SecureMed: Secure Medical Computation using GPU-Accelerated Homomorphic Encryption Scheme

Alhassan Khedr, *Member, IEEE,* and Glenn Gulak, *Senior Member, IEEE*

***Abstract*—Sharing the medical records of individuals among healthcare providers and researchers around the world can accelerate advances in medical research. While the idea seems increasingly practical due to cloud data services, maintaining patient privacy is of paramount importance. Standard encryption algorithms help protect sensitive data from outside attackers but they cannot be used to compute on this sensitive data while being encrypted. Homomorphic Encryption (HE) presents a very useful tool that can compute on encrypted data without the need to decrypt it. In this work, we describe an optimized NTRU-based implementation of the GSW homomorphic encryption scheme. Our results show a factor of $58\times$ improvement in CPU performance compared to other recent work on encrypted medical data under the same security settings. Our system is built to be easily portable to GPUs resulting in an additional speedup of up to a factor of $104\times$ (and $410\times$) to offer an overall speedup of $6085\times$ (and $24011\times$) using a single GPU (or four GPUs), respectively.**

***Index Terms*—Homomorphic Encryption, FHE, NTRU, Medical Applications, Relational Operations, Implementation, GPU.**

I. Introduction

The privacy of sensitive personal information is an increasingly important topic as a result of the increased availability of cloud services. These privacy issues arise due to the legitimate concern of a) having a security breach on these cloud servers or b) the leakage of this sensitive information due to an honest but curious individual at the cloud service provider. Standard encryption schemes try to address the first concern by devising encryption schemes that are harder to break, yet they don't solve the possible misuse of this sensitive data by the cloud service providers.

Homomorphic encryption (HE) presents a tool that can solve both types of privacy concerns. The clients are given the possibility of encrypting their sensitive information before sending it to the cloud. The cloud will then compute over their encrypted data without the need for the decryption key. By using HE, servers guarantee to the clients that their valuable information is never in the clear. A fully homomorphic encryption scheme (FHE) is an encryption scheme that allows evaluation of arbitrary functions on encrypted data.

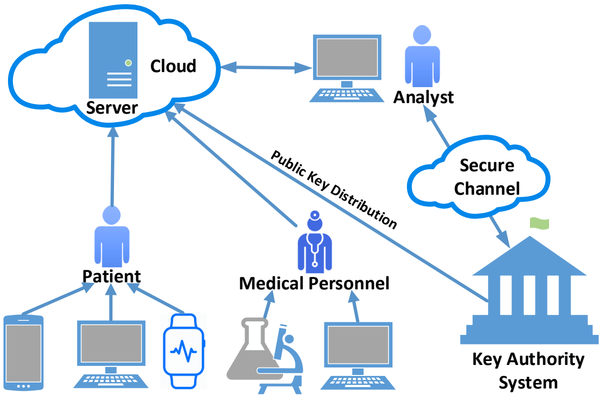

Fig. 1: High level block diagram for the medical data/key-distribution system.

In the era of Internet of Things (IoT), homomorphic encryption can be used to encrypt the data measured by wearable and portable medical devices prior to uploading them to the cloud. This can be very useful to help researchers and clinicians to conduct research or diagnose using this secure data. These devices can store public encryption keys produced by a centralized entity, which is also responsible for the distribution of secret keys to the hospitals through secure channels. The challenge here is that these portable devices have modest computing power compared to general purpose processors. Fortunately, these devices only need to encrypt the measurement data. Since performance of the encryption function is not time latency critical, these embedded processors can encrypt sensor data within seconds instead of milliseconds and still offer acceptable performance to real applications.

A proposed way to handle medical data measurements, analysis, and key distribution system is demonstrated in Figure 1. A central medical authority, such as the Ontario Laboratories Information System (OLIS), will be responsible for generating secret and public keys. Public keys will then be distributed, using the wide-area network, among medical laboratories and downloaded to portable and wearable medical devices. The medical data generated by the medical personnel in laboratories and by patients using their devices will then be encrypted by the public key and uploaded to the cloud. All patient medical data can be stored on the cloud servers safely as the HE scheme is provably secure against attacks. Analysts, administrators, or clinicians can run experiments on the encrypted medical data without having any secret keys. In order to finally decrypt the encrypted experiment results, the researchers will need to gain access to the secret keys from the key authority system using a secure channel. This secure

Alhassan Khedr is with the Department of Electrical and Computer Engineering, University of Toronto, Toronto, ON, Canada.
E-mail: alhassan@ece.utoronto.ca
Glenn Gulak is with the Department of Electrical and Computer Engineering, University of Toronto, Toronto, ON, Canada.
E-mail: gulak@ece.utoronto.ca

channel can be a virtual private network (VPN), a secure physical flash drive, or any other secure way to gain access to the secret keys.

The first FHE scheme was proposed by Craig Gentry [1], [2]. Since then, we have seen rapid development in the theory and implementation of homomorphic encryption schemes. HE schemes can now be based on a variety of cryptographic assumptions – approximate greatest common divisors [3], [4], learning with errors (LWE) [5], [6], [7], [8], Ring-LWE (RLWE) [9], [10], [11], [12] and NTRU [13], [14], [15], [16], [17]. Simple computations such as matching entries in a database and computing on the matched items were impossible using standard encryption schemes until the work done in SHIELD [12] where multiple classification functions were implemented including searching in databases and computing on the result using a homomorphic encryption scheme.

The main contributions of this work is the introduction of a NTRU based version of SHIELD [12] to reduce its computational complexity by a factor of $4.15\times$. Our GPU implementation of the HE scheme acheives a ciphertext (Ctxt) multiplication run time of 0.838 milliseconds, and the CPU implementation requires 87.8 milliseconds (See Table V for the design environment). Our CPU implementation acheives a speedup of $58\times$ over work in [18]. Our GPU implementation gives us a further $104\times$ (and $410\times$) speedup with overall speedup of $6085\times$ (and $24011\times$) over [18] using a single GPU (and four GPUs), respectively.

This paper is organized as follows. Section II presents related work. In Section III we introduce the improved encryption scheme. Some examples of secure medical applications are introduced in Section IV. Performance results are introduced in Section VI. Finally we conclude in Section VII.

II. Related Work

NTRU is a ring-based encryption scheme first proposed in [13]. Previous constructions of ring-based FHE schemes including NTRU are [14], [15], [5], [6], [11]. One of the drawbacks of these schemes is the need to maintain a so-called “modulus chain” which increases the size of the prime number and consequently increases the ring dimension for the same security level [19]. They also need to perform expensive modulus and key switching operations. In [14], [15], their homomorphic evaluations are on plaintexts mod 2 (binary arithmetic). Furthermore, the evaluation keys used in the Ctxt multiplication are in the GByte range which significantly limits their performance. The work in [20] is the closest one to our work since they also combined the NTRU scheme with GSW scheme concepts in [7]. Yet, in [20], they still require the flatten operation in the GSW scheme which leads to large memory usage and more computation time. They are also able to decrypt only a single bit from one polynomial and discard the remaining $\ell - 1$ polynomials. Whereas in our implementation, we extract a single bit of the message out of each polynomial which helps us retrieve ℓ bits messages. Moreover, in [20], since they use the flatten operation and their matrices are broken down into bits, they execute matrix multiplication using circular convolution rather than using the NTT transform. This is reasonable in the case of bit-wise multiplication, but when they tried to reduce the computation and storage costs by grouping bits together, the circular convolution became much harder without the NTT transform.

