Privacy-Preserving Chi-Squared Tests Using Homomorphic Encryption

Charlotte LeMay

May 6, 2022

1 Introduction

1.1 Motivation

In the wake of the rapid development of genome sequencing technology, genomic data has become widely available. This has allowed for medical research on large quantities of genetic data, such as Genome-Wide Association Studies (GWASs), which seek statistical to associate the incidence of traits and diseases with single-nucleotide polymorphisms (SNPs), or individual DNA base pairs which have multiple common variations in the population. GWASs have successfully identified risk factors for many conditions, including prostate cancer [GSM$^+$07], Crohn's disease [WBZ$^+$10], and Alzheimer's disease [KDW$^+$12]. To facilitate this research, many open-access genomic datasets have been made publically available, such as the 1000 genomes project [Siv08].

The potential for better ability to predict and manage disease is a source for optimism about the future. However, the easy availability of the genomes of real people raises serious privacy concerns. Various strategies exist for identifying subjects based on genomic sequence data, even when the data is anonymized [EN14]. The individuals who consented to have their genomes sequenced very likely did not consent to having their risk of developing various diseases made into publically available information. The ideal scenario would be to allow for medical research computations to be performed on genomic data in such a way that only the results of these computations can be obtained, and not the genomes themselves.

Modern cryptography provides a tool that could achieve this scenario, namely Fully Homomorphic Encryption (FHE). Broadly speaking, FHE allows for the encrypted computation of a function f with inputs $x_1, x_2, ..., x_n$. That is, a FHE scheme allows a party given encryptions $E(x_1), E(x_2), ..., E(x_n)$ to compute an encryption $E(f(x_1, x_2, ..., x_n))$, without ever learning the value of $x_1, ..., x_n$ or $f(x_1, ..., x_n)$. A scheme with this intriguing property was first developed by Gentry [Gen09], and in the past decade, many more constructions and improvements have followed. Many new constructions have been so-called levelled FHE schemes, which are more efficient than Gentry's FHE, but which evaluate a more limited class of functions, namely functions with bounded circuit depth. This survey will focus on levelled FHE schemes and their applications to a particular problem within secure GWASs, namely the evaluation of the chi^2 statistic.

1.2 Overview

We begin by defining the notion of a levelled FHE scheme, and its correctness and security properties. We also define the underlying assumption behind many efficient levelled FHE schemes, namely the Ring

Learning with Errors (RLWE) assumption. We then describe in detail the construction of three RLWE-based levelled FHE schemes, and how these schemes have been deployed in the computation of the χ^2 statistic.

2 Definitions

2.1 Homomorphic Encryption

Craig Gentry discovered a means of achieving Fully Homomorphic Encryption (FHE), or encryption in which arbitrary-depth computation can be performed on ciphertexts, and the result can still be recovered by decryption [Gen09]. This uses a method called bootstrapping, which is asymptotically fast but which, as of the present writing, introduces large constants and is inefficient for practical use.

Therefore, since we focus in this paper on a practical application, we will define the more relaxed notion of levelled homomorphic encryption (LHE), which requires a maximum computation depth to be specified in advance. We will refer to this value as the encryption level and denote it by L. Informally, we expect ciphertexts encrypted at level L to be able to be multiplied up to L times and still return a valid answer.

Formally, define the *depth* of an arithmetic circuit C to be the length of the longest chain of successive operations the circuit. For example, $((4+1)\times(2\times(3+5)))$ has depth 3. Another relevant quantity is the *multiplicative depth*, or the length of the longest chain of successive multiplications (2 in the prior example. In typical constructions of homomorphic encryption, multiplication will amplify noise much more than addition, so multiplicative depth is more important. However our definitions will focus on depth since addition can in principle still amplify noise.

We now define homomorphic encryption, roughly basing our definitions on those of [ABC$^+$15].

A *homomorphic encryption* (HE) scheme for a circuit family $\mathcal{C}$ is a collection of probabilistic polynomial-time algorithms $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt}, \mathsf{Eval})$ where:

- $\mathsf{Setup}(1^\lambda, L)$ takes a security parameter λ and starting encryption level L, and outputs model parameters pp.
- $\mathsf{KeyGen}(\mathsf{pp})$ takes the model parameters, and outputs a secret key, public key, and evaluation key $(\mathsf{sk}, \mathsf{pk}, \mathsf{evk})$.
- $\mathsf{Encrypt}(\mathsf{pk}, m)$ takes a public key and a message m, and outputs a ciphertext ct.
- $\mathsf{Decrypt}(\mathsf{sk}, \mathsf{ct})$ takes a secret key and a ciphertext ct, and outputs a message m.
- $\mathsf{Eval}(\mathsf{evk}, C, (\mathsf{ct}_1, \mathsf{ct}_2, \ldots, \mathsf{ct}_k))$ takes the evaluation key, a circuit $C \in \mathcal{C}$ with m inputs, and input ciphertexts $\mathsf{ct}_1, \ldots, \mathsf{ct}_k$, and outputs a new ciphertext ct.

