 1. Since $\sum_{j=1}^{\infty} A^j = (I - A)^{-1}$ it suffices to obtain the $(i, j)^{th}$ entry of $(I - A)^{-1}$. By Cramer's rule, this is given by $det(M_{i,j})/det(M)$ where $M = I - A$.

C. Proposed System

Our proposed system has three phases: pre-computation, cloud computation, and post computation. In the pre-computation phase, input data elements are encrypted under FHE and streamed to the cloud. In the cloud-computation phase, for each data element, the cloud generates the matrix representing the computation and aggregates the values via matrix multiplication. The final matrix that is the product of all matrices is downloaded and decrypted at the client. To compute the result of the computation in the post-computation phase, the client evaluates the determinant of the decrypted matrix.

D. Using HElib

The HELib library we will use relies on the BGV FHE encryption scheme [11]. An important optimization in HELib is that it considers packed ciphertexts. This extension of the BGV scheme allows for a vector of input bits to be encrypted/packed in a single ciphertext. A homomorphic operation on two packed ciphertexts simply returns the ciphertext corresponding to applying the operation coordinate-wise on the corresponding vectors encrypted in the ciphertexts.

In our approach, we show how the packed ciphertext allows for efficient matrix representation and multiplication. We pack the entries of a matrix diagonally. More precisely, given a $n \times n$ matrix, we encrypt it into n packed-ciphertexts where the i^{th} ciphertext c_i encodes the entries with indices $(0, i \bmod n), (1, i+1 \bmod n), ..., (n-1, i+n-1 \bmod n)$. For any matrix A, let $A\{i\}$ denote the i^{th} diagonal as described above. Then for any two matrices A and B, we can compute the i^{th} diagonal of $C = AB$ using the following fomula

$$C\{i\} = \sum_{j=0}^{n-1} A\{j\} B\{i-j \bmod n\}^j \tag{2}$$

where $M\{i\}^j$ denotes the rotation of the vector $M\{i\}$ by j positions to the right. This approach not only allows us to harness the packed representation of the input, but it also outputs a matrix with the same representation; this means we can repeatedly multiply several matrices.

E. Optimizations and Scalability

Our proposed approach works for any boolean function that can be represented as a branching program. A branching program can compute any boolean function by simply branching on all n input bits so that each of 2^n inputs result in a unique path. The size of this program is exponential in n. However, we are interested in what functions are representable by polynomial-size branching programs. From complexity theory, we know that any logspace computations can be represented as a branching program and can encompass a wide variety of problems. Therefore, our approach can be used for any logspace computable boolean functions. The size of the matrix corresponds to the size of the graph. For the equation in our case study, we construct a branching program of size ~ 600.

We observe that if the matrix representing the function is sparse, we need less than n packed ciphertexts to encode it. Since we employ the diagonal packing, we need the elements to be concentrated in the diagonals; such matrices are known as band matrices. A band matrix of width w has non-zero entries only in the diagonals $A\{-w\}, A\{-w+1\}, \ldots, A\{w\}$. Furthermore, multiplying two band matrices with width w results in a band matrix of width at most $w+1$. Moreover, each diagonal of the resulting matrix will depend on at most w ciphertexts of each input matrix. However, to utilize this optimization there are two issues: First, we need to understand what kind of branching programs translate into small-width band matrices. Second, what functions are computable by such branching programs ?

The first issue is relatively simpler to answer since there is a corresponding notion of a width of a branching program which will translate to the width of the corresponding matrix. A branching program is said to have width w if for all $i \in [n]$, the number of vertices reachable in exactly i steps from the start node is at most w. Such branching programs are described as *layered branching programs*, where there are only w nodes in each layer and edges go across only from each layer to the next layer. Now, given any width w branching program, the corresponding matrix representing the program will have width at most $w+1$.

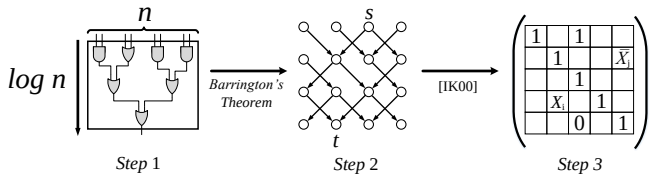

Fig. 3. Converting an algorithm (i.e. a circuit) to matrix form.

Next question is what functions can be computed by small-width branching programs ? Towards answering this, we recall a fundamental theorem due to Barrington [25], from complexity theory. Barrington's theorem states that the complexity of functions computable by constant-width branching programs is exactly the set of circuits with depth $O(\log n)$, referred to in the literature as NC^1. The class NC^1 is sufficiently rich and in particular includes all polynomial equations. The branching program we construct for our case study has width 10, and the corresponding band matrix has width 9.

