Flattening NTRU for Evaluation Key Free Homomorphic Encryption

Yarkın Doröz[1] and Berk Sunar[1]

Worcester Polytechnic Institute
{ydoroz,sunar}@wpi.edu

Abstract. We propose a new FHE scheme F-NTRU that adopts the flattening technique proposed in GSW to derive an NTRU based scheme that (similar to GSW) does not require evaluation keys or key switching. Our scheme eliminates the decision small polynomial ratio (DSPR) assumption but relies only on the standard R-LWE assumption. It uses wide key distributions, and hence is immune to the Subfield Lattice Attack. In practice, our scheme achieves competitive timings compared to the existing schemes. We are able to compute a homomorphic multiplication in 5.8 msec and 17.6 msec for 5 and 30 levels, respectively, without amortization. Furthermore, our scheme features small ciphertexts, e.g. 567 KB for 30 levels, and eliminates the need for storing and managing costly evaluation keys. In addition, we present a slightly modified version of F-NTRU that is capable to support integer operations with a very large message space along with noise analysis for all cases. The assurance gained by using wide key distributions along with the message space flexibility of the scheme, i.e. bits, binary polynomials, and integers with a large message space, allows the use the proposed scheme in a wide array of applications.

Keywords: Homomorphic Encryption, Flattening, NTRU, DSPR, Subfield Lattice Attack

1 Introduction

The notion of fully homomorphic encryption (FHE) scheme stayed as an open question for a few decades since its introduction by Rivest et al. [1]. In the meantime, numerous schemes featured limited homomorphic functionality, e.g. restricted to evaluate a non-universal set of gates, or prohibitive growth in noise or ciphertext sizes restricting the evaluation depth. These schemes are referred commonly referred to as *partially* homomorphic cryptosystems. Some of these schemes and their homomorphic properties are Elgamal encryption system [2] additive homomorphism in $\mathbb{Z}_q$, Goldwasser-Micali cryptosystem [3] additive homomorphism in $\mathbb{Z}_2$, RSA cryptosystem [4] multiplicative homomorphism on $\mathbb{Z}_q$ and Paillier cryptosystem [5] additive homomorphism in $\mathbb{Z}_q$. Later, in 2009 the first working FHE scheme was constructed by Gentry [6,7]. However, the proposed scheme was lacking in performance and it was not possible to implement anything practical. For instance, its most crucial operation called *bootstrapping*, which is used to restore the noise in ciphertext to an acceptable level, was taking 30 seconds in this very first implementation. After the introduction of the first FHE scheme, we witnessed the introduction of many new FHE constructions aimed at bringing FHE closer to real-world applications and with fewer assumptions. Several integer-based constructions and learning with error (LWE)-based constructions may be found in [8,9,10] and in [11,12,13], respectively.

These recent constructions brought along an impressive array of optimization techniques that can compete with the expensive bootstrapping operation. Such one scheme, which is based on LWE, was constructed by Brakerski, Gentry and Vaikuntanathan (BGV) [14]. The scheme uses a method called *modulus switching* to mitigate noise growth in ciphertexts. By applying modulus switching at each multiplicative level, exponential noise growth is prevented thereby maintaining noise at a fixed level throughout the homomorphic evaluation levels. The scheme was implemented as a software library HElib [15] using C++. The library was used by Gentry, Halevi and Smart [16] to implement the AES circuit homomorphically. The authors also introduced SIMD techniques [17] to batch multiple messages and process homomorphic AES circuits in parallel. They achieved amortized runtime of 2 seconds for 120 parallel blocks. Later, a new technique is introduced by Brakerski [18] that is applicable to LWE schemes. It uses tensor products to decrease noise growth from quadratic to linear.

Recently, another leveled FHE scheme was presented by López-Alt, Tromer, Vaikuntanathan (LTV) in [19]. Their scheme is based on a variant of NTRU [20] constructed earlier by Stehlé and Steinfeld [21]. The scheme is capable of processing homomorphic functions for various users with each having different public-secret key pairs. The authors introduced a new key switching technique called *relinearization*. By using it alongside with modulus switching, they are capable to mitigate the noise growth and keep the growth linear in size over the levels. However, the relinearization method requires evaluation keys to operate for each level of computation. This brings a prohibitve memory requirement especially for *deep* circuit evaluations.

The NTRU variant in [21] was later modified and implemented by Bos et al. in [22]. They adopt the tensor product technique in [18] and achieved a scale-invariant scheme with limited noise growth on homomorphic operations. Also, with the use of the tensor product technique the authors managed to improve the security of the LTV scheme [19] by using much higher levels of noise and thereby remove the Decisional Small Polynomial Ratio (DSPR) assumption. Instead the scheme relies only on the standard lattice reductions as in [21]. However, as the authors also note, the YASHE scheme brings in large evaluation key and complicated key switching procedure. Specifically, the size of the evaluation keys are $\ell_{w,q}^3$ in which q is modulus, w is radix and $\ell = \lfloor \log q \rfloor$. In [22] the authors introduce a modification ($\mathsf{YASHE'}$) to their scheme to eliminate the problems of expensive tensor product calculations and large evaluation keys. However, this modification re-introduces the DSPR assumption due to increase in noise.

Another modified LTV-FHE implementation was presented by Doröz et al. in [23]. The security of their scheme depends on the DSPR and R-LWE assumptions as in [19]. Their implementation uses the relinearization and modulus switching methods as in [19] to cope with noise. They use a ring structure to create a specialized modulus, i.e. select modulus as $q = p^L$ in which p is a prime number. With the specialization the authors managed to significantly reduce the evaluation key size by creating evaluation keys only for the first level and by reusing them for to other levels simply by computing modular reduction. This reduces the size of the evaluation keys significantly. They evaluate AES homomorphically and achieved amortized runtime of 51 seconds per block.