Let M be the space of possible plaintexts given λ. We say that an HE scheme is *correct* if it satisfies:

- **Correct decryption:** For all $\lambda, L \in \mathbb{Z}^+$, and all messages $m \in M$, if $(\mathsf{sk}, \mathsf{pk}, \mathsf{evk}) \leftarrow \mathsf{KeyGen}(\mathsf{Setup}(1^\lambda, L))$, then
$$\Pr\left[\mathsf{Decrypt}(\mathsf{sk}, \mathsf{Encrypt}(\mathsf{pk}, m)) = m\right] = 1.$$
- **Correct evaluation:** There exists a negligible function $\mathrm{negl}(\cdot)$ such that for all $\lambda, L \in \mathbb{Z}^+$, all circuits $C \in \mathcal{C}$ of depth at most L, and all $m_1, \ldots, m_k \in M$, if $(\mathsf{sk}, \mathsf{pk}, \mathsf{evk}) \leftarrow \mathsf{KeyGen}(\mathsf{Setup}(1^\lambda, L))$, and $\mathsf{ct}_i \leftarrow \mathsf{Encrypt}(\mathsf{pk}, m_i)$, then
$$\Pr\left[\mathsf{Decrypt}(m, \mathsf{Eval}(\mathsf{evk}, C, (\mathsf{ct}_1, \ldots, \mathsf{ct}_k))) = C(m_1, \ldots, m_k)\right] \geq 1 - \mathrm{negl}(\lambda).$$

Note that the above definitions do not rule out the following trivial construction: encrypt ciphertexts with a regular public-key encryption scheme, append a circuit description at evaluation time, and then decrypt by recovering the original message and computing the circuit on it. This construction defeats the purpose of using homomorphic encryption by forcing the secret key holder to perform the computation themselves. No "computation on ciphertexts" is being done at all.

We therefore introduce the following additional requirement. We say that an HE scheme is *compact* if it satisfies:

- **Compactness:** There exists a polynomial p such that for all λ, all L, all circuits $C \in \mathcal{C}$ of depth at most L, and all $\mathsf{ct}_1, \ldots, \mathsf{ct}_k \in \mathcal{CT}_0$, if $(\mathsf{sk}, \mathsf{pk}, \mathsf{evk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$ and $\mathsf{ct} \leftarrow \mathsf{Eval}(\mathsf{evk}, C, (\mathsf{ct}_1, \ldots, \mathsf{ct}_k))$, then the length of ct is bounded by $p(\lambda)$. This bound is independent of L and of the size of C.

Note that this size bound applies to the output of the evaluation circuit. In the schemes we will describe, freshly encrypted ciphertexts will have a length proportional to the encryption level L, and will become shorter as they are evaluated on. This is allowed, so long as the length of the eventual output does not depend on L.

A *levelled homomorphic encryption* scheme is a homomorphic encryption scheme that satisfies correct decryption, correct evaluation, and compactness. We say it is a *levelled fully homomorphic encryption* if when setup with parameter L, it can evaluate on all arithmetic circuits of depth at most L.

2.2 Security

We have not so far discussed security, but of course as we are defining an encryption scheme, we must have some notion of security. The common notions of cryptographic security in the context of public key encryption are as follows:

- **CPA Security:** The multi-purpose security definition with the oracles $\mathsf{O}_1, \mathsf{O}_2$ providing no functionality whatsoever. Note that the adversary can always perform arbitrary encryptions on its own since the encryption key pk is public.

- **CCA-1 Security:** The multi-purpose security definition with oracle $\mathsf{O}_1 = \{\mathsf{Decrypt}(\mathsf{sk}, \cdot)\}$ and O_2 providing no functionality.

- **CCA-2 Security:** The multi-purpose security definition with oracles $\mathsf{O}_1 = \mathsf{O}_2 = \{\mathsf{Decrypt}(\mathsf{sk}, \cdot)\}$, and where we do not permit the adversary $\mathcal{A}_2$ to query $\mathsf{Decrypt}(\mathsf{sk}, c)$.

- *Multi-purpose security definition*: An encryption scheme $(\mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt})$ satisfies ______ security if for all polynomials p, for all λ, and for all adversaries $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$,

$$\left| \Pr[\mathsf{Game}(\mathcal{A}, \lambda, 0) \to 1] - \Pr[\mathsf{Game}(\mathcal{A}, \lambda, 1) \to 1] \right| \leq \frac{1}{p(\lambda)},$$

where $\mathsf{Game}(\mathcal{A}, \lambda, b)$ denotes the output of:

$$(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda),\ m_0, m_1, \mathsf{state} \leftarrow \mathcal{A}_1^{\mathsf{O}_1}(1^\lambda, \mathsf{pk}),$$

$$c \leftarrow \mathsf{Encrypt}(\mathsf{pk}, m_b),\ \hat{b} \leftarrow \mathcal{A}_2^{\mathsf{O}_2}(c, \mathsf{state}),\ \text{output}\hat{b}.$$

CCA-2 security is not achievable with a homomorphic encryption scheme. This is because an adversary could easily find some function f such that $f(m_0) \neq f(m_1)$, homomorphically evaluate f on the ciphertext c, and use the decryption oracle to get either $f(m_0)$ or $f(m_1)$ and thereby distinguish $b = 0$ from $b = 1$.

The best security we can therefore hope for is CCA-1 security. In fact, a scheme that uses the bootstrapping method of Gentry can also not be CCA-1 secure, since this method involves encryptions of the secret key which an adversary could decrypt. For levelled homomorphic schemes, CCA-1 security is a possibility, but the constructions we give here are only CPA secure. For a CCA-1 key recovery attack that works on the BGV scheme, see [CT14].

2.3 Ring LWE

The following is a special case of the Ring-LWE problem described in [LPR10], which will suffice for the purposes of this paper.