V. Performance Evaluation

A. Experimental Setup

Since the HELib library is not thread-safe, to compare the performance of the two approaches we must simulate the computation to estimate the time taken on parallel machines. To accomplish this, we first perform real benchmarks of each individual operation on a single thread, and then use that time to estimate the computational costs for our parallel experiments. In our estimate we assume that each parallel machine has instantaneous access to the input encryptions and results of computations from other machines. (In essence, we ignore the data transfer and sharing costs.)

a) Naïve method: To compute Equation 1, we need to compute QT^3 and $(1/2)^3 RR$ and then compare the two results. To evaluate QT^3 we first generate the n addends in computing QT^2 by multiplying each bit of QT with QT and then shifting the result. More precisely, $Q_i = (QT \times QT_i) \ll i$. Then QT^3 is obtained by computing $Q_i \times QT$ for every i and summing them up. Towards this, we again expand each of these multiplications into n addends each to get a total of n^2 addends. We put $-\frac{1}{2^3} RR$ as another addend, which is computed by left shifting and taking 2's complement. To sum the n^2+1 addends, we repeatedly use the 3-2 compressors and a single 2-bit adder.

b) Matrix method: For every input, we generate the packed ciphertexts homomorphically. This simply uses the rotate and select operations, and we need to generate only 10 packed ciphertexts for our branching program. We evaluate the product of the matrices using Equation 2. Computing each diagonal of the resulting matrix requires at most 10 rotate and 10 multiply operations and 10 additions.

B. Experimental Results

To evaluate our approach we considered processing and aggregating several different numbers of samples. In Figure 4,

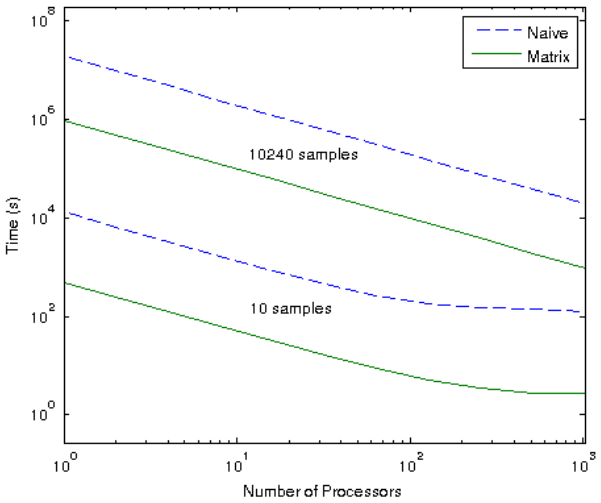

Fig. 4. Simulated computation time for matrix method vs. naive method of Long QT detection. The matrix method is consistently about 20x faster.

we provide our results for processing 10 and 10240 samples. We can see that both approaches improve with the number of processors. However, the matrix approach is consistently better than the naive approach by a factor of 20. The main reason for this is that the cost of computation increases exponentially with the depth of the computation and the depth of the naive approach is significantly higher than that of the matrix approach. One tradeoff that is not represented in the graph is the amount of network bandwidth used. The communication from the patient's end will be the same for both approaches, but at the doctor's end, an entire matrix (i.e. roughly 20 ciphertexts will have to be downloaded in our approach while only 1 ciphertext will need to be downloaded in the naive approach. However, in our application, this will only be done once every day (or few hours), so this should not be a significant problem. Another difference with the matrix approach is that the doctor's computer must compute the determinant of the decrypted matrix in order to get the result; in the naive approach, the result will be directly available (or be a simple "OR" of the decrypted bits).

VI. CONCLUSIONS AND FUTURE WORK

The advent of small, low-power sensors and embedded microprocessors has cleared many of the hurdles for medical remote monitoring. Serious privacy issues remain, though, which conventional techniques fail to address. In theory, FHE is a solution, but in practice it is too slow. We have presented a technique that improves computational efficiency and scalability of FHE for a large set of applications, resulting in 20x speedup for a representative application. The next stage of our research will focus on creating a true parallel/GPU implementation of this technique, and will most likely involve modifying HElib to use a thread-safe number theory library.

REFERENCES

[1] Andrew Reichman, "File storage costs less in the cloud than in-house," Tech. Rep., Forrester, August 2011.

[2] Chandrakant D. Patel and Amip J. Shah, *Cost Model for Planning, Development and Operation of a Data Center*, HP Laboratories Palo Alto, June 2005.

[3] Scott Good, "Why Healthcare Must Embrace Cloud Computing," *Forbes*, May 2013.

[4] 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.

[5] "CareCloud," March 2014.

[6] "EVault," March 2014.

[7] "Homomorphic Encryption Breakthrough," July 2009.

[8] Shai Halevi and Victor Shoup, "Algorithms in helib," *IACR Cryptology ePrint Archive*, vol. 2014, pp. 106, 2014.

[9] 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.

[10] 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.

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

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] Craig Gentry, "Fully homomorphic encryption using ideal lattices.," in *STOC*, 2009, vol. 9, pp. 169–178.

[18] Zvika Brakerski and Vinod Vaikuntanathan, "Lattice-based fhe as secure as pke," in *ITCS*, 2014, pp. 1–12.

[19] Yuval Ishai and Anat Paskin, "Evaluating branching programs on encrypted data," in *TCC*, 2007, pp. 575–594.

[20] Tomas Sander, Adam L. Young, and Moti Yung, "Non-interactive cryptocomputing for nc^1," in *FOCS*, 1999, pp. 554–567.

[21] "World's Thinnest 3-Lead ECG Patch," July 2014.

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

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

[24] Louis Sigurd Fridericia, "Die Systolendauer im Elektrokardiogramm bei normalen Menschen und bei Herzkranken," *Acta Medica Scandinavica*, vol. 53, pp. 469–486, 1920.

[25] David A. Mix Barrington, "Bounded-width polynomial-size branching programs recognize exactly those languages in nc^1," *J. Comput. Syst. Sci.*, vol. 38, no. 1, pp. 150–164, 1989.