Motivated by the complex noise management techniques, e.g. *relinearization, modulus switching, bootstrapping*, of the earlier FHE schemes Gentry, Sahai and Waters [24] proposed a new scheme based on the approximate eigenvector problem. The system uses matrix additions and multiplications which makes it asymptotically faster. At first, they define the GSW scheme as a *somewhat homomorphic scheme* since for a depth L circuit with B-bounded parameters the noise grows with a double exponential B^{2^L}. To convert the scheme into a leveled FHE, they introduce a *Flatten* operation which decomposes the ciphertext entries into bits. The secret key is also kept in a special powers of two form. With these modifications, the noise performance is improved greatly. For a depth L circuit with B-bounded secret key entries and 1-bounded (flattened) ciphertexts, the error magnitude is at most $(N+1)^L B$ for $N = \log(q)(n+1)$. However, ciphertexts still take a considerable space $\Theta(n^2 \log(q)^2)$ and as noted by GSW [24] the scheme may not be as efficient as existing leveled schemes in practice.

Applications. The advancements in FHE constructions led many researchers to experiment on practicality of various homomorphic applications. Lauter et al. investigates evaluation of homomorphic averages, standard deviations, and logistical regressions, which are underlying tools for real-word applications, in [25]. They also investigates evaluation of genomic data algorithms like Pearson Goodness-of-Fit test, the D' and r^2-measures of linkage disequilibrium, the Estimation Maximization (EM) algorithm for haplotyping, and the Cochran-Armitage Test for Trend homomorphically in [26]. Another medical application is presented by Bos et. al in [27]. They implement a cardiovascular disease prediction service homomorphically which patients send their health information in secret and an analysis is performed in the cloud server. Aside from the medical there are other FHE applications as well. Like in [28] Doröz et al. proposed a bandwidth efficient private information retrieval (PIR) scheme using the NTRU based homomorphic encryption scheme. Another study by Çetin et al. exposes the limitations of word-size arithmetic in FHE and suggests solutions [29], i.e. convergence based iterative division, comparison, zero check and thresholding algorithms. The same authors study homomorphic sorting algorithms with respect to multiplicative depth and proposes a new scheme that is more suitable for FHE implementations with low multiplicative depth; and homomorphic autocomplete, in [30] and [31], respectively. In another work, Dai et al. implement existing schemes using GPU to achieve significant acceleration. In [32] they implemented sorting and achieved 12 – 41 times speedup over [30] depending on the sorting size. In [33] they implemented the PIR scheme of [28] and achieved 13–34 times speedup depending on the database size. Lastly, Dai et al. implemented AES and Prince block ciphers in [34] and achieved 7.3 and 1.28 seconds of runtime, respectively.

The Subfield Lattice Attack. To overcome the efficiency bottleneck of FHE numerous optimizations along with customized selections of parameters have been proposed in the literature, e.g. special form lattice dimensions and moduli permitting efficient number theoretical transforms and batching, assumptions regarding the distributions of noise and parameters, etc. Indeed, these optimizations yielded significant advances in performance sizes over the years, e.g. faster evaluation with smaller evaluation key.

On the downside many of these assumptions are still open to debate from a security point of view. A very recent work by Albrecht, Bai and Ducas [35] painfully demonstrated this fact. The authors exploit the presence of a subfield to solve the NTRU problem for large moduli q and show that when the NTRU parameters are chosen poorly then the DSPR problem is not as hard as believed thereby invalidating the underlying security assumption in LTV [19,23] and YASHE' [22]. Thus, the asymptotic security of both schemes is significantly reduced. The authors suggest a more cautious choice of parameters to minimize the effect of the attack. Even with such a remedy, the lattice dimension needs to be increased to restore the projected security level. Moreover, the selection of parameters renders batching rather difficult further diminishing the performance of both schemes.

Our Contribution. In this work, we present a new leveled FHE scheme F-NTRU that is based on the Stehlé and Steinfeld variant of NTRU [21] and adopts Flattening – the noise management technique introduced in (GSW) [24]. In a nutshell, we are able to achieve the best of both implementations in [22], i.e. elimination of expensive evaluation keys and reduction of security assumptions. Specifically,

– We introduce the first NTRU based FHE implementation that uses the *Flattening* method of GSW. Our scheme uses the encryption methods of NTRU for ciphertext creation, converts them into a matrix structure and makes use of the *Flattening* noise management technique.
– Similar to the YASHE construction [22] our scheme uses a wide key distribution and hence only relies on standard lattice reductions as in [21] (no DSPR assumption). Thus, our scheme is *immune to the Subfield Lattice Attack* by Albrecht, Bai and Ducas [35].
– By employing *Flattening*, we are able to multiply ciphertexts with only linear increase in the size of the noise level. Our scheme does not use any expensive noise reduction techniques such as relinearization and does not require prohibitively large evaluation keys.
– Our construction does not require evaluation keys. In contrast, YASHE evaluation keys grow as $\tilde{\mathcal{O}}(L^4)$ where L represents the evaluation depth. This makes it impossible to use YASHE in deep evaluations.

– F-NTRU achieves significantly smaller ciphertext sizes compared to YASHE for the same multiplicative depth.
– We introduce a variant of the scheme that reduces the ciphertext size and improves the homomorphic evaluation speed at the expense of the DSPR assumption.
– Using the noise asymmetry property of our scheme, we are able to provide small parameters which results in fast homomorphic multiplications in range of milliseconds even for large multiplicative levels, e.g. 17 msec for $L = 30$.
– Finally, via a simple polynomial to integer mapping our scheme becomes able to support homomorphic *integer* arithmetic. We present noise analysis in this case as well. Featuring a very large message space, the integer version of F-NTRU is capable to support a wide range of applications where such arithmetic is required.

2 Overview of NTRU Based FHE Schemes and GSW

In this section we briefly summarize the schemes relevant to our construction. We also briefly summarize schemes that we compare against later.