For $n = 2^d$, Let $\Phi_n(X)$ be the cyclotomic polynomial, that is, the unique irreducible polynomial that divides $x^n - 1$ and does not divide $x^k - 1$ for any $k < n$. Then the Ring Learning with Errors (RLWE) problem is as follows:

Definition 1 (Decision ring-LWE)**.** *Given a security parameter λ, let $n, q \in \mathbb{Z}$ be bounded by a polynomial in λ, n a power of 2, $R_q = \mathbb{Z}_q[X]/\langle\Phi_n(X)\rangle$, and χ an error distribution over R_q. Then the $\mathsf{R\text{-}LWE}_{n,q,\chi}$ problem is to distinguish the following distributions:*

- *(a, u), with $a, u \overset{R}{\leftarrow} R_q$.*
- *$(a, as + e)$, with $s \overset{R}{\leftarrow} R_q$ fixed for all samples, and $a \overset{R}{\leftarrow} R_q$ and $e \leftarrow \chi$ redrawn for each sample.*

It was shown in [LPR10] that there exists a quantum reduction to $\mathsf{R\text{-}LWE}_{n,q,\chi}$ from the approximate shortest vector problem (SVP_γ) in the worst case, with approximation factor $\gamma = \text{poly}(\lambda)/\alpha$, where $\alpha < 1$ corresponds roughly to the width of χ. For polynomial-size γ, the best known algorithms for SVP_γ are exponential [Sch87], so it is commonly assumed that average-case $\mathsf{R\text{-}LWE}_{n,q,\chi}$ is hard to solve in polynomial time.

3 Constructions

We will describe three instantiations of CPA-secure levelled fully homomorphic encryption based on RLWE, all of which have been applied to the secure GWAS problem. The first, BGV, can be considered the canonical example of ring-based levelled FHE. The second, YASHE, is another common variant that can sometimes be more efficient for small plaintext spaces. The third, CKKS, is a variant of BGV that is optimized for approximate, real-number computations.

3.1 BGV

The following is the construction of [BGV14]. We omit the details of the modulus-switching mechanism as that is not the focus of this paper, and simply describe its functionality. In essence, multiplication of ciphertexts results in a ciphertext encrypted under a larger secret key, and a special operation called $\mathsf{Refresh}$ reduces this back to the original secret key, while also reducing the size of the modulus. A fuller expanation can be found in [BGV14] (or in Problem 5 of this class's first homework set).

The construction is as follows:

- $\mathsf{Setup}(1^\lambda, L)$: Choose an integer n, moduli $Q_1 < Q_2 < \cdots < Q_L$ (with each Q_ℓ of size roughly double that of $Q_{\ell-1}$), a plaintext modulus t, and distributions χ_{err} and χ_{key} over $R = \mathbb{Z}[X]/\Phi_n(X)$. Output parameters $\mathsf{pp} = (L, n, \{Q_i\}, t, \chi_{err}, \chi_{key})$.

- $\mathsf{KeyGen}(\mathsf{pp})$: Sample $s' \xleftarrow{R} \chi_{key}$, and set $\mathbf{s} = (s', 1)$. Sample $a' \xleftarrow{R} R_q^n$ and $\mathbf{e} \xleftarrow{R} \chi_{err}$, and set $\mathbf{a} \leftarrow (a', -a's' + t\mathbf{e})$. Output $(\mathsf{pk}, \mathsf{sk}, \mathsf{evk}) \leftarrow (\mathbf{a}, \mathbf{s}, \mathsf{RefreshKeyGen}(\mathsf{pp}, \mathbf{s}))$.

- $\mathsf{Encrypt}(\mathsf{pk} = \mathbf{a}, m \in R_t)$: Sample $r \leftarrow R_2$ and $\mathbf{e}' \leftarrow \chi_{err}$. Output $\left[(0, m) + t\mathbf{e}' + \mathbf{a}r\right]_{Q_L}$.

- $\mathsf{Decrypt}(\mathsf{sk} = \mathbf{s}, \mathbf{c})$: Take in $\mathbf{c}$ encrypted with modulus Q_ℓ. Output $\left[[\mathbf{s}^T\mathbf{c}]_{Q_\ell}\right]_t$.

- $\mathsf{Eval}(\mathsf{rk}, C, \mathbf{c}_1, \ldots, \mathbf{c}_n)$: Composition based on C of:

  $\mathsf{Add}(\mathsf{rk}, \mathbf{c}_1, \mathbf{c}_2)$: Take in $\mathbf{c}_1$ and $\mathbf{c}_2$ encrypted with modulus Q_ℓ. Set $\mathbf{c}_+ \leftarrow \mathbf{c}_1 + \mathbf{c}_2$. Depending on the level of noise growth, output either $\mathbf{c}_+$ or $\mathsf{Refresh}(\mathsf{rk}, \mathbf{c}_+, Q_\ell, Q_{\ell-1})$.

  $\mathsf{Mult}(\mathsf{rk}, \mathbf{c}_1, \mathbf{c}_2)$: Take in $\mathbf{c}_1$ and $\mathbf{c}_2$ encrypted with modulus Q_ℓ. Set $\mathbf{c}_\times$ to be the coefficients of the quadratic terms of the product of $\mathbf{c}_1$ and $\mathbf{c}_2$ considered as linear polynomials in the secret key entries. Output $\mathsf{Refresh}(\mathsf{rk}, \mathbf{c}_\times, Q_\ell, Q_{\ell-1})$.

- $\mathsf{RefreshKeyGen}(\mathsf{pp}, \mathsf{sk})$: Output a refresh key rk.