Based on [5], Halevi and Shoup designed a homomorphic encryption library [21], [11], but due to the need of additional large data structures and functions, the performance of their library was diminished. In [10] they implemented a variant of the RLWE FHE scheme. Our results also show considerable speedups over their implementation. Another homomorphic library was developed by Rohloff, Cousins, and Peikert [22]. In their paper they implement primary building blocks in hardware to accelerate their system. Presently, there are no performance results available with which to compare our library. Our work is a NTRU variant of SHIELD [12] which was based on RLWE. This resulted in a $4\times$ reduction in the ciphertext size and $2\times$ speedup in performance compared to [12]. Other researchers have proposed implementations but they were either an incomplete implementations of an HE scheme capable of only performing one multiplication operation [23], or based on other cryptographic assumptions such as approximate greatest common divisor, ideal lattices, etc.

Some applications analyzed in this paper were primarily inspired from [24], [18], [25]. The work in [18] exhibits slow running time. The work in [24] on the other hand has considerably faster running times but at the expense of having an incomplete implementation of the HE scheme that needs to provide the client with information about the depth of the computation made in order to correctly decrypt the result. The authors in [24] mentioned a slowdown by a factor of $50\times$ when using the complete HE implementation. This may raise security concerns from the server side due to the leakage of some of information about the applied function. In [26] Fujitsu laboratories used simple polynomial multiplication to compute correlation between different biometric samples but their performance is not representative since their function $F = C_1 \times C_2$ needs only a single ciphertext multiplication. For a simple and general overview of homomorphic encryption concepts the reader is encouraged to read [27], [28].

III. The Encryption System

Notation: For an odd prime number q we identify the ring $\mathbb{Z}/q\mathbb{Z}$ (or $\mathbb{Z}_q$) with the interval $(-q/2, q/2) \cap \mathbb{Z}$. The notation $[x]_q$ denotes reducing x modulo q. Our implementation uses polynomial rings defined by the cyclotomic polynomials $R = \mathbb{Z}[X]/\Phi_m(X)$, where $\Phi_m(X) = x^n + 1$ is the irreducible mth cyclotomic polynomial, in which n is a power of 2 and $m = 2n$. We let $R_q = R/qR$. Any type of multiplication including matrix and polynomial multiplication is denoted by the multiplication operator '$\cdot$'. Rounding up to the nearest integer is denoted by $\lceil a \rceil$. Matrices of rings are defined as $A_{M\times N}$, where $A_{ij} \in R_q$ and M, N are the matrix dimensions. $I_{\ell\times\ell}$ represents the identity matrix of rings.

A. The Encryption Scheme

The parameters of the system are n, the degree of the number field; q, the modulus; σ_k and σ_c, the standard deviation of the discrete Gaussian error distribution in the key space and ciphertext space respectively; $\ell \triangleq \lceil \log q \rceil$ that governs the number of ring elements in a ciphertext. The setting of these parameters depends on the security level λ (e.g., $\lambda = 80$ or 128 bits) as well as the complexity of functions we expect to evaluate on ciphertexts.

Bit Decompose Function "$\mathsf{BD}()$": The bit decompose function $\mathsf{BD}(integer)$ takes an ℓ-bit input integer, then outputs a row vector with size ℓ containing the bit decomposition of this integer. Similarly, $\mathsf{BD}(polynomial)$ takes an input polynomial of size n, where each coefficient is an ℓ-bit integer, then outputs an ℓ-sized row vector of polynomials (each of size n) containing the bit decomposition of each coefficient of the input polynomial, yielding a matrix of size $l \times n$. Finally, $\mathsf{BD}(\textit{Matrix of polynomials})$ takes an input matrix of polynomials of size $x \times y$ (each polynomial is of size n with integer coefficients), then outputs a matrix of polynomials expanded by a factor ℓ in the column dimension, yielding a matrix of size $x \times y\ell$, where each consecutive ℓ elements along the row contain the bit representation of each coefficient of each of the input polynomials. For example, the bit decompose of the input polynomial matrix $B_{x\times y\times n}$ is $\mathsf{BD}(B_{x\times y\times n}) = \beta_{x\times y\ell\times n}$. The reader should note that despite the fact that the polynomial coefficients of matrix $\beta_{x\times y\ell\times n}$ are single bit values, the storage requirement of matrix β in CPU or GPU memory is not equal to $x \times y\ell \times n$ bits. This is due to the fact that the smallest addressable unit of memory is a byte (i.e., Byte Addressable). Hence, β requires $x \times y\ell \times n$ *bytes* of storage. This results in the further observation that the storage requirement of $\beta_{x\times y\ell\times n}$ is 8× the storage requirement of $B_{x\times y\times n}$.

Bit Decompose Inverse Function "$\mathsf{BDI}()$": As the name reveals, the $\mathsf{BDI}()$ function is the inverse of $\mathsf{BD}()$. The $\mathsf{BDI}()$ function groups consecutive ℓ coefficients along a row (the coefficients don't need to be binary), and outputs the integer corresponding to those ℓ bits. Mathematically, the $\mathsf{BDI}()$ function can be defined as multiplying the *expanded* matrix of polynomials $\beta_{x\times y\ell}$ from the right by the matrix $\alpha_{y\ell\times y}$ defined in (1) (polynomial dimension n will be omitted from this point forward for clarity). Hence $B_{x\times y} = BDI(\beta_{x\times y\ell}) = \beta_{x\times y\ell} \cdot \alpha_{y\ell\times y}$.

$$\alpha_{y\ell\times y} = \begin{pmatrix} 1 & 0 & 0 & \cdots & 0 \\ 2 & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 2^{\ell-1} & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 \\ 0 & 2 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 2^{\ell-1} & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \\ 0 & 0 & 0 & \cdots & 2 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 2^{\ell-1} \end{pmatrix} \tag{1}$$

In this work, we present a NTRU variant of the encryption scheme presented in [12] in order to reduce computational complexity and to speedup our operations, as will be detailed below. Our encryption system works as follows.

- $\mathsf{Keygen}(1^\lambda)$: Choose two polynomials $f_{1\times1}, g_{1\times1} \leftarrow D_{\mathbb{Z}^n,\sigma_k}$ such that (a) f is invertible in the ring R_q; and (b) $f \equiv 1 \pmod 2$. This is done by simply sampling the polynomial f from the distribution $D_{\mathbb{Z}^n,\sigma_k}$ until it satisfies conditions (a) and (b).

  The public key pk and the secret key sk can be computed from (2).

  $$\mathsf{pk} = h_{1\times1} = g_{1\times1} \cdot f_{1\times1}^{-1} \in R_q \quad \mathsf{sk} = f_{1\times1} \in R_q \tag{2}$$

- $\mathsf{Enc}(\mathsf{pk}, m)$: The message space of our encryption scheme is R_q. Encrypt the plain text polynomial $\mu \in R_q$ by calculating

  $$C_{\ell\times1} = \mu \cdot \mathsf{BDI}(I_{\ell\times\ell}) + S_{\ell\times1} \cdot h_{1\times1} + E_{\ell\times1} \tag{3}$$

  where $S_{\ell\times1}, E_{\ell\times1} \leftarrow D_{R_q^{\ell\times1},\sigma_c}$ are sampled from a discrete Gaussian distribution with standard deviation σ_c.
- $\mathsf{Dec}(\mathsf{sk}, C)$: Given the ciphertext C, the plaintext $\mu \in R_q$ is restored by multiplying C by the secret-key f as follows :

  $$\begin{aligned} C_{\ell\times1} \ \cdot \ f_{1\times1} &= (\mu \cdot \mathsf{BDI}(I_{\ell\times\ell}) \\ &\quad + S_{\ell\times1} \cdot h_{1\times1} + E_{\ell\times1}) \cdot f_{1\times1} \\ &= \mu \cdot \mathsf{BDI}(I_{\ell\times\ell}) \cdot f_{1\times1} \\ &\quad + S_{\ell\times1} \cdot h_{1\times1} \ \cdot \ f_{1\times1} + E_{\ell\times1} \ \cdot \ f_{1\times1} \\ &= \mu \cdot \mathsf{BDI}(I_{\ell\times\ell}) \cdot f_{1\times1} \\ &\quad + S_{\ell\times1} \cdot g_{1\times1} + E_{\ell\times1} \ \cdot \ f_{1\times1} \\ &= \mu \cdot \mathsf{BDI}(I_{\ell\times\ell}) \cdot f_{1\times1} + error_{\ell\times1}. \end{aligned} \tag{4}$$