2.1 Stehlé and Steinfeld's NTRU Variant

In [21] Stehlé and Steinfeld introduced a modification to the NTRU [20] scheme to make the scheme secure under the assumed quantum hardness of the standard worst-case lattice problem. The scheme fixes the ring to $R = \mathbb{Z}[x]/\langle x^n + 1\rangle$ where n is a power of 2 and chooses $q \leq \text{Poly}(n)$. Also, chooses a message space $\mathbb{Z}_p$. Using a discrete Gaussian distribution random samples f', g are chosen where the secret key becomes $f = pf' + 1$. Set public key $h = pf^{-1}g$. A message μ is encrypted by computing $c = hs + pe + \mu \mod q$ where s, e are Gaussian distribution samples. To decrypt we simply compute $\mu = c \cdot f \mod p$.

2.2 DHS Scheme

Recently, Doröz, Hu and Sunar implemented single-key variation of the multi-key LTV-FHE scheme. This scheme is introduced in 2012 by López-Alt, Tromer and Vaikuntanathan (LTV) [19] as a full-fledged, multi-key and leveled fully homomorphic encryption scheme which is based on the NTRU cryptosystem. The NTRU scheme is a public key cryptosystem that is introduced by Stehlé and Steinfeld [21]. New additions to the NTRU scheme are operations for noise and key control are relinearization and modulus switching. Another addition to the scheme is adding random noise to the public key for security.

The DHS implementation has the operational ring $R_q = \mathbb{Z}_q[x]/\langle x^n + 1\rangle$ in which q is our modulus and n is the degree of the polynomial. The scheme uses a truncated Gaussian error distribution function χ for sampling polynomials. The truncated distribution χ is B-bounded which means coefficient sizes are between range $[-B, B]$. The implementation has following four primitive functions:

- **KeyGen.** Using the security parameter λ, we create a sequence of modulus for each level as $q_i = q^{d-i}$ in which q is a prime. Later, we sample polynomials $g \in \chi$ and $f' \in \chi$ and compute secret key $f = 2f' + 1$ and public key $h = 2gf^{-1}$ in $\mathbb{Z}_{q_0}$. In the next step, we compute the evaluation keys $\zeta_\tau^{(0)}(x) = hs_\tau + 2e_\tau + 2^\tau f$ which $\{s_\tau, e_\tau\} \in \chi$ and $\tau = [0, \lfloor \log q_0 \rfloor]$. Computed evaluation keys are for level index 0. For the rest of the levels we recycle the evaluation keys using the ring structure. To use the evaluation keys for level index i, we simply compute $\zeta_\tau^{(i)}(x) = \zeta_\tau(x) \mod q_i$. This reduces the memory requirement significantly.
- **Encrypt.** In order to encrypt a message bit μ for level i, we compute $c^{(i)} = h^{(i)}s + 2e + b$ which $\{s, e\} \in \chi$ and $h^{(i)} = h^{(0)} \mod q_i$.
- **Decrypt.** In order to decrypt at level i, we compute $\mu = \lceil c^{(i)} f^{(i)} \rfloor_{q_i} \pmod 2$.
- **Evaluation.** Homomorphic evaluation requires relinearization and modulus switching after each multiplication or an addition operation. Since the additive operations do not create significant noise like multiplications, we apply the noise reduction techniques only after a multiplication. Relinearization is computed as $\tilde{c}^{(i)}(x) = \sum_\tau \zeta_\tau^{(i)}(x) \tilde{c}_\tau^{(i-1)}(x)$. The polynomials $\tilde{c}_\tau^{(i-1)}(x)$ have the form of $\tilde{c}^{(i-1)}(x) = \sum_\tau 2^\tau \tilde{c}_\tau^{(i-1)}(x)$. Relinearization is followed by modulus switching which we compute as $\tilde{c}^{(i)}(x) = \lfloor \frac{q_i}{q_{i-1}} \tilde{c}^{(i)}(x) \rceil_2$. This reduces the noise level by $\log(q_i/q_{i-1})$ bits. In modulus switching we need to match the parity bits of the messages between the old and new moduli: $\lfloor \cdot \rceil_2$.

Note that the DHS implementation uses cyclotomic polynomials $\Phi_m(x)$ to define the ring instead of $\langle x^n + 1 \rangle$. The degree of the polynomial is $n = \phi(m)$ where ϕ is the Euler's totient function. This setup allows batching of multiple messages into the same ciphertext polynomial and thereby enables parallel processing.

2.3 BLLN

The BLLN scheme was introduced by Bos et al. in [22]. BLLN is based of the scheme proposed by Stehlé and Steinfeld [21]. The authors create a scheme called YASHE by using the construction of [21] and modify it by applying the tensor product technique of [18] to curb the noise growth in multiplications. With this it becomes possible to use a high level of noise in encryptions with which the schemes becomes DSPR hard as in [21]. However, this brings large evaluation keys into the scheme and it makes difficult to use in practice: the evaluation key consists of $\ell^3 = \mathcal{O}((\log(q))^3)$ ciphertexts. To mitigate this problem, in the same reference the authors introduced another scheme called YASHE'. They discard the tensor product and decrease the size of the evaluation keys, i.e. ℓ ciphertexts. However, they have to reduce the noise levels on fresh ciphertexts which brings back the DSPR security assumption. In the following, we first give the Basic scheme and later explain YASHE and YASHE' constructions using the Basic scheme.

Basic. We set the ring as $R = \mathbb{Z}[X]/\langle x^n + 1 \rangle$, t as the plaintext modulus, χ_{err} and χ_{key} as the Gaussian distribution for sampling. The scheme has the following primitive functions:

- **ParamsGen.** For security parameter λ we create $n = n(\lambda)$, $q = q(\lambda)$, $\chi_{\text{key}} = \chi_{\text{key}}(\lambda)$ and $\chi_{\text{err}} = \chi_{\text{err}}(\lambda)$.
- **KeyGen.** We sample $f', g \in \chi_{\text{key}}$. Set secret key $f = tf' + 1$ and public key $h = tgf^{-1}$.
- **Encrypt.** To encrypt message μ, we sample $s, e \in \chi_{\text{err}}$ and compute $c = \lfloor q/t \rfloor \mu + e + hs$.
- **Decrypt.** To decrypt a message simply compute $\mu = \left\lfloor \frac{t}{q} fc \right\rceil$ where $\lfloor \cdot \rceil$ is rounding to the nearest.