- $\mathsf{Refresh}(\mathsf{rk}, \mathbf{c}, Q_\ell, Q_{\ell-1})$: Output a new ciphertext encoding the same message as $\mathbf{c}$, under the original secret key sk from key generation, and modulus $Q_{\ell-1}$.

The BGV scheme is characterized by ciphertexts which are pairs polynomials in R, and by the modulus switching strategy. A message with $n \log t$ bits is encoded at level ℓ as a vector with $2n \log Q_\ell$ bits. Assuming our final modulus Q_1 is of size close to t, $2n \log Q_\ell \approx 2n\ell \log t$, so the encryption rate of the scheme (for fresh encryption where $\ell = L$) is approximately $\frac{1}{2L}$.

The key issue for the efficiency of BGV is the Refresh step. Evaluation would run faster if this step were not needed after every level of multiplication.

3.2 YASHE

YASHE (Yet Another Somewhat Homomorphic Encryption scheme) [BLLN13] has single polynomial ciphertexts rather than pairs of polynomials, and replaces the Refresh step with a key-switching step that does not change the modulus.

The construction is as follows:

- $\mathsf{Setup}(1^\lambda, L)$: Choose an integer n, a ciphertext modulus Q and plaintext modulus t, and distributions χ_{key}, χ_{err} over $R = \mathbb{Z}[X]/\Phi_n(X)$. Output $\mathsf{pp} = (L, n, Q, t, \chi_{err}, \chi_{key})$.

- $\mathsf{KeyGen}(\mathsf{pp})$: Sample $s', a' \leftarrow \chi_{key}$, and set $s \leftarrow ts' + 1$, resampling s' if s is not invertible. Set $a \leftarrow \left[ta's^{-1}\right]_Q$. Output $(\mathsf{pk}, \mathsf{sk}, \mathsf{evk}) = (a, s, \mathsf{SwitchKeyGen}(\mathsf{pp}, s))$.

- $\mathsf{Encrypt}(\mathsf{pk} = a, m \in R_t)$: Sample $e, r \leftarrow \chi_{err}$, and output $\mathbf{c} \leftarrow \left[\lfloor \frac{Q}{t} \rfloor \cdot m + e + ar\right]_Q \in R_Q$.

- $\mathsf{Decrypt}(\mathsf{sk} = s, \mathbf{c})$: Output $m \leftarrow \left[\left\lfloor \frac{t}{Q} \cdot [s \cdot \mathbf{c}]_Q \right\rceil\right]_t$.

- $\mathsf{Eval}(\mathsf{evk}, C, \mathbf{c}_1, ..., \mathbf{c}_n)$: Composition based on C of:

  $\mathsf{Add}(\mathbf{c}_1, \mathbf{c}_2)$: Output $\mathbf{c}_+ = [\mathbf{c}_1 + \mathbf{c}_2]_Q$.

  $\mathsf{Mult}(\mathsf{evk}, \mathbf{c}_1, \mathbf{c}_2)$: Output $\mathbf{c}_\times = \mathsf{SwitchKey}\left(\mathsf{evk}, \left[\left\lfloor \frac{t}{Q} \cdot \mathbf{c}_1 \cdot \mathbf{c}_2 \right\rceil\right]_t\right)$.
- $\mathsf{SwitchKeyGen}(\mathsf{pp}, \mathsf{sk})$: Output a switching key evk.
- $\mathsf{KeySwitch}(\mathsf{evk}, \mathbf{c}_1)$: If evk was constructed as $\mathsf{SwitchKeyGen}(\mathsf{sk}_1, \mathsf{sk}_2)$ and $\mathsf{Decrypt}(\mathsf{sk}_1, \mathbf{c}_1) = m$, output $\mathbf{c}_2$ such that $\mathsf{Decrypt}(\mathsf{sk}_2, \mathbf{c}_2) = m$, and $\mathbf{c}_2$ has a lower encryption level corresponding to larger noise.

The YASHE scheme encrypts a message in R_t with $n \log t$ bits as a ciphertext in R_Q with $n \log Q$ bits. The encryption rate is therefore $\frac{\log t}{\log Q}$, which depends on the setting of the parameter Q. Q is usually set to be on the order of $t \cdot 2^{\mathrm{poly}(\lambda)}$, so we expect an approximately $1/\mathrm{poly}(\lambda)$ encryption rate. How the encryption rate and evaluation efficiency compare with BGV depends on the plaintext modulus, with BGV slightly more efficient for large t, and YASHE slightly more efficient for small t [CS16].

3.3 CKKS

Cheon, Kim, Kim, and Song constructed a variant of BGV that is optimized for arithmetic on real numbers. [CKKS17] This scheme is important as our intended application, computing χ^2, is really best expressed in the setting of real-number arithmetic rather than integer arithmetic.

Since in real-number arithmetic, we are generally only interested in the most significant digits of the computation, the CKKS construction is much looser with error. Rather than using number theory and modular arithmetic to make the error vanish after decryption, decryption simply yields the message plus some error e, and the bits affected by the error are discarded.