  The decrypted Ctxt in (4) can be reformulated as in (5) and the bit representation of each coefficient μf_i of the μf polynomial can be represented as in (6).

$$\begin{aligned} C_{\ell\times 1} \ \cdot \ f_{1\times 1} &= \mu f \cdot \begin{bmatrix} 1 \\ 2 \\ \vdots \\ 2^{\ell-1} \end{bmatrix} + error_{\ell\times 1} \\ &= \begin{bmatrix} \mu f \\ 2\mu f \\ \vdots \\ 2^{\ell-1}\mu f \end{bmatrix} + error_{\ell\times 1} \end{aligned} \tag{5}$$

$$\mu f_i = \begin{bmatrix} (\mu f_i)_{\ell-1} & (\mu f_i)_{\ell-2} & \cdots & (\mu f_i)_2 & (\mu f_i)_1 & (\mu f_i)_0 \end{bmatrix} \tag{6}$$

The result in (5) consists of an $\ell \times 1$ vector of polynomials which are in the form $\mu f, 2\mu f, \cdots, 2^{\ell-1}\mu f$ in addition to the error vector. Ignoring the modular reduction effect for simplicity, which would effect only the lower bits, the shifted versions of μf can be visualized as in (7).

$$\begin{bmatrix} \mu f \\ 2\mu f \\ \\ \vdots \\ 2^{\ell-2}\mu f \\ 2^{\ell-1}\mu f \end{bmatrix} = \begin{bmatrix} (\mu f)_{\ell-1} & \cdots & (\mu f)_1 & (\mu f)_0 \\ (\mu f)_{\ell-2} & \cdots & (\mu f)_0 & 0 \\ \vdots & \vdots & \ddots & \vdots \\ (\mu f)_1 & (\mu f)_0 & \cdots & 0 \\ (\mu f)_0 & 0 & \cdots & 0 \end{bmatrix} \tag{7}$$

Note that the left most column of the result in (7) are the individual bits of the μf polynomial. This is true for each coefficient "i" in the polynomial μf. This way we can successfully construct back μf, assuming that the error is less than $q/2$, which can then be multiplied by f^{-1} to recover μ.

Remark: In addition to the ability to recover the message polynomial μ, the ℓ ring elements in the ciphertext facilitate and manage the noise growth in homomorphic operations (in particular, homomorphic multiplication) as we will describe shortly.

1) Homomorphic Operations: For input ciphertexts $C_{\ell\times 1}$ and $D_{\ell\times 1} \in R_q^{\ell\times 1}$ encrypting μ_1 and μ_2 respectively, homomorphic operations are defined as follows.

- $\mathsf{ADD}(C, D)$: To add the two ciphertexts $C_{\ell\times 1}$ and $D_{\ell\times 1}$, simply output $C_{\ell\times 1} + D_{\ell\times 1}$, which is an entry-wise addition.
- $\mathsf{MULT}(C, D)$: To multiply the two ciphertexts $C_{\ell\times 1}$ and $D_{\ell\times 1}$, output $\mathsf{BD}(C_{\ell\times 1}) \cdot D_{\ell\times 1}$.

Correctness of homomorphic addition is immediate, however correctness is not that obvious for homomorphic multiplication. It is clear that the multiplication algorithm is asymmetric in the input ciphertexts C and D. That is, we treat the components of D as a whole, whereas the components of C are broken up into their "bit-wise decompositions". This is a "feature" that is inherited from the work of BV [8]. It is shown below that this multiplication method is correct and gives a slow noise-growth rate.

The correctness of the multiplication operation is evident from the decryption operation in (8). Matrix dimensions are removed for clarity.

$$\begin{aligned} \mathsf{BD}(C) \cdot D \cdot f &= \mathsf{BD}(C) \cdot (\mu_2 \cdot \mathsf{BDI}(I) + S_2 \cdot h + E_2) \cdot f \\ &= \mathsf{BD}(C) \cdot (\mu_2 \cdot \mathsf{BDI}(I) \cdot f + S_2 \cdot g + E_2 \cdot f) \\ &= \mu_2 \cdot C \cdot f + \mathsf{BD}(C) \cdot (S_2 \cdot g + E_2 \cdot f) \\ &= \mu_2 \cdot (\mu_1 \cdot \mathsf{BDI}(I) \cdot f + S_1 \cdot g + E_1 \cdot f) \\ &\quad + \mathsf{BD}(C) \cdot (S_2 \cdot g + E_2 \cdot f) \\ &= \mu_2 \cdot \mu_1 \cdot \mathsf{BDI}(I) \cdot f + \mu_2 \cdot (S_1 \cdot g + E_1 \cdot f) \\ &\quad + \mathsf{BD}(C) \cdot (S_2 \cdot g + E_2 \cdot f) \\ &= \mu_2 \cdot \mu_1 \cdot \mathsf{BDI}(I) \cdot f + \mu_2 \cdot error_1 \\ &\quad + \mathsf{BD}(C) \cdot error_2 \\ &= \mu_2 \cdot \mu_1 \cdot \mathsf{BDI}(I) \cdot f + error. \end{aligned} \tag{8}$$

Note that the last line in (8) is the encryption of $\mu = \mu_2 \cdot \mu_1 \cdot f$. Note also that $\mathsf{BD}(C_{\ell\times 1}) \cdot \ \mathsf{BDI}(I_{\ell\times\ell}) = I_{\ell\times\ell} \cdot \ C_{\ell\times 1} = C$.

Noise Analysis: Correct decryption depends crucially on the ciphertext noise being bounded. Thus, it is important to understand how homomorphic operations increase ciphertext noise. Let C be a fresh ciphertext. We make the following observations, after [8].

- Homomorphic addition of v ciphertexts increases the noise by a factor of v, in the worst case. In practice, since the coefficients of the error polynomials follow a Gaussian distribution, the factor is closer to $O(\sqrt{v})$.
- Homomorphic multiplication is significantly more interesting. Multiplication of two fresh ciphertexts $Z_1 = C_1 \times C_2$ where $C_1 = \mathsf{Enc}(\mu_1)$ and $C_2 = \mathsf{Enc}(\mu_2)$, with the messages μ_i bounded by κ and error bounded by B, increases the error (E_2) to $E_2 = O(B \cdot \|\kappa\|_1 + B \cdot n \log q)$ in the worst case, and $E_2 = O(B \cdot \|\kappa\|_1 + B \cdot \sqrt{n \log q})$ in practice. Here, $\|\mu\|_1$ denotes the ℓ_1 norm of the message polynomial μ. Multiplying by another fresh Ctxt $Z_2 = Z_1 \times C_3$ increases the error to $E_3 = E_2 \cdot \|\kappa\|_1 + B \cdot n \log q$. Substituting with E_2

$$\begin{aligned} E_3 &= (B \cdot \|\kappa\|_1 + B \cdot n \log q) \cdot \|\kappa\|_1 + B \cdot n \log q \\ &= B \cdot \|\kappa\|_1^2 + B \cdot \|\kappa\|_1 \cdot n \log q + B \cdot n \log q \\ &= B \cdot (\|\kappa\|_1^2 + (\|\kappa\|_1 + 1) \cdot n \log q) \end{aligned} \tag{9}$$