There are 3 notations need explanation before we summarize the YASHE scheme. The authors use $\otimes$ for the tensor product. The notation $P_{w,q}$ is used to convert a number to an array with powers of w, i.e. $P_{w,q}(x) \to (x, xw, xw^2, \ldots, xw^{\ell_{w,q}-1})$ for $\ell_{w,q} = (\log_w q + 2)$. Last notation is $D_{w,q}$ which decomposes a value into its word sizes, i.e. $D_{w,q}(x) \to (x_0, x_1, \ldots, x_{\ell_{w,q}-1})$ where $x = \sum_{i=0}^{\ell_{w,q}-1} x_i w^i$.

YASHE. The Basic scheme explained above is used to construct a leveled FHE scheme. The primitive operations of the scheme is as follows:

- **ParamsGen.** The parameters are selected the same way as in Basic scheme.
- **KeyGen.** The public and secret key pairs are selected using the $\mathsf{Basic.KeyGen}$ routine, i.e. $h, f \leftarrow \mathsf{Basic.KeyGen}$. Sample $\mathbf{e}, \mathbf{s} \in \chi_{\text{err}}^{\ell_{w,q}^3}$. Compute the evaluation keys
$$\zeta = \left[f^{-1} P_{w,q}(D_{w,q}(f) \otimes D_{w,q}(f)) + \mathbf{e} + h\mathbf{s}\right] \in R^{\ell_{w,q}^3}.$$
- **Encrypt.** Encrypt message μ as in Basic scheme.
- **Decrypt.** Decrypt message as in Basic scheme.
- **KeySwitch.** Compute $\langle D_{w,q}(c), \zeta \rangle$ where c is a ciphertext.
- **Addition.** Addition of two ciphertexts c_1 and c_2 is $c = c_1 + c_2$.
- **Multiplication.** Multiplication of two ciphertexts is computed as
$$c = \left[\left\lfloor \frac{t}{q} P_{w,q}(c_1) \otimes P_{w,q}(c_2) \right\rceil\right]$$
and later apply $\mathsf{KeySwitch}$ to c and output.

$\mathsf{YASHE'}$. The primitive operations are as follows:

- **ParamsGen.** The parameters are selected the same way as in Basic scheme.
- **KeyGen.** The public and secret key pairs are selected using the $\mathsf{Basic.KeyGen}$ routine, i.e. $h, f \leftarrow \mathsf{Basic.KeyGen}$. Sample $\mathbf{e}, \mathbf{s} \in \chi_{\text{err}}^{\ell_{w,q}}$. Compute the evaluation keys
$$\zeta = \left[f^{-1} P_{w,q}(D_{w,q}(f) \otimes D_{w,q}(f)) + \mathbf{e} + h\mathbf{s}\right] \in R^{\ell_{w,q}}.$$
- **Encrypt.** Encrypt message μ as in Basic scheme.
- **Decrypt.** Decrypt message as in Basic scheme.

- **KeySwitch.** Compute $\langle D_{w,q}(c), \zeta \rangle$ where c is a ciphertext.
- **Addition.** Addition of two ciphertexts c_1 and c_2 is $c = c_1 + c_2$.
- **Multiplication.** Multiplication of two ciphertexts is computed as

$$c = \left[\left\lfloor \frac{t}{q} c_1 c_2 \right\rceil \right]$$

and later apply KeySwitch to c and output.

2.4 Impact of the Subfield Lattice Attack on LTV and YASHE'

On the downside many of these assumptions are still open to debate from a security point of view. A very recent work by Albrecht, Bai and Ducas [35] painfully demonstrated this fact. The authors exploit the presence of a subfield to solve the NTRU problem for large moduli q and show that when the NTRU parameters are chosen poorly it becomes possible to norm-down the NTRU public key h to a subfield yielding an easier lattice problem. Consequently, and any sufficiently good solution may be lifted to a short vector in the full NTRU lattice. The attack works when the secret key f is chosen from a narrow distribution, e.g. $||f|| \leqq \sqrt{q}$ and when the polynomial modulus is chosen such that a subfield of reasonable size exists. In this setting, Albrecht et al. show that the DSPR problem is not as hard as believed thereby invalidating the basic assumption in the LTV [19,23] and YASHE' [22] schemes. Thus, the subfield lattice attack significantly diminishes the asymptotic security of both schemes.

Both LTV and YASHE' rely on the secret key from being sampled form a narrow distribution to support even a single homomorphic multiplication. This eliminates the possibility of sampling the key from a wider distribution. The adverse effect of the attack could be mitigated, by maximizing size of the subfield as recommended in [35]. Even then, the lattice dimension and parameters need to be increased to restore the projected security level of LTV and YASHE'. Another important side effect of the Subfield Lattice Attack is that it makes the selection of parameters that support *batching* rather difficult.

2.5 GSW Scheme

Gentry, Sahai and Waters [24] proposed this new scheme based on the hardness of the approximate eigenvector problem. The construction consists of simple matrix addition and multiplication operations to perform homomorphic addition and homomorphic multiplication. The advantage of the scheme is that it eliminates the need of relinearization, storage of evaluation keys and even modulus switching. As many other homomorphic schemes the security is based on the LWE problem.