In place of

The construction is as follows:

- $\mathsf{Setup}(1^\lambda, L)$: Choose an integer n, ciphertext moduli $Q_1, ..., Q_L$ (usuallly set as $Q_\ell = 2^\ell$), an integer P and distributions χ_{err}, χ_{key}, and χ_{enc} over $R = \mathbb{Z}[X]/\Phi_n(X)$. Output $\mathsf{pp} = (L, n, \{Q_i\}, P, \chi_{err}, \chi_{key}, \chi_{enc})$.
- $\mathsf{KeyGen}(\mathsf{pp})$: Sample $s' \leftarrow \chi_{key}$, $a' \xleftarrow{\mathrm{R}} R_{Q_L}$, and $e \leftarrow \chi_{err}$. Set $\mathbf{s} \leftarrow (1, s')$ and $\mathbf{a} \leftarrow (b', a')$, where $b' \leftarrow [-a's' + e]_{Q_L}$. Sample $a' \xleftarrow{\mathrm{R}} R_L$, and $e' \leftarrow \chi_{err}$ and set $\mathbf{v} \leftarrow (b'', a'')$ where $b' \leftarrow \left[-a's + e' + Ps^2\right]_{P \cdot Q_L}$. Output $(\mathsf{pk}, \mathsf{sk}, \mathsf{evk}) \leftarrow (\mathbf{a}, \mathbf{s}, \mathbf{v})$.
- $\mathsf{Encrypt}(\mathsf{pk} = \mathbf{a}, m \in \mathbb{R}[X]/\Phi_n(X))$: Sample $r \leftarrow \chi_{enc}$ and $\mathbf{e} \leftarrow \chi^2_{err}$, and output $\mathbf{c} \leftarrow [(m, 0) + \mathbf{e} + \mathbf{a}r]_{Q_L}$.
- $\mathsf{Decrypt}(\mathsf{sk} = \mathbf{s}, \mathbf{c})$: Take in $\mathbf{c}$ encrypted at level ℓ, and output $\left[\mathbf{s}^T \mathbf{c}\right]_{Q_\ell}$.
- $\mathsf{Eval}(\mathsf{evk}, C, \mathbf{c}_1, ..., \mathbf{c}_n)$: Composition based on C of:

  $\mathsf{Add}(\mathbf{c}_1, \mathbf{c}_2)$: Take in $\mathbf{c}_1$ and $\mathbf{c}_2$ encrypted at level ℓ, and ouput $\mathbf{c}_+ \leftarrow [\mathbf{c}_1 + \mathbf{c}_2]_{Q_\ell}$.

  $\mathsf{Mult}(\mathsf{evk} = \mathbf{v}, \mathbf{c}_1, \mathbf{c}_2)$: Ouput $\mathbf{c}_\times \leftarrow \left[(d_0, d_1) + \left\lfloor P^{-1} \cdot d_2 \cdot \mathbf{v} \right\rceil\right]_{Q_\ell}$, where $(d_0, d_1, d_2) = (b_1 b_2, a_1 b_2 + a_2 b_1, a_1 a_2)$.
- $\mathsf{Rescale}(\ell \to \ell', \mathbf{c})$: Given $\ell, \ell' \in [L]$ and $\mathbf{c} \in R^2_\ell$, output $\mathbf{c}' \leftarrow \left[\left\lfloor \frac{Q_{\ell'}}{Q_\ell} \cdot \mathbf{c} \right\rceil\right]_{Q_{\ell'}}$.

The key addition of this construction to BGV is its encoding of non-integral polynomials, and its the rescaling procedure, which truncates off the least significant digits of the encrypted value, and stores the new value in the smaller modulus under smaller precision. This prevents noise from building up during the course of the computation. Rescaling can be performed to truncate noise whenever it is needed during evaluation, with its use varying with the application.

Because of the non-integral nature of plaintexts, encryption rate is more difficult to evaluate than for the previous constructions. It is a close relative of BGV, but depending on the desired precision one can pack more or less information into its ciphertexts.

4 Applications

4.1 χ^2 Tests for GWASs

Each of the three constructions above has been used as a model for homomorphically evaluating the chi-squared statistic. We will begin by giving a mathematical formulation of the χ^2 test for a GWAS study, then describe the specifics of its implementation.

In the setting of a GWAS, we assume a collection of Single Nucleotide Variants (SNVs), which are individual DNA locuses in which members of the population have one of two possible alleles, A and B (each representing either the nucleotides A, G, T, or C, or possibly $\perp$ indicating that the locus is deleted in this variation). Since human beings are diploid and have two copies of all (non-sex-linked) chromosomes, each individual will have either the genotype AA, AB, or BB for each locus.

Suppose we are focused on a particular SNV, and we sample the genomes of N individuals, of which some have a particular disease (the cases), and the rest do not (the control). Then by the representation $AA \to 0$, $AB \to 1$, $BB \to 2$, we can represent an individual's genome by a value in $\{0,1,2\}$. The participant genomes, then, can be represented by a vector $\mathbf{s} \in \{0,1,2\}^N$. The i-th entry of $\mathbf{s}$, s_i, corresponds to the number of copies person i has of the A allele at locus j. Let $\mathbf{y} \in \{0,1\}^N$ be a vector corresponding to the disease status of a person ($y_i = 1$ if person i have the disease, and $y_i = 0$ if they do not).

Define the following allele counts (table adapted from [BGPG20]):

	A	B	Total
Control	n_{00}	n_{01}	r_0
Cases	n_{10}	n_{11}	r_1
Total	c_0	c_1	$2N$

We can compute n_{11} by $\mathbf{y}^T\mathbf{s}$, c_1 by $(1,1,...,1)^T\mathbf{s}$, and r_1 by $(1,1,...1)^T\mathbf{y}$. As we will see, these values are sufficient to compute the χ^2 statistic.