Following this analysis, multiplying L sequential Ctxts will increase the error to

$$\begin{aligned} E_L &= B \cdot (\|\kappa\|_1^{L-1} + (\|\kappa\|_1^{L-2} + \|\kappa\|_1^{L-3} + \cdots + 1) \cdot n \log q) \\ &= B \cdot (\|\kappa\|_1^{L-1} + \frac{1 - \|\kappa\|_1^{L-1}}{1 - \|\kappa\|_1} \cdot n \log q) \end{aligned} \tag{10}$$

How to Set Parameters: We base the security of our system relying on the analysis in [17], [19], and [29]. Let f be the function that we are evaluating that computes the multiplication of v ciphertexts. Let $\mathsf{error}_f(B, n, q)$ denote how much the error grows when evaluating a function f on ciphertexts in R_q

with an initial error of magnitude B. For correct decryption, we need

$$\mathsf{error}_f(B, n, q) < q/2 \tag{11}$$

Since error grows slower in our scheme relative to other FHE schemes [18], [24], [25], and since there is no need for a chain of moduli to control the noise growth, q can be set to be correspondingly smaller for the same security level. Following the analysis of Lindner and Peikert [19], for a security level of λ bits, we need

$$n > \log(q/\sigma)(\lambda + 110)/7.2 \tag{12}$$

Since our $\log q$ is smaller relative to [18], [24], [25], we can set our n to be smaller, for the same security level λ. In turn, since we now have a smaller n, our new $\mathsf{error}_f(B, n, q)$ is smaller, leading to an even smaller q, and so on. The optimal parameters are obtained by solving both the above inequalities together.

We are aware that the analysis done in [19] was made by using the older BKZ not the new, more efficient, BKZ 2.0 algorithm [30]. This is because Lindner-Peikert equations result in more conservative parameters than the ones from BKZ 2.0, as reported in [31], [32].

To protect our NTRU scheme against Subfield Lattice Attacks [29], the "*perfect immunity*", as described by the authors, can be achieved when

$$\sqrt{2} \cdot \sigma_k^2 \cdot \grave{n} > \sqrt{\grave{n} q/\pi e} \tag{13}$$

where $\grave{n} = n/2$. Fixing q and $\grave{n}$, the parameter σ_k should satisfy (14).

$$\sigma_k > \sqrt[4]{\frac{q}{2\grave{n}\pi e}} \tag{14}$$

which results in the condition that σ_k should be larger than or equal 23. The vulnerability factor[1] (15) is a ratio which represents the extent to which this NTRU scheme is vulnerable to subfield lattice attacks. The scheme is immune when $F < 1$.

$$F = \frac{\sqrt{\grave{n} q/\pi e}}{\sqrt{2}\sigma_k^2 \grave{n}} \tag{15}$$

We set $\sigma_k = 25$ to achieve vulnerability factor $F = 0.79$. Table I summarizes our final parameter selection.

Our encryption scheme is also immune against the recent attack on NTRU in [33]. Using the data in [33] (Table 2, Figure 1), which use polynomials f and g sampled from $[-1, 0, 1]$, to extrapolate a generalized equation for the minimum $\log(q)$ below which the algorithm in [33] fails[2], results in (16).

$$min(\log(q)) = 1.0127e^{0.3867 \log(n)} \tag{16}$$

Table I: Parameter Selection and Keys/Ctxt Sizes.

Parameter	**NTRU** (This work)
λ	173
n	1024
ℓ	31
σ_k	25
σ_c	10
SK size	$n \times \ell = 3.968\ KBytes$
PK size	$n \times \ell = 3.968\ KBytes$
Ctxt size	$\ell \times n \times \ell = 123.008\ KBytes$

Substituting $\log(n) = 10$ in (16) results in $min(\log(q)) = 48$ bits which offers a safe margin from the $\log(q) = 31$ bits used in this work. In addition, we sample the polynomials f and g from a Gaussian distribution with standard deviation $\sigma_k = 25$, not from $[-1, 0, 1]$. This improves our security level since it is directly proportional to σ, as can be deduced from (12) and (15).

IV. Candidate Applications

We chose to outline some candidate applications, first outlined in [24], [18], [25], in order to be able to compare the performance of our method.

A. Relational Operations

Originally, homomorphic addition and multiplication were the only available operations that can be applied on encrypted data. Afterwards, [12], [25] implemented a homomorphic equality operation through comparing individual bits of the encrypted data. Homomorphic relational operations $>$ and $<$ can enable new applications. The idea was originally presented in [34] where they introduced the idea of conditional gates. Later in [25] they applied this idea to homomorphic encryption. The relational operations $a > b$ results in one bit only, which is equal to 1 if $a > b$, and 0, otherwise. Individual bits of each input a and b are encrypted and denote them as a_i and b_i, respectively. Assume we have k-bit numbers, we start with the least significant bit and the output is given by z_k, where

$$z_0 = 0 \qquad z_{i+1} = (1 - (a_i - b_i)^2)z_i + a_i(1 - b_i). \tag{17}$$

if any of the two inputs were not encrypted, the complexity of (17) can be greatly reduced. For example, if b is not encrypted, the resulting equations will be

$$z_0 = 0, \quad z_{i+1} = \begin{cases} (1 - a_i)z_i + a_i & \text{if } b_i = 0 \\ a_i z_i & \text{if } b_i = 1 \end{cases} \tag{18}$$

Blood Pressure Test: As a proof of concept of the benefits of the homomorphic relational operations, a secure blood pressure application was built. The application simply accepts the encrypted systolic and diastolic pressures and returns the blood pressure classification according to Table II.

To implement this blood pressure example, we compare the input blood pressure to multiple ranges to decide the

[1] Equation (13) only indicates sufficiency to achieve perfect immunity, but it does not indicate to what extent the encryption scheme is vulnerable against subfield lattice attacks. This is represented by the vulnerability factor F in (15).

[2] We use the points in Table 2 / Figure 1 in [33] with $\log(r) = 1$ and 2 to get the worst case $min(\log(q))$.

Table II: Blood Pressure Classification

Category	Systolic mm Hg	Diastolic mm Hg
Hypotension	< 90	< 60
Desired	$90 - 119$	$60 - 79$
Prehypertension	$120 - 139$	$80 - 89$
Stage 1 hypertension	140–159	90–99
Stage 2 hypertension	160–179	100–109
Hypertensive emergency	≥ 180	≥ 110

Function 1: Secure Blood Pressure Classification

```
Input: Encrypted Blood Pressure Sys_Enc, Dia_Enc
Output: Blood Pressure Classification

Sys_Ref = [90, 120, 140, 160, 180]
Dia_Ref = [60, 80, 90, 100, 110]
For each number "i" in the reference lists {
    Res_Sys += Check_Greater(Sys_Enc, Sys_Ref[i]);
    Res_Dia += Check_Greater(Dia_Enc, Dia_Ref[i]);
}
Return Res_Sys and Res_Dia
```

blood pressure classification. This can be simply done using Function 1. The Res_Sys and Res_Dia parameters in Function 1 will each contain an encrypted number ranging from 0 to 5 indicating the classification of the input encrypted blood pressure measurements compared to the blood pressure range in the systolic Sys_Ref and diastolic Dia_Ref reference lists, respectively.

It is worth mentioning that the relational operation applications are numerous. The blood pressure example is just an illustrative example of its usefulness and power. For example, calculating the CHADS$_2$ Score for Atrial Fibrillation Stroke Risk is straightforward [35] in a similar manner.

Framingham Coronary Heart Disease Risk Score (FCRS): The FCRS is an algorithm used to estimate the 10-year cardiovascular risk of a person. For more information about this algorithm, the reader is referred to [36], [37]. The algorithm is based on several factors, including sex, age, cigarette smoking, total cholesterol, high-density lipoprotein (HDL) cholesterol, and systolic blood pressure. Each variable is compared against a range of numbers and a score is assigned for each range and added to the overall Framingham score. For example, if a man's age was ≥50 years, 6 points are added to the total score.