We explain the scheme in four primitive functions in below. Before the explanations we should note some of the preliminaries that is used in the primitive functions. A vector $\boldsymbol{a} = (a_0, \ldots, a_{k-1})$ is split into its bits using the function $\mathsf{BitDecomp}(\boldsymbol{a}) = (a_{0,0}, \ldots a_{0,\ell-1}, \ldots, a_{k-1,0} \ldots, a_{k-1,\ell-1})$. We are able to reconstruct the elements from the bit representations using the inverse of the $\mathsf{BitDecomp}$ as $\mathsf{BitDecomp}^{-1}(\boldsymbol{a}) = (\sum 2^j a_{0,j}, \dots, \sum 2^j a_{k-1,j})$. The most important function that the scheme uses is *flattening*. It keeps the ciphertexts bounded so that the noise increase after multiplicative operations are limited. We simply evaluate it as $\mathsf{Flatten}(\boldsymbol{a}) = \mathsf{BitDecomp}(\mathsf{BitDecomp}^{-1}(\boldsymbol{a}))$. The last function is $\mathsf{Powersof2}(\boldsymbol{a}) = (a_0, 2a_0, \dots, 2^{\ell-1}a_0, \dots, a_k, 2a_k, \dots, 2^{\ell-1}a_k)$ which multiplies the vector elements with powers of two. Using these functions, the encryption scheme is defined with the following primitives:

- **Setup.** We select λ as the security parameter and L as multiplicative depth. Then compute lattice dimension $n = n(\lambda, L)$, error distribution $\chi = \chi(\lambda, L)$, parameter $m = m(\lambda, L) = \mathcal{O}(n \log q)$. Also, set $\ell = \lceil \log q \rceil + 1$ and $N = (n+1)\ell$.
- **KeyGen.** We sample $\boldsymbol{t} \leftarrow \mathbb{Z}_q^n$ and compute $\boldsymbol{s} \leftarrow (1, -t_1, -t_2, \dots, -t_n)$. Then, set the secret key $\boldsymbol{v} = \mathsf{Powersof2}(\boldsymbol{s})$. The public key matrix is computed by first generating uniform matrix $B \leftarrow \mathbb{Z}_q^{m \times n}$ and error vector $\boldsymbol{e} \leftarrow \chi^m$. We set $(n+1)$ column matrix A having $\boldsymbol{b} = B \cdot \boldsymbol{t} + \boldsymbol{e}$ as the first column and $\boldsymbol{b}$ in rest of the columns as the public key.
- **Encrypt.** A message μ is encrypted by simply computing $C = \mathsf{Flatten}(\mu I_N + \mathsf{BitDecomp}(R \cdot A)) \in \mathbb{Z}_q^{N \times N}$. In the equation R is selected as a uniform matrix $R \in \{0,1\}^{N \times m}$.
- **Decrypt.** Select a row of the matrix, i.e. C_i as the i-th row of matrix C. Compute $x_i \leftarrow \langle C_i, \boldsymbol{v} \rangle$ and the message as $\mu' = \lfloor x_i / v_i \rceil$.

The beauty of the GSW scheme is that straightforward addition and multiplication of ciphertext matrices suffice to compute homomorphic additions and multiplications. First, lets observe that the construction holds the property $C \cdot \boldsymbol{v} = \mu \cdot \boldsymbol{v} + \boldsymbol{e}$, since secret key $\boldsymbol{v}$ is the approximate eigenvector related to the ciphertext. Therefore, adding ciphertext matrices has the effect of adding the corresponding eigenvalues (messages): $(C_1 + C_2) \cdot \mathbf{v} = (\mu_1 + \mu_2) \cdot \mathbf{v} + (\mathbf{e}_1 + \mathbf{e}_2)$. Similarly the matrix product of the ciphertexts (in any order) multiplies the eigenvalues: $C_1 \cdot C_2 \cdot \mathbf{v} = C_1(\mu_2 \cdot \mathbf{v} + \mathbf{e}_2) = \mu_2(\mu_1 \cdot \mathbf{v} + \mathbf{e}_1) + C_1 \cdot \mathbf{e}_2 = \mu_1\mu_2 \cdot \mathbf{v} + \mathbf{e}$.

3 Our proposal: F-NTRU Scheme

Here we propose a new scheme called F-NTRU that shares the goals of the GSW construction [24], i.e. no evaluation keys, no expensive relinearization operations, no modulus switching and simple homomorphic additions and multiplications. To this end we adopt the flattening approach of [24] and apply it to the NTRU variant, i.e. NTRU', by Stehlé and Steinfeld [21].

Preliminaries. We work in $R_q = \mathbb{Z}_q[x]/\langle x^n + 1 \rangle$. We adapt two functions from [24] to work in the polynomial setting as follows:

- **Bit-Decomposition:** The $\mathsf{BitDecomp}$ function takes a ciphertext polynomial $c(x)$ and splits them into binary polynomials and forms a vector as:

$$\boldsymbol{c}(x) = \mathsf{BitDecomp}(c(x)) = [c_{\ell-1}(x) c_{\ell-2}(x) \dots c_0(x)].$$

Here a polynomial $c_i(x)$ with index i represents the binary polynomial that is formed using the i^{th} bit index of the coefficients of $c(x)$. One may easily reconstruct the ciphertext $c(x)$ by simply computing:

$$c(x) = \sum_{i=0}^{\ell-1} 2^i \cdot c_i(x) \in R_q.$$

Note that when we are computing $\mathsf{BitDecomp}^{-1}$, the elements in the vector do not necessarily have to be binary polynomials. It is possible that polynomials can contain coefficients that are not bits.

– **Flatten:** When we are performing arithmetic operations, i.e. addition and multiplication, in our scheme, the elements of a flattened ciphertext vector loose their binary form. These extra bits in the coefficients of the polynomials cause additional noise in subsequent arithmetic operations. To prevent this, we use Flatten. Flatten restructures all the elements of the ciphertext vector into binary polynomials. We evaluate the Flatten operation as follows:

$$\mathsf{Flatten}\left(\boldsymbol{c}(x)\right) = \mathsf{BitDecomp}\left(\mathsf{BitDecomp}^{-1}\left(\boldsymbol{c}(x)\right)\right).$$

Here in the equation $\mathsf{BitDecomp}^{-1}$ converts the ciphertext vector into a full ciphertext polynomial in R_q. Later, using BitDecomp we convert the ciphertext into a binary polynomial vector again. Basically, this method carries over the extra bits of an element in the vector from least significant to most significant and performs a modular reduction using q to prevent overflow in the overall scheme.