The χ^2 statistic at SNV j measures the likelihood of the results of the study, under the null hypothesis that disease status and genotype at SNV j are uncorrelated. It is computed as follows:

$$\chi^2 = \sum_{i\in\{0,1\}} \sum_{j\in\{0,1\}} \frac{\left(n_{ij} - c_j \cdot \frac{r_i}{2N}\right)^2}{c_j \cdot \frac{r_i}{2N}}.$$

This expression can be simplified into the following form:

$$\chi^2 = \frac{2N \cdot (2Nn_{11} - r_1c_1)^2}{r_1(2N - r_1)\cdot c_1(2N - c_1)}. \tag{1}$$

This is the form of the expression used by [BGPG20]. To get the form used by [KL15], we must assume that $r_0 = r_1 = N$, since in their setting, we assume separate collections of case and control data of equal size. Under this assumption,

$$\chi^2 = \frac{2N \cdot (n_{10} - n_{11})^2}{(n_{10} + n_{11}) \cdot (2N - (n_{10} + n_{11}))}. \tag{2}$$

4.2 χ^2 Test with BGV and YASHE

Kim and Lauter [KL15] have applied both BGV and YASHE to homomorphically evaluating the χ^2 statistic by Eq. (2).

Let M be the number of SNV locuses. The input data needed to compute χ^2 consists of $\mathbf{y} \in \{0,1\}^N$, the vector indicating the disease status of each patient, and $\mathbf{S} \in \{0,1,2\}^{N \times M}$, the matrix whose (i, j) entry is the number of copies of allele A participant i has at locus j.

Since the expression for χ^2 is a fraction, and none of the HE schemes described so far have supported a division operation, the schemes described will compute the numerator and denominator of the expression separately, and allow the decryptor to decrypt at the end. Therefore the encrypted inputs to HE will $\mathbf{y}$ and $\mathbf{S}$, and the encrypted outputs will be the numerator and denominator of χ^2. Once decrypted, these will yield two vectors, χ^2_{num} and χ^2_{den}, each of length M. Dividing corresponding entries will yield the χ^2 value at each SNV.

However, in the work of [KL15], the disease status vector $\mathbf{y}$ is not encrypted – rather it is implicit, since they assume a separation between the encrypted case and control data. That is, in their conception, the algorithm works with two separate lists of encrypted genotypes, one corresponding to disease cases and one corresponding to the control group. We can treat this as equivalent to the algorithm taking $\mathbf{y}$ in the clear. This is a laxer security requirement than [BGPG20] will assume below, and if the aggregator of genotype ciphertexts knows the identity of the message senders, it will learn information about their the disease status. Once could imagine scenarios in which this would be undesirable, such as if the aggregator is in a position to sell this information to insurance companies. This potential issue will be fixed in the second construction.

[KL15] also do not directly compute χ^2_{num} and χ^2_{den}, but instead only $\mathbf{n}_{01} + \mathbf{n}_{11}$ and $\mathbf{n}_{01} - \mathbf{n}_{11}$, and leave the rest of Eq. (2) to the decryptor, so this instead is the output value.

Below are shown the Encode functions used to represent rows of $\mathbf{S}_i$ (that is, individual genomes) as messages compatible with the corresponding HE Encrypt function. The encryption of $\mathbf{S}$ will be the list consisting of the encryption of each row's encoding. This method of encoding allows each individual participant to encrypt their genome separately, without sharing their data.

For BGV, we assume that the plaintext modulus $[\Phi_m(X)]_t$ factors as $f_1 f_2 \cdots f_\ell$, where the f_i are irreducible polynomials, and $\ell \geq M$. Then $\mathsf{CRT}(r_1, r_2, ..., r_\ell)$ indicates the unique polynomial in R_t whose residue modulo f_i is r_i for $i \in [\ell]$. The most important feature of this CRT or residue number system representation is that the multiplicative and additive properties of the representation are the same as for the vector of residues treated component-wise, so we pack ℓ pieces of information into a single ciphertext.

- $\mathsf{BGV.Encode}(\mathbf{S}_i \in \{0,1,2\}^M)$: On input a row $\mathbf{S}_i$ of $\mathbf{S}$, append zeroes to $\mathbf{S}_i$ to make $\mathbf{S}'_i \in \{0,1,2\}^\ell$, and output $m_i \leftarrow \mathsf{CRT}(\mathbf{S}'_i)$.

For YASHE, we allow the ability to pack multiple participant genomes within one ciphertext. Let $n' = \lfloor \frac{n}{M} \rfloor$, where $n = m/2$ is the degree of the modulus $\Phi_m(X)$ of R. We may pack n' participants into one ciphertext, which gives efficiency gains for $n' > 1$.

- $\mathsf{YASHE.Encode}(\mathbf{S}_{i\cdot n'},\ldots,\mathbf{S}_{i\cdot n'+n'-1} \in \{0,1,2\}^M)$: On input n' rows $\mathbf{S}_{i\cdot n'}\ldots,\mathbf{S}_{i\cdot n'+n'-1}$ of $\mathbf{S}$, output

  $$m_i \leftarrow \sum_{0\le b<n'} \sum_{1\le j\le M} \mathbf{S}_{i\cdot n'+b,j} X^{b\cdot M+(j-1)} \in R_t,$$

  or simply the polynomial whose coefficients are the entries of given rows.

This type of encoding works because the polynomial addition is performed by adding the individual components, and Kim and Lauter's method only uses homomorphic addition, not multiplication. (However they avoid multiplication by making the decryptor perform the necessary multiplications after decryption.)