To implement this algorithm, the relational operations described in Section IV-A are not sufficient because it gives a "1" if the input is larger than a certain number and "0" otherwise. If we have an input "b" compared against "a" as a lower bound and "c" as an upper bound, we need to implement $a < b < c$ function which gives only a "1" if "b" is within this range and "0" otherwise.

The straight-forward way to implement $a < b < c$, is to implement it in two steps, namely $a < b$ and then $b < c$ and multiply the results. The only downside to this approach is that in our encryption scheme, in fact all GSW derived schemes in general, the homomorphic multiplication is asymmetric preventing the possibility of multiplying two non-fresh Ctxts. We mean by non-fresh Ctxts the Ctxts that contain results of Ctxt multiplications. In order to overcome this challenge, we reformulated the equations for $a < b < c$ such that they always multiply fresh Ctxts by an accumulator.

"$a < b < c$" Reformulated: As mentioned, to implement such a function we could multiply the results of individual relational operations. We used this idea to re-format the equations needed to implement this function. For a k-bit input "b" with encrypted bits $[b_0, b_1, \cdots, b_{k-1}]$ compared against "a" and "c" (encrypted or unencrypted) we can implement the function $a < b < c$ as follows: First we define for bit "0"

$$x_0 = b_0(1 - a_0) \qquad y_0 = c_0(1 - b_0) \qquad z_0 = 0. \tag{19}$$

and then for each bit "i"

$$\begin{aligned} x_{i+1} =& (1 - (b_i - a_i)^2)x_i + b_i(1 - a_i) \\ y_{i+1} =& (1 - (c_i - b_i)^2)y_i + c_i(1 - b_i) \\ z_{i+1} =& (1 - (b_i - a_i)^2)(1 - (c_i - b_i)^2)z_i \\ &+ (1 - (b_i - a_i)^2)c_i(1 - b_i)x_i \\ &+ (1 - (c_i - b_i)^2)b_i(1 - a_i)y_i \end{aligned} \tag{20}$$

where x_i, y_i, z_i are accumulators, b_i is encrypted bit i of input b, and a_i, c_i are the bits of the lower and upper bounds, respectively. z_k will contain the result of the relational operation. In the case that a_i and c_i are plaintexts, equations (20) can be further simplified as in (18).

It is worth mentioning that the number of Ctxt multiplications needed for this circuit, in the case of plaintext a and c, are 8 multiplications per bit, compared to only 2 Ctxts multiplications per bit for other HE schemes that supports multiplying two non-fresh Ctxts.

B. Encrypted Genomic Data

In [24] the authors described statistical algorithms used in genomic association studies. In this paper, we briefly summarize some of their algorithms namely the Pearson Goodness-of-Fit test and the Cochran-Armitage test for Trend (CATT). For more information about equation derivations we refer the reader to [24].

In order to implement the algorithms presented in [24], genotypes and phenotypes are encoded and encrypted as follows:

Genotype Encoding: As in [24], a table containing genotype information is constructed in which each row corresponds to genotype information about a single person. For bi-allelic genes, each person's gene is encoded using three ciphertexts c_{AA}, c_{Aa}, c_{aa}. These ciphertexts will encrypt a "1" only in the case that the equality statement in (21) is satisfied, otherwise they encrypt "0". For the case that the person's genotype at the specified locus is not known, all Ctxts will encrypt "0".

$$\begin{aligned} c_{AA} &= Enc(\text{gene} == AA) \\ c_{Aa} &= Enc(\text{gene} == Aa) \\ c_{aa} &= Enc(\text{gene} == aa) \end{aligned} \tag{21}$$

Genotype/Phenotype Correlation Encoding: We modified the phenotype encoding compared to [24] in order to reduce pre-computation time and to have less noise in the final encrypted result. This will allow us to further compute on the resulting ciphertext if needed. We generate a correlation matrix between genotypes {AA, Aa, aa} and phenotypes {affected, unaffected} to generate a 3×2 correlation matrix with Ctxts encrypting a "1" at a single location corresponding to this person's genotype/phenotype condition, "0" otherwise, as shown in Table III. If the genotype or phenotype of this person is unknown, then all Ctxts will be encrypting a "0".

Table III: Genotype/Phenotype Correlation Matrix

	AA	Aa	aa
Unaffected	z_{00}	z_{01}	z_{02}
Affected	z_{10}	z_{11}	z_{12}

Pearson Goodness-of-Fit Test: This test is used to check if a gene is in Hardy-Weinberg Equilibrium (HWE). The HWE is responsible for testing if the gene allele frequencies are independent. Assume A and a are two alleles in a given gene, and that N_{AA}, N_{Aa}, and N_{aa} are the corresponding number of genotypes AA, Aa, and aa, respectively, as in (23). Let $N = N_{AA} + N_{Aa} + N_{aa}$ be the total number of genotypes. Since homomorphic division is expensive to perform, in [24] they simplified the computations needed and computed the following parameters homomorphically

$$\begin{aligned} \alpha &= (4N_{AA}N_{aa} - N_{Aa}^2)^2 & \beta_1 &= 2(2N_{AA} + N_{Aa})^2 \\ \beta_2 &= (2N_{AA} + N_{Aa})(2N_{aa} + N_{Aa}) & \beta_3 &= 2(2N_{aa} + N_{Aa})^2 \end{aligned} \tag{22}$$

where N_{ij} can be computed as

$$N_{AA} = \sum_i c_{AA}^{(i)} \quad N_{Aa} = \sum_i c_{Aa}^{(i)} \quad N_{aa} = \sum_i c_{aa}^{(i)} \tag{23}$$

these parameters are then sent to the client to compute the final deviation test statistic

$$X^2 = \frac{\alpha}{2N}\left(\frac{1}{\beta_1} + \frac{1}{\beta_2} + \frac{1}{\beta_3}\right) \tag{24}$$

Cochran-Armitage Test for Trend (CATT): This study is used to determine if an allel is associated to a disease or not. The test can be computed as

$$X^2 = \frac{\alpha}{\beta} \tag{25}$$

where

$$\alpha = N\left(\sum_{i=0}^{2} w_i(N_{0i}R_1 - N_{1i}R_0)\right)^2 \tag{26}$$

$$\beta = R_0R_1\left(\sum_{i=0}^{2} w_i^2 C_i(N - C_i) - 2w_1w_2C_1C_2\right) \tag{27}$$

$$N_{xy} = \sum_i z_{xy}^{(i)}, \quad where \;\; x \in \{0,1\}, \;\; y \in \{0,1,2\} \tag{28}$$

where w_i are predetermined weights $\in \{0, 1, 2\}$ as described in [24], N_{ij} represents the number of individuals who are affected/unaffected with a certain disease and have genotype ij as in (28). The parameters N_{ij}, R_i, C_i are described in Table IV.

Table IV: Affected/Unaffected Genes Statistics

	AA	Aa	aa	Sum
Unaffected	N_{00}	N_{01}	N_{02}	R_0
Affected	N_{10}	N_{11}	N_{12}	R_1
Sum	C_0	C_1	C_2	N

C. Predictive Analysis

Predictive equations can be used to check for different diseases. In order to compute the predictive equation, patients' information must be used. To protect such information, homomorphic encryption can be used to encrypt these data and privately compute the regression equations [18]. To demonstrate the logistic regression analysis, in [18] they used a model that predicts the likelihood of having a heart attack in an unspecified period. The predictive model is given by the following logistic regression function

$$P(x) = \frac{e^x}{e^x - 1} \tag{29}$$

where x is represented as

$$\begin{aligned} x =& 0.072 \cdot a + 0.013 \cdot sys - 0.029 \cdot dia + 0.008 \cdot chol \\ & - 0.053 \cdot ht + 0.021 \cdot wt \end{aligned} \tag{30}$$

where a is the age, sys is the systolic blood pressure, dia is the diastolic blood pressure, $chol$ is the cholesterol level, ht is the height, wt is the weight.