The F-NTRU Scheme. The primitive operations of F-NTRU are defined as follows:

– **KeyGen:** We use the key generation method of NTRU' to select our parameters. For a security parameter λ, we choose our message modulus as 2, modulus $q = q(\lambda)$, polynomial degree $n = n(\lambda)$ where n is power of 2. Also we set Gaussian distributions $\chi_{\text{err}} = \chi_{\text{err}}(\lambda)$ and $\chi_{\text{key}} = \chi_{\text{key}}(\lambda)$. Sample $g, f' \in \chi_{\text{key}}$ and set public key $h = 2gf^{-1}$ and secret key $f = 2f' + 1$.

– **Encrypt:** In order to encrypt a message μ, we create a vector with length $\ell = \log q$. Later, we fill the elements with encryptions of zeros using the NTRU' scheme:

$$\begin{aligned} \boldsymbol{c} &= \{\mathsf{Enc}_{\ell-1}(0), \mathsf{Enc}_{\ell-2}(0), \ldots, \mathsf{Enc}_0(0)\} \\ &= \{c_{\ell-1}, c_{\ell-2}, \ldots, c_0\}, \end{aligned}$$

where $\mathsf{Enc}_i(0) = hs_i + 2e_i + 0$. We call this the *ciphertext vector*. We take the transpose of the vector to list the elements in row and apply BitDecomp to the ciphertexts to turn the vector into a $\ell \times \ell$ matrix:

$$c = \mathsf{BitDecomp}(\boldsymbol{c}^\top)$$

$C' = \mathsf{Flatten}(C + \tilde{C}) =$

$$\begin{bmatrix} c_{(3,3)} + \tilde{c}_{(3,3)} + \mu + \tilde{\mu} & c_{(3,2)} + \tilde{c}_{(3,2)} & c_{(3,1)} + \tilde{c}_{(3,1)} & c_{(3,0)} + \tilde{c}_{(3,0)} \\ c_{(2,3)} + \tilde{c}_{(2,3)} & c_{(2,2)} + \tilde{c}_{(2,2)} + \mu + \tilde{\mu} & c_{(2,1)} + \tilde{c}_{(2,1)} & c_{(2,0)} + \tilde{c}_{(2,0)} \\ c_{(1,3)} + \tilde{c}_{(1,3)} & c_{(1,2)} + \tilde{c}_{(1,2)} & c_{(1,1)} + \tilde{c}_{(1,1)} + \mu + \tilde{\mu} & c_{(1,0)} + \tilde{c}_{(1,0)} \\ c_{(0,3)} + \tilde{c}_{(0,3)} & c_{(0,2)} + \tilde{c}_{(0,2)} & c_{(0,1)} + \tilde{c}_{(0,1)} & c_{(0,0)} + \tilde{c}_{(0,0)} + \mu + \tilde{\mu} \end{bmatrix}$$

Fig. 1. Homomorphic XOR operation

Then, we use the matrix to encrypt a message μ by evaluating:

$$C = \mathsf{Flatten}(I_\ell \cdot \mu + c).$$

Here, in the equation I_ℓ is identity matrix with size ℓ.

– **Decrypt:** To decrypt the message we take the first row of the matrix and apply Inverse-Bit-Decomposition to form a NTRU' ciphertext:

$$\mathsf{BitDecomp}^{-1}\{c_{(0,\ell-1)}, c_{(0,\ell-2)}, \dots, c_{(0,1)}, c_{(0,0)}\} = c_0.$$

Once the NTRU' ciphertext is constructed, we are able to decrypt the message using the secret key f for the NTRU' scheme as $\lfloor c_0 f \rceil \bmod 2 = \mu$.

– **Eval.** The homomorphic XOR and AND operations are simply computed as matrix addition and multiplication operations, respectively followed by a Flatten operation, i.e.

$$C' = \mathsf{Flatten}(C + \tilde{C}) \quad , \quad C' = \mathsf{Flatten}(C \cdot \tilde{C}).$$

3.1 Correctness of Homomorphic Circuit Evaluation

The correctness of encryption/decryption trivially follows from the correctness of the NTRU' scheme. In this section we briefly demonstrate how the NTRU' ciphertext and associated F-NTRU ciphertext matrix forms are preserved thus allowing homomorphic evaluation. For clarity we use ciphertexts of sizes $\ell = 4$.

First note that $\mathsf{BitDecomp}^{-1}(C)$ is actually a vector that contains the encryptions of message bits scaled by powers of 2:

$$\mathsf{BitDecomp}^{-1}(\mathsf{Flatten}(I_\ell \cdot \mu + c)) = \\ \left[c_{\ell-1} + 2^{\ell-1} \cdot \mu, \quad \dots \ , \ c_1 + 2^1 \cdot \mu, \ c_0 + 2^0 \cdot \mu\right]^\top.$$

This is the form we need to preserve throughout homomorphic evaluations for correctness.