Let $n' = N$ in the case of BGV. Algorithm 1 shows how Kim and Lauter compute Eq. (2). We distinguish ciphertexts from unencrypted values by putting $[\cdot]$ around ciphertexts.

Algorithm 1 Computing χ^2 with BGV or YASHE

Input: $\mathbf{y}$, $[\mathbf{S}]$

Output: $[\mathbf{n}_{01}+\mathbf{n}_{11}] = \left\{[\mathbf{n}_{01j}+\mathbf{n}_{11j}]\right\}_{j\in[M]}, [\mathbf{n}_{01}-\mathbf{n}_{11}] = \left\{[\mathbf{n}_{01j}-\mathbf{n}_{11j}]\right\}_{j\in[M]}$

```
1: $\mathbf{u}_{n'} \leftarrow \{1\}_{i=1}^{n'} \in R^N$
2: **for** $j \in [M]$ **do**
3:     $[\mathbf{n}_{01j}] \leftarrow (\mathbf{u}_{n'} - \mathbf{y})^T [\mathbf{S}_j]$
4:     $[\mathbf{n}_{11j}] \leftarrow \mathbf{y}^T [\mathbf{S}_j]$
5:     $[\mathbf{n}_{01j}+\mathbf{n}_{11j}] \leftarrow [\mathbf{n}_{01j}] + [\mathbf{n}_{11j}]$
6:     $[\mathbf{n}_{01j}-\mathbf{n}_{11j}] \leftarrow [\mathbf{n}_{01j}] - [\mathbf{n}_{11j}]$
7: **end for**
```

This algorithm requires on the order of Mn' additions, with steps 3 and 4 being the most significant contribution. It requires only a multiplicative depth of 1 (in steps 3 and 4). It never requires multiplying two encrypted values – homomorphic multiplication is not actually required! However, this is an artifact of the offloading of all multiplicative work onto the decryptor, which seems in some sense to defeat the purpose of homomorphic encryption. It allows a potential adversary to learn the sum and difference of the values $\mathbf{n}_{01}$ and $\mathbf{n}_{11}$, from which can be computed these values themselves. These are the counts for each allele in the case group, and while leaking simple counts may not seem at first to be an issue, it could become problematic in a scenario where, for example, it might leak ethnic information about the group studied.

4.3 χ^2 Test with CKKS

The work of [BGPG20] uses a construction of HE that is essentially the same as that of [CKKS17], with a few optimizations made for better efficiency in the specific low-depth setting of the GWAS. The primary modifications are:

- Instead of using moduli of the form $Q_\ell = 2^\ell$, they use moduli of the form $Q_\ell = \prod_{i=1}^{\ell} q_i$, where $\{q_1,\ldots q_L\}$ are primes of similar size.
- Multiplication and key switching now involve the residue number system (RNS) – decomposing ciphertexts into their residues modulo the q_i, performing those operations on the residues, and recombining the residues. This requires on the order of ℓ^2 number-theoretic transforms, but for low-depth computation (where ℓ will be small), it gives an efficiency improvement.

The encoding method used is similar to the BGV method shown above, using RNS over the polynomial ring (as opposed to RNS over the integers), to encode both **y** and **S** as ciphertexts.

Algorithm 2 shows the method used by [BGPG20] to compute χ^2 using CKKS. After decryption, the final computation of χ^2 will be found by multiplying $2N$ with the ratio of the numerator and denominator at each index.

Algorithm 2 Computing χ^2 with CKKS

```
Input: $[\mathbf{y}]$, $[\mathbf{S}]$
Output: $[\chi^2_{num}] = \left\{\left[\chi^2_{j,num}\right]\right\}_{j\in[M]}, [\chi^2_{den}] = \left\{\left[\chi^2_{j,num}\right]\right\}_{j\in[M]}$
 1: $\mathbf{u}_M \leftarrow \{1\}_{i=1}^M \in R^M$, $\mathbf{u}_N \leftarrow \{1\}_{i=1}^N \in R^N$
 2: $\gamma \leftarrow 0.1 \cdot N^{-2}$
 3: $\mathbf{d} \leftarrow 2N \cdot \mathbf{u}_M$
 4: $\mathbf{d}^* \leftarrow \gamma \cdot \mathbf{d}$
 5: $[\mathbf{r}_1] \leftarrow 2 \cdot [\mathbf{y}]^T \mathbf{u}_N$    (Computed with binary tree addition)
 6: $[\mathbf{r}_1^*] \leftarrow \gamma \cdot \mathbf{r}_1$
 7: for $j \in [M]$ do
 8:     $[\mathbf{n}_{11}] \leftarrow [\mathbf{y}]^T [\mathbf{S}_j]$
 9:     $[\mathbf{c}_1] \leftarrow \mathbf{u}_N^T [\mathbf{S}_j]$
10:     $[\mathbf{c}_1^*] \leftarrow \gamma \cdot [\mathbf{c}_1]$
11:     $\left[\chi^2_{j,num}\right] = \left([\mathbf{n}_{11}] \star \mathbf{d}^* - [\mathbf{c}_1] \star [\mathbf{r}_1^*]\right)^2$
12:     $\left[\chi^2_{j,den}\right] = [\mathbf{c}_1] \star \left(\mathbf{d}^* - [\mathbf{c}_1^*]\right) \star [\mathbf{r}_1] \star \left(\mathbf{d}^* - [\mathbf{r}_1^*]\right)$
13: end for
```

The value γ is introduced in order to keep the magnitude of the numerator and denominator small enough that the decrypted message fits inside a singe RNS modulus, to reduce the number of such moduli needed. The multiplicative depth in this scheme is 3 (the value stored in step 11 depends on the square, which depends on the multiplication of $[\mathbf{n}_{11}]$ and $\mathbf{d}^*$, which depends on the multiplication in step 8), so 4 moduli are used.