To implement the exponential expression in (29), a Taylor series can be used up to an acceptable order to result in a sufficient accuracy. In [18] they represented (29) up to degree 7

$$P(x) = \frac{e^x}{e^x - 1} = \frac{1}{2} + \frac{1}{4}x - \frac{1}{48}x^3 + \frac{1}{480}x^5 - \frac{17}{80640}x^7 + O(x^9) \tag{31}$$

Since homomorphic encryption relies on integer arithmetic, (30) and (31) can be normalized with appropriate accuracy to the following

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JBHI.2017.2657458, IEEE Journal of Biomedical and Health Informatics

Table V: Design Environment.

Item	Specification
CPU	Intel Core-i7 5930K
# of CPU Cores	4
# of Threads	8
CPU Frequency	3.5 GHz
Cache Size	15 MB
System Memory	32 GB DDR4
Operating System	Windows 8.1 64-bits
Programming IDE	Visual Studio 2012 Ultimate edition
GPU	NVIDIA GeForce GTX980
Maxwell Version	GM204
# of CUDA Cores	2048
GPU Core Frequency	1126 MHz
GPU Memory	4 GB
GPU L2 Cache	2 MB

$$z = 72 \cdot a + 13 \cdot sys - 29 \cdot dia + 8 \cdot chol - 53 \cdot ht + 21 \cdot wt \quad (32)$$

$$F(z) = 40320 + 20160z - 1680z^3 + 168z^5 - 17z^7 \quad (33)$$

Inputs to (32) and (33) are represented as integer numbers and encoded in binary format in adjacent polynomial slots. When these binary polynomials are multiplied together, they result in a correct output as was described in [18]. Multiplication by constants in (32) were implemented through sequential additions. To compute the correct output, the result of (32) is sent back to the client to be re-encrypted and applied to (33) to compute the final result. Again, for more details, refer to [18].

V. Security against attacks

Our homomorphic encryption scheme and algorithms, and indeed all known FHE schemes, are proven secure in the IND-CPA sense (i.e., under a chosen plaintext attack). This is the standard notion of security for FHE schemes as in [1], [2], [6]. The algorithms in Section IV are secure against external attackers. They are also secure against an honest but curious server that wants to learn the underlying encrypted data without trying to actively change it. It is trivial that any homomorphic encryption scheme can be broken by CCA2 (i.e., if the adversary can make decryption queries after the challenge). It can also be broken by CCA1 attacks [38] (i.e., if the adversary can make decryption queries, but only before the challenge). The correctness of any FHE algorithm relies on the honesty of the server that it will execute the exact algorithm.

VI. Performance Results

Design Environment: A summary of the specifications of the system used to implement our work for the purpose of benchmarking is found in Table V.

Table VI summarizes the sets of parameters used for our library compared to [12], [24], [18]. We mean by the *Effective Security Level* that we substitute the reported parameters of each encryption scheme in Equation (12) and get the resulting λ. The following is a further explanation of why our parameters are smaller than the others despite the fact that we all derived our parameters from [19]. Our work and SHIELD [12] have the same n and $\log q$ because they are both based on [7]. The difference in the Ctxt and key sizes are due to the translation from the RLWE to the NTRU schemes. On the other hand, the work in [18] and [24] are both based on [31]. In [31], $\log q$ increases with the circuit depth as well as with the number of least significant bits reserved for the plaintext $\log t$ (the condition for correctness is $q > 2 \cdot t \cdot error$). In our scheme, $\log q$ increases only with the circuit depth, as we encode our plaintext in the most significant bit of each of the $\log q$ polynomials (the condition for correctness is $q > 2 \cdot error$). Also since we use the asymmetric property of the Ctxt multiplication in our favor, our noise growth, as well as our parameters, are smaller for the same circuit depth. We would like to point out that in [24], they did not use the 512MB evaluation key as in [18] to speedup their ctxt operations. This came at the expense that they generate a different secret decryption key for different circuits with different circuit depth. Also, the large $\log q$ used in [18], [24] along with the very small distribution used in the key generation, $\chi_{key} = \{-1, 0, 1\}$, make these schemes very vulnerable to the subfield lattice attacks in [29].

Table VII summarizes the performance results of the homomorphic operations for our library compared to the [12], [24], [18]. It can be seen from this table that we have a $58\times$ speedup for the multiplication operation of our CPU implementation compared to work in [18]. By additionally exploiting the parallelizable properties that our HE library has, we get another $104\times$ speedup by distributing the HE computations on the GPU cores. This resulted in an overall $6085\times$ speedup for the multiplication operation compared to work in [18] and a $286\times$ compared to [24].

Scalability: After exploring the parallelism in our system, we further explored the ability for its scalability across multiple GPU instances. Experiments were made using four GPUs cards connected to the same CPU to measure the loss in performance due to cross GPU communication. By partitioning large problems into small ones, we managed to schedule the work among all four GPUs to get a speedup of 3.946×, which indicates that we managed to hide almost all the communication overhead.

A. Candidate Applications Performance

The performance results of the Pearson Goodness-of-fit test and the CAAT test described in Section IV-B, the predictive analysis described in Section IV-C, and the relational operations and blood pressure application described in Section IV-A are summarized in Table VIII. We would like to point out that the reported results for the work in [25] corresponds to the ring Z_{14}, this is because in our blood pressure example we need to add more than two numbers to get the final correct result.

Table VI: Comparison between the parameters in this work and in [12], [18], [24].

Parameter	This Work	SHIELD [12]	[18]	[24]
Base Scheme	NTRU	RLWE	NTRU	NTRU
Polynomial Dimension "n"	1024	1024	16384	8192
Modulus Bit Width "$\log q$"	31	31	512	384
Plaintext Modulus	2^{30}	2^{30}	2^{40}	2^{n+1}
Key Standard Deviation "σ_{key}"	25	10	1	1
Effective Security Level "λ" (12)	173	157	120	44
Vulnerability Factor "F" (15)	0.79	N/A	3×10^{74}	2.3×10^{55}
Ctxt Size	$n \cdot \log^2 q = 123$KB	$4 \cdot n \cdot \log^2 q = 492$KB	$n \cdot \log q = 1024$KB	$n \cdot \log q = 384$KB
Key Size	$n \cdot \log q = 3.9$KB	$2 \cdot n \cdot \log q = 7.9$KB	$n \cdot \log q = 1024$KB	$n \cdot \log q = 384$KB
Evaluation Key	N/A	N/A	$n \cdot \log^2 q = 512$MB	Required but not used
Need to Disclose Circuit Depth?	No	No	No	Yes

Table VII: Performance comparison between this work and the work in [12], [18], [24]. The tuple $(n, \log q)$ is written below each scheme.

Operation	This Work (1024, 31) (CPU) (msec)	This Work (1024, 31) (GPU) (msec)	SHIELD [12] (1024, 31) (GPU) (msec)	GPU Speedup over SHIELD	Work in [18] (16384, 512) (CPU) (sec)	GPU Speedup over [18]	Work in [24] (8192, 384) (CPU) (sec)	GPU Speedup over [24]
Encrypt	23	0.16	23	143×	0.58	3625×	0.78	4875×
Decrypt	5	1	5	5×	0.55	550×	0.74	740×
Add	0.25	0.07	0.2	2.85×	0.001	14.29×	0.003	42.86×
Multiply	87.8	0.838	3.477	4.15×	5.1	6085.92×	0.24	286.4×

Table VIII: Candidate applications performance in milliseconds. The parameter k in the relational operation application represents the number of bits used to represent the numbers being compared. The tuple $(n, \log q)$ is written below each scheme.