Homomorphic XOR. A homomorphic XOR operation between two ciphertext matrices is computed as shown in Figure 1. When we apply $\mathsf{BitDecomp}^{(-1)}$ to

$C' = \mathsf{Flatten}(C \cdot \tilde{C}) = [\boldsymbol{c'}_3, \boldsymbol{c'}_2, \boldsymbol{c'}_1, \boldsymbol{c'}_0]^\top =$

$$\begin{bmatrix} c_{(3,3)} + \mu & c_{(3,2)} & c_{(3,1)} & c_{(3,0)} \\ c_{(2,3)} & c_{(2,2)} + \mu & c_{(2,1)} & c_{(2,0)} \\ c_{(1,3)} & c_{(1,2)} & c_{(1,1)} + \mu & c_{(1,0)} \\ c_{(0,3)} & c_{(0,2)} & c_{(0,1)} & c_{(0,0)} + \mu \end{bmatrix} \cdot \begin{bmatrix} \tilde{c}_{(3,3)} + \tilde{\mu} & \tilde{c}_{(3,2)} & \tilde{c}_{(3,1)} & \tilde{c}_{(3,0)} \\ \tilde{c}_{(2,3)} & \tilde{c}_{(2,2)} + \tilde{\mu} & \tilde{c}_{(2,1)} & \tilde{c}_{(2,0)} \\ \tilde{c}_{(1,3)} & \tilde{c}_{(1,2)} & \tilde{c}_{(1,1)} + \tilde{\mu} & \tilde{c}_{(1,0)} \\ \tilde{c}_{(0,3)} & \tilde{c}_{(0,2)} & \tilde{c}_{(0,1)} & \tilde{c}_{(0,0)} + \tilde{\mu} \end{bmatrix}$$

Fig. 2. Homomorphic AND operation

the rows of the addition matrix we obtain:

$$[(c_3 + \tilde{c}_3) + 8(\mu + \tilde{\mu}), (c_2 + \tilde{c}_2) + 4(\mu + \tilde{\mu}),$$
$$(c_1 + \tilde{c}_1) + 2(\mu + \tilde{\mu}), (c_0 + \tilde{c}_0) + 1(\mu + \tilde{\mu})]$$

The ciphertext vector is still valid for the following two reasons:

1. The first part of the addition $c_i + \tilde{c}_i$ still holds an encryption of zero. The only difference is that it is noisier compared to a fresh encryption.
2. The second term in each entry is the message scaled by powers of two, i.e. $2^i \cdot (\mu + \tilde{\mu})$ as in a fresh ciphertext.

Homomorphic AND. A homomorphic AND operation between two ciphertext matrices is computed as shown in Figure 2. We summarize the derivation of the rows of the product matrix as follows:

Row 0: In Table 1 we show columns of the vector $\boldsymbol{c'}_0 = \mathsf{BitDecomp}(c'_0(x))$. In the last row of the table we evaluate $\mathsf{BitDecomp}^{-1}(\boldsymbol{c'}_0)$. The last column contains the respective powers of 2 used in $\mathsf{BitDecomp}^{-1}$ for easy reference.

$\boldsymbol{c'}_{(0,3)}$	$c_{(0,3)} \cdot (\tilde{c}_{(3,3)} + \tilde{\mu})$	$+c_{(0,2)} \cdot \tilde{c}_{(2,3)}$	$+c_{(0,1)} \cdot \tilde{c}_{(1,3)}$	$+(c_{(0,0)} + \mu) \cdot \tilde{c}_{(0,3)}$	8
$\boldsymbol{c'}_{(0,2)}$	$c_{(0,3)} \cdot \tilde{c}_{(3,2)}$	$+c_{(0,2)} \cdot (\tilde{c}_{(2,2)} + \tilde{\mu})$	$+c_{(0,1)} \cdot \tilde{c}_{(1,2)}$	$+(c_{(0,0)} + \mu) \cdot \tilde{c}_{(0,2)}$	4
$\boldsymbol{c'}_{(0,1)}$	$c_{(0,3)} \cdot \tilde{c}_{(3,1)}$	$+c_{(0,2)} \cdot \tilde{c}_{(2,1)}$	$+c_{(0,1)} \cdot (\tilde{c}_{(1,1)} + \tilde{\mu})$	$+(c_{(0,0)} + \mu) \cdot \tilde{c}_{(0,1)}$	2
$\boldsymbol{c'}_{(0,0)}$	$c_{(0,3)} \cdot \tilde{c}_{(3,0)}$	$+c_{(0,2)} \cdot \tilde{c}_{(2,0)}$	$+c_{(0,1)} \cdot \tilde{c}_{(1,0)}$	$+(c_{(0,0)} + \mu) \cdot (\tilde{c}_{(0,0)} + \tilde{\mu})$	1
c'_0	$\underbrace{c_{(0,3)} \cdot \tilde{c}_3 + c_{(0,2)} \cdot \tilde{c}_2 + c_{(0,1)} \cdot \tilde{c}_1 + c_{(0,0)} \cdot \tilde{c}_0 + c_0 \cdot \tilde{\mu} + \tilde{c}_0 \cdot \mu}_{\tilde{c}_0} + \mu \cdot \tilde{\mu}$				

Table 1. Derivation of Row 0 of product ciphertext

The final ciphertext in Table 1 has the proper ciphertext form. The first part of the ciphertext $c_{(0,3)} \cdot \tilde{c}_3 + c_{(0,2)} \cdot \tilde{c}_2 + c_{(0,1)} \cdot \tilde{c}_1 + c_{(0,0)} \cdot \tilde{c}_0 + c_0 \cdot \tilde{\mu} + \tilde{c}_0 \cdot \mu$ is still an encryption of zero since ciphertexts c_i and $\tilde{c}_i$ are encryptions of zero and their multiplication with a binary polynomial will result in zero encryptions as long as noise is contained.