This algorithm requires on the order of MN additions and MN homomorphic multiplications, with step 8 and 9 dominating the running time (the products in steps 8 and 9 are significantly slower than the $\star$ products since it is the SIMD optimization cannot be used). The product in step 8 in particular is slower than its equivalent in Algorithm 1, since once $[\mathbf{y}]$ is encrypted, it is no longer a 0/1 vector, and nontrivial multiplications must be computed.

In comparison to [KL15], the scheme of [BGPG20] requires greater multiplicative complexity. However, this is because [BGPG20] delegates a great deal of computation to the unencrypted steps, both separating the case and control groups beforehand (or equivalently leaving **y** unencrypted), and requiring the decryptor to perform multiplications with the decrypted data. [KL15] leaves only one division and multiplication by $2N$ (for each entry) to after decryption, and so it is closer to being in line with the spirit of homomorphic encryption.

The choice of the CKKS construction for use in Algorithm 2 is important because CKKS supports real-number mutliplication, rather than just integer multiplication. This allows the multiplicative factor γ to be introduced, which in turn allows the number of RNS moduli needed to be very small, which reduces ciphertext size and computation time. Thus CKKS allows for secure and fast GWAS computations – [BGPG20] reports that a study of $N = 100,000$ individuals and $M = 500,000$ SNVs ran in 5.6 hours. This is a potential demonstration of the capability of HE to be implemented practically for real-world tasks.

References

[ABC$^+$15] Frederik Armknecht, Colin Boyd, Christopher Carr, Kristian Gjøsteen, Angela Jäschke, Christian A Reuter, and Martin Strand. A Guide to Fully Homomorphic Encryption. *Cryptology ePrint Archive*, 2015.

[BGPG20] Marcelo Blatt, Alexander Gusev, Yuriy Polyakov, and Shafi Goldwasser. Secure Large-scale Genome-wide Association Studies using Homomorphic Encryption. *Proceedings of the National Academy of Sciences*, 117(21):11608–11613, 2020.

[BGV14] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) Fully Homomorphic Encryption without Bootstrapping. *ACM Transactions on Computation Theory (TOCT)*, 6(3):1–36, 2014.

[BLLN13] Joppe W Bos, Kristin Lauter, Jake Loftus, and Michael Naehrig. Improved Security for a Ring-Based Fully Homomorphic Encryption Scheme. In *IMA International Conference on Cryptography and Coding*, pages 45–64. Springer, 2013.

[CKKS17] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. Homomorphic Encryption for Arithmetic of Approximate Numbers. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 409–437. Springer, 2017.

[CS16] Ana Costache and Nigel P Smart. Which ring based somewhat homomorphic encryption scheme is best? In *Cryptographers' Track at the RSA Conference*, pages 325–340. Springer, 2016.

[CT14] Massimo Chenal and Qiang Tang. On key recovery attacks against existing somewhat homomorphic encryption schemes. In *International Conference on Cryptology and Information Security in Latin America*, pages 239–258. Springer, 2014.

[EN14] Yaniv Erlich and Arvind Narayanan. Routes for Breaching and Protecting Genetic Privacy. *Nature Reviews Genetics*, 15(6):409–421, 2014.

[Gen09] Craig Gentry. *A Fully Homomorphic Encryption Scheme.* Stanford university, 2009.

[GSM$^+$07] Julius Gudmundsson, Patrick Sulem, Andrei Manolescu, Laufey T Amundadottir, Daniel Gudbjartsson, Agnar Helgason, Thorunn Rafnar, Jon T Bergthorsson, Bjarni A Agnarsson, Adam Baker, et al. Genome-wide Association Study Identifies a Second Prostate Cancer Susceptibility Variant at 8q24. *Nature genetics*, 39(5):631–637, 2007.

[KDW$^+$12] MI Kamboh, FY Demirci, X Wang, RL Minster, MM Carrasquillo, VS Pankratz, SG Younkin, AJ Saykin, G Jun, C Baldwin, et al. Genome-wide Association Study of alzheimer's Disease. *Translational psychiatry*, 2(5):e117–e117, 2012.

[KL15] Miran Kim and Kristin Lauter. Private Genome Analysis through Homomorphic Encryption. In *BMC medical informatics and decision making*, volume 15, pages 1–12. BioMed Central, 2015.

[LPR10] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On Ideal Lattices and Learning with Errors over Rings. In *Annual international conference on the theory and applications of cryptographic techniques*, pages 1–23. Springer, 2010.

[Sch87] Claus-Peter Schnorr. A Hierarchy of Polynomial Time Lattice Basis Reduction Algorithms. *Theoretical computer science*, 53(2-3):201–224, 1987.

[Siv08] Nayanah Siva. 1000 Genomes Project. *Nature biotechnology*, 26(3):256–257, 2008.

[WBZ$^+$10] Kai Wang, Robert Baldassano, Haitao Zhang, Hui-Qi Qu, Marcin Imielinski, Subra Kugathasan, Vito Annese, Marla Dubinsky, Jerome I Rotter, Richard K Russell, et al. Comparative Genetic Analysis of Inflammatory Bowel Disease and Type 1 Diabetes Implicates Multiple Loci with Opposite Effects. *Human molecular genetics*, 19(10):2059–2067, 2010.