Application	This Work (1024, 31)	Work in [24] (8192, 384)	Speedup over [24]	Work in [18] (16384, 512)	Speedup over [18]	Work in [25]	Speedup over [25]
Pearson Goodness-of-fit Test	8.452	1360	160.9×	NA	NA	NA	NA
CAAT	22.28	3630	162.9×	NA	NA	NA	NA
Predictive Analysis (32)	0.77	NA	NA	196	254.5×	NA	NA
Predictive Analysis (33)	13.69	NA	NA	$80,000$	5834.7×	NA	NA
Relational Operation	$t_1 = 2.514 \times k$	NA	NA	NA	NA	$t_2 = 30.75 \times k$	12.2×
Blood Pressure	$6 \times t_1$	NA	NA	NA	NA	$6 \times t_2$	12.2×

VII. Conclusion

We formulated, optimized, and implemented an NTRU-based variant of the HE scheme of [12], [7], [8] which achieves much slower growth of noise, and thus much better parameters than previous HE schemes. Compared to the work in [18], our GPU implementation (GM204 Maxwell architecture) acheives a speedup of 6085× in Ctxt multiplication, which represents the bottleneck for most HE schemes. Representative medical applications, namely Pearson Goodness-of-fit test [24], Cochran-Armitage test for trend (CATT) [24], predictive analysis [18], and relational operations [25] were implemented and scored speedups of 160.9×, 162.9×, 80000×, and 12.2×, respectively.

Acknowledgment

We would like to thank the authors of [29] Martin Albrecht, Shi Bai, and Léo Ducasthe for their insightful replies to our questions that helped us tune the parameters of our work to protect it against subfield lattice attacks. Financial support by NSERC is greatly acknowledged.

References

[1] C. Gentry, "Fully Homomorphic Encryption Using Ideal Lattices," in *Proceedings of the 41st Annual ACM Symposium on Theory of Computing*, ser. STOC '09, New York, NY, USA, 2009, pp. 169–178. [Online]. Available: doi.acm.org/10.1145/1536414.1536440

[2] ——, "A fully homomorphic encryption scheme," Ph.D. dissertation, Stanford University, 2009, crypto.stanford.edu/craig.

[3] J.-S. Coron, A. Mandal, D. Naccache, and M. Tibouchi, "Fully Homomorphic Encryption over the Integers with Shorter Public Keys," in *Advances in Cryptology – CRYPTO 2011*, ser. Lecture Notes in Computer Science, P. Rogaway, Ed. Springer Berlin Heidelberg, 2011, vol. 6841, pp. 487–504. [Online]. Available: dx.doi.org/10.1007/978-3-642-22792-9_28

[4] M. Dijk, C. Gentry, S. Halevi, and V. Vaikuntanathan, "Fully Homomorphic Encryption over the Integers," in *Advances in Cryptology – EUROCRYPT 2010*, ser. Lecture Notes in Computer Science,

H. Gilbert, Ed. Springer Berlin Heidelberg, 2010, vol. 6110, pp. 24–43. [Online]. Available: dx.doi.org/10.1007/978-3-642-13190-5_2

[5] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, "(Leveled) Fully Homomorphic Encryption Without Bootstrapping," in *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, ser. ITCS '12, New York, NY, USA, 2012, pp. 309–325. [Online]. Available: doi.acm.org/10.1145/2090236.2090262

[6] Z. Brakerski and V. Vaikuntanathan, "Efficient Fully Homomorphic Encryption from (Standard) LWE," in *Foundations of Computer Science (FOCS), 2011 IEEE 52nd Annual Symposium on*, 2011, pp. 97–106.

[7] C. Gentry, A. Sahai, and B. Waters, "Homomorphic Encryption from Learning with Errors: Conceptually-Simpler, Asymptotically-Faster, Attribute-Based," in *Advances in Cryptology – CRYPTO 2013*, ser. Lecture Notes in Computer Science, R. Canetti and J. Garay, Eds. Springer Berlin Heidelberg, 2013, vol. 8042, pp. 75–92. [Online]. Available: dx.doi.org/10.1007/978-3-642-40041-4_5

[8] Z. Brakerski and V. Vaikuntanathan, "Lattice-based FHE As Secure As PKE," in *Proceedings of the 5th Conference on Innovations in Theoretical Computer Science*, ser. ITCS '14, New York, NY, USA, 2014, pp. 1–12. [Online]. Available: doi.acm.org/10.1145/2554797.2554799

[9] ——, "Fully Homomorphic Encryption from Ring-LWE and Security for Key Dependent Messages," in *Advances in Cryptology – CRYPTO 2011*, ser. Lecture Notes in Computer Science, P. Rogaway, Ed. Springer Berlin Heidelberg, 2011, vol. 6841, pp. 505–524. [Online]. Available: dx.doi.org/10.1007/978-3-642-22792-9_29

[10] M. Naehrig, K. Lauter, and V. Vaikuntanathan, "Can Homomorphic Encryption Be Practical?," in *Proceedings of the 3rd ACM Workshop on Cloud Computing Security Workshop*, ser. CCSW '11, New York, NY, USA, 2011, pp. 113–124. [Online]. Available: doi.acm.org/10.1145/2046660.2046682

[11] C. Gentry, S. Halevi, and N. Smart, "Homomorphic Evaluation of the AES Circuit," in *Advances in Cryptology – CRYPTO 2012*, ser. Lecture Notes in Computer Science, R. Safavi-Naini and R. Canetti, Eds. Springer Berlin Heidelberg, 2012, vol. 7417, pp. 850–867. [Online]. Available: dx.doi.org/10.1007/978-3-642-32009-5_49

[12] A. Khedr, G. Gulak, and V. Vaikuntanathan, "SHIELD: Scalable Homomorphic Implementation of Encrypted Data-Classifiers," *IEEE Transactions on Computers*, vol. 65, no. 9, pp. 2848–2858, Sept 2016.

[13] J. Hoffstein, J. Pipher, and J. Silverman, "NTRU: A ring-based public key cryptosystem," in *Algorithmic Number Theory*, ser. Lecture Notes in Computer Science, J. Buhler, Ed. Springer Berlin Heidelberg, 1998, vol. 1423, pp. 267–288. [Online]. Available: http://dx.doi.org/10.1007/BFb0054868

[14] W. Wang, Y. Hu, L. Chen, X. Huang, and B. Sunar, "Accelerating fully homomorphic encryption using GPU," in *High Performance Extreme Computing (HPEC), 2012 IEEE Conference on*, 2012, pp. 1–5.

[15] Y. Doroz, Y. Hu, and B. Sunar, "Homomorphic AES Evaluation using NTRU," Cryptology ePrint Archive, Report 2014/039, 2014, http://eprint.iacr.org/.

[16] Y. Doroz, B. Sunar, and G. Hammouri, "Bandwidth Efficient PIR from NTRU," in *Cryptology ePrint Archive*, 2014, pp. 1–12.

[17] A. López-Alt, E. Tromer, and V. Vaikuntanathan, "On-the-fly Multiparty Computation on the Cloud via Multikey Fully Homomorphic Encryption," in *Proceedings of the Forty-fourth Annual ACM Symposium on Theory of Computing*, ser. STOC '12. New York, NY, USA: ACM, 2012, pp. 1219–1234. [Online]. Available: http://doi.acm.org/10.1145/2213977.2214086

[18] J. W. Bos, K. Lauter, and M. Naehrig, "Private predictive analysis on encrypted medical data.," in *Journal of biomedical informatics*. Elsevier Inc., 2014, pp. 234–243. [Online]. Available: http://www.ncbi.nlm.nih.gov/pubmed/24835616

[19] R. Lindner and C. Peikert, "Better Key Sizes (and Attacks) for LWE-based Encryption," in *Proceedings of the 11th International Conference on Topics in Cryptology: CT-RSA 2011*, ser. CT-RSA'11. Berlin, Heidelberg: Springer-Verlag, 2011, pp. 319–339. [Online]. Available: http://dl.acm.org/citation.cfm?id=1964621.1964651

[20] Y. Doröz and B. Sunar, "Flattening NTRU for Evaluation Key Free Homomorphic Encryption," Cryptology ePrint Archive, Report 2016/315, 2016, http://eprint.iacr.org/.

[21] S. Halevi and V. Shoup. (2013) Design and Implementation of a Homomorphic-Encryption Library. researcher.ibm.com/researcher/files/us-shaih/he-library.pdf.

[22] D. Cousins, K. Rohloff, C. Peikert, and R. Schantz, "An update on SIPHER (Scalable Implementation of Primitives for Homomorphic EncRyption) ; FPGA implementation using Simulink," in *High Performance Extreme Computing (HPEC), 2012 IEEE Conference on*, 2012, pp. 1–5.

[23] M. Yasuda, T. Shimoyama, J. Kogure, K. Yokoyama, and T. Koshiba, "Secure pattern matching using somewhat homomorphic encryption," in *Proceedings of the 2013 ACM Workshop on Cloud Computing Security Workshop*, ser. CCSW '13, New York, NY, USA, 2013, pp. 65–76. [Online]. Available: http://doi.acm.org/10.1145/2517488.2517497

[24] K. Lauter, A. Lopez-Alt, and M. Naehrig, "Private Computation on Encrypted Genomic Data," Tech. Rep. MSR-TR-2014-93, June 2014, http://research.microsoft.com/apps/pubs/default.aspx?id=219979.

[25] J. H. Cheon, M. Kim, and M. Kim, "Search-and-compute on encrypted data," in *Financial Cryptography and Data Security: FC 2015 International Workshops, BITCOIN, WAHC, and Wearable, San Juan, Puerto Rico, January 30, 2015, Revised Selected Papers*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015, pp. 142–159. [Online]. Available: http://dx.doi.org/10.1007/978-3-662-48051-9_11

[26] M. Yasuda, T. Shimoyama, J. Kogure, K. Yokoyama, and T. Koshiba, "Packed Homomorphic Encryption Based on Ideal Lattices and Its Application to Biometrics," in *Security Engineering and Intelligence Informatics*, ser. Lecture Notes in Computer Science, A. Cuzzocrea, C. Kittl, D. Simos, E. Weippl, and L. Xu, Eds. Springer Berlin Heidelberg, 2013, vol. 8128, pp. 55–74. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-40588-4_5

[27] C. Aguilar-Melchor, S. Fau, C. Fontaine, G. Gogniat, and R. Sirdey, "Recent Advances in Homomorphic Encryption: A Possible Future for Signal Processing in the Encrypted Domain," *Signal Processing Magazine, IEEE*, vol. 30, no. 2, pp. 108–117, 2013.

[28] B. Hayes, "Alice and Bob in Cipherspace," ser. American Scientist, 2012, vol. 100, no. 5, pp. 362–367.

[29] L. D. Martin Albrecht, Shi Bai, "A subfield lattice attack on overstretched NTRU assumptions: Cryptanalysis of some FHE and Graded Encoding Schemes," Cryptology ePrint Archive, Report 2016/127, 2016, http://eprint.iacr.org/2016/127.

[30] Y. Chen and P. Q. Nguyen, *BKZ 2.0: Better Lattice Security Estimates*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 1–20. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-25385-0_1

[31] J. W. Bos, K. Lauter, J. Loftus, and M. Naehrig, "Improved Security for a Ring-Based Fully Homomorphic Encryption Scheme," Cryptology ePrint Archive, Report 2013/075, 2013, http://eprint.iacr.org/.

[32] T. Lepoint and M. Naehrig, *A Comparison of the Homomorphic Encryption Schemes FV and YASHE*. Springer International Publishing, 2014, pp. 318–335. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-06734-6_20

[33] P. Kirchner and P.-A. Fouque, "Comparison between Subfield and Straightforward Attacks on NTRU," Cryptology ePrint Archive, Report 2016/717, 2016, http://eprint.iacr.org/2016/717.

[34] B. Schoenmakers and P. Tuyls, "Practical Two-Party Computation Based on the Conditional Gate," in *Advances in Cryptology - ASIACRYPT 2004*, ser. Lecture Notes in Computer Science, P. Lee, Ed. Springer Berlin Heidelberg, 2004, vol. 3329, pp. 119–136. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-30539-2_10

[35] B. F. Gage, C. van Walraven, L. Pearce, R. G. Hart, P. J. Koudstaal, B. Boode, and P. Petersen, "Selecting Patients With Atrial Fibrillation for Anticoagulation: Stroke Risk Stratification in Patients Taking Aspirin," *Circulation*, vol. 110, no. 16, pp. 2287–2292, 2004. [Online]. Available: http://circ.ahajournals.org/content/110/16/2287.abstract

[36] P. W. F. Wilson, R. B. D'Agostino, D. Levy, A. M. Belanger, H. Silbershatz, and W. B. Kannel, "Prediction of Coronary Heart Disease Using Risk Factor Categories," *Circulation*, vol. 97, no. 18, pp. 1837–1847, 1998. [Online]. Available: http://circ.ahajournals.org/content/97/18/1837.abstract

[37] R. B. D'Agostino, R. S. Vasan, M. J. Pencina, P. A. Wolf, M. Cobain, J. M. Massaro, and W. B. Kannel, "General Cardiovascular Risk Profile for Use in Primary Care: The Framingham Heart Study," *Circulation*, vol. 117, no. 6, pp. 743–753, 2008. [Online]. Available: http://circ.ahajournals.org/content/117/6/743.abstract

[38] M. Chenal and Q. Tang, "On Key Recovery Attacks against Existing Somewhat Homomorphic Encryption Schemes," in *The third International Conference on Cryptology and Information Security in Latin America, Latincrypt 2014*, 2014, pp. 1–28. [Online]. Available: https://orbilu.uni.lu/handle/10993/18106

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JBHI.2017.2657458, IEEE Journal of Biomedical and Health Informatics

Alhassan Khedr received his M.Sc and B.Sc degrees from Electronics and Electrical Communications Engineering Department, Faculty of Engineering, Cairo University, Cairo, Egypt in 2008 and 2011 respectively. After graduation, he was appointed as a Teaching Assistant in Cairo University and American University of Cairo for 3 years. He received numerous awards for his excellence as a student and as a teaching assistant. He was among the team responsible for developing and fabricating CUSPARC the first fully developed Egyptian embedded processor. Alhassan joined Electronics and Computer Engineering Department at University of Toronto to pursue his PhD degree in 2011. Alhassan main research interests include algorithm optimization and VLSI implementation of high performance algorithms. He is also interested in parallel and multi/many core processor architecture design and computer arithmetic.

Dr. Glenn Gulak is a Professor in the Department of Electrical and Computer Engineering at the University of Toronto. He is a Senior Member of the IEEE and a registered Professional Engineer in the Province of Ontario. His present research interests are in the areas of algorithms, circuits, and CMOS implementations of high-performance baseband digital communication systems. He has authored or co-authored more than 150 publications in refereed journals and refereed conference proceedings. In addition, he has received numerous teaching awards for undergraduate courses taught in both the Department of Computer Science and the Department of Electrical and Computer Engineering at the University of Toronto. From Jan. 1985 to Jan. 1988 he was a Research Associate in the Information Systems Laboratory and the Computer Systems Laboratory at Stanford University. He has served on the ISSCC Signal Processing Technical Subcommittee from 1990 to 1999, ISSCC Technical Vice-Chair in 2000 and served as the Technical Program Chair for ISSCC 2001. He served on the Technology Directions Subcommittee for ISSCC from 2005 to 2008. He currently serves as the Vice-President of the Publications Committee for the IEEE Solid-State Circuits Society and a member of the IEEE PSPB.