$\boldsymbol{c'}_{(1,3)}$	$c_{(1,3)} \cdot (\tilde{c}_{(3,3)} + \tilde{\mu})$	$+c_{(1,2)} \cdot \tilde{c}_{(2,3)}$	$+(c_{(1,1)} + \mu) \cdot \tilde{c}_{(1,3)}$	$+c_{(1,0)} \cdot \tilde{c}_{(0,3)}$	8
$\boldsymbol{c'}_{(1,2)}$	$c_{(1,3)} \cdot \tilde{c}_{(3,2)}$	$+c_{(1,2)} \cdot (\tilde{c}_{(2,2)} + \tilde{\mu})$	$+(c_{(1,1)} + \mu) \cdot \tilde{c}_{(1,2)}$	$+c_{(1,0)} \cdot \tilde{c}_{(0,2)}$	4
$\boldsymbol{c'}_{(1,1)}$	$c_{(1,3)} \cdot \tilde{c}_{(3,1)}$	$+c_{(1,2)} \cdot \tilde{c}_{(2,1)}$	$+(c_{(1,1)} + \mu) \cdot (\tilde{c}_{(1,1)} + \tilde{\mu})$	$+c_{(1,0)} \cdot \tilde{c}_{(0,1)}$	2
$\boldsymbol{c'}_{(1,0)}$	$c_{(1,3)} \cdot \tilde{c}_{(3,0)}$	$+c_{(1,2)} \cdot \tilde{c}_{(2,0)}$	$+(c_{(1,1)} + \mu) \cdot \tilde{c}_{(1,0)}$	$+c_{(1,0)} \cdot (\tilde{c}_{(0,0)} + \tilde{\mu})$	1
c'_1	$\underbrace{c_{(1,3)} \cdot \tilde{c}_3 + c_{(1,2)} \cdot \tilde{c}_2 + c_{(1,1)} \cdot \tilde{c}_1 + c_{(1,0)} \cdot \tilde{c}_0 + c_1 \cdot \tilde{\mu} + \tilde{c}_1 \cdot \mu}_{\bar{c}_1} + 2(\mu \cdot \tilde{\mu})$				

Table 2. Derivation of Row 1 of product ciphertext

Row 1: We apply the same arithmetic for the row number 1 and form a similar construction in Table 2. We achieve a similar derivation for second row in Table 2. The only difference is that now the message is scaled by 2.

We can now generalize the arithmetic for each row i as follows:

$$c'_i = \underbrace{\sum_{j=0}^{\ell-1} c_{(i,j)} \cdot \tilde{c}_j + c_i \cdot \tilde{\mu} + \tilde{c}_i \cdot \mu}_{\bar{c}_i} + 2^i(\mu \cdot \tilde{\mu}). \qquad (1)$$

As shown in Equation 1, after a matrix multiplication the ciphertext vector elements contains a (noisier) encryption of zero along with the message scaled by powers of 2, i.e. $\boldsymbol{c'} = \{\bar{c}_{\ell-1} + 2^{\ell-1}\bar{\mu}, \ldots \bar{c}_1 + 2^1\bar{\mu}, \bar{c}_0 + 2^0\bar{\mu}\}$ in which $\bar{\mu} = \mu \cdot \tilde{\mu}$. Hence correctness holds throughout circuit evaluation as long as noise is contained.

4 Optimizations

In our scheme we are able to perform certain optimizations on the arithmetic operations to reduce computational complexity and the memory footprint. The two main optimizations we perform are:

Using High Radix Representations. The flattened ciphertext matrix takes up a large space, $\ell^2 = O(\log(q)^2)$. To mitigate we may us a higher radix representation, i.e. a larger radix 2^ω and group ω bits for each element of the ciphertext matrix. When we apply $\mathsf{BitDecomp}^{-1}$, we use the powers of the chosen radix $(2^w)^i$ instead of powers of 2^i to reconstruct the ciphertext. With this approach we drastically reduce the matrix size: $\ell' = \ell/\omega$. Note that the number of zero encryptions of $\mathsf{NTRU'}$ should be equal to ℓ' as well. Using the high radix encoding the ciphertext size is reduced by ω times. In addition, the ciphertext matrix size is reduced by ω^2 which decreases the complexity of matrix multiplication significantly.

Matrix Multiplication. A straight schoolbook matrix multiplication takes $\mathcal{O}(\ell^3)$ time and it may reduced to $\mathcal{O}(\ell^{2.374})$ time using Coppersmith-Winograd algorithm. However, to evaluate deep circuits we will need a large modulus and even with the Coppersmith-Winograd algorithm multiplication will be slow. Therefore, we change matrix multiplication into a matrix-vector multiplication as follows:

$$\mathsf{BitDecomp}^{-1}\left(\mathsf{BitDecomp}(\boldsymbol{c}) \cdot \mathsf{BitDecomp}(\tilde{\boldsymbol{c}})\right) = \mathsf{BitDecomp}(\boldsymbol{c}) \cdot \tilde{\boldsymbol{c}}.$$

A schoolbook matrix-vector multiplication has $\mathcal{O}(\ell^2)$ runtime which is faster than a matrix multiplication. The only downside of the algorithm is that a binary by a high radix polynomial multiplication should be implemented instead of a binary by binary polynomial multiplication. Although binary by high radix polynomial multiplication is slower, the time gap is closed in the higher level while computing the matrix-vector product for large ℓ values.

5 Security Analysis

Our scheme uses the NTRU' encryption scheme adopted from [21]. In F-NTRU , an attacker has access only to a ciphertext vector which holds NTRU' ciphertexts build with *fresh* encryptions of zero. Therefore as long as NTRU' ciphertexts are secure, our scheme is also secure. According to the Stehlé and Steinfeld [21], their scheme is IND-CPA secure under the hardness of R-LWE assumption as long as the error has a *wide distribution* $\mathbb{D}_{\mathbb{Z}^n,\sigma}$, i.e. $\sigma > \sqrt{q} \cdot \text{poly(n)}$. However, in the DHS and YASHE' schemes such a high noise instantiations are not possible due to the noise growth in homomorphic multiplications. Therefore, both schemes use narrow error distributions and additionally assume hardness of the Decisional Small Polynomial Ratio (DSPR) Problem along with the R-LWE assumption.

Fortunately, our scheme has a better control on noise management. The (size of the) noise increases linearly with the multiplicative depth. Therefore, we are able to use a *wide error distribution* and achieve IND-CPA security as in the original scheme by Stehlé and Steinfeld in [21]. Thus the standard deviation σ_{key} of the discrete Gaussian distribution $\mathbb{D}_{\mathbb{Z}^n,\sigma_{\text{key}}}$ needs to be set as:

$$\sigma_{\text{key}} > 2n\sqrt{\log\left(8nq\right)} \cdot q^{1/2+\epsilon}$$

for $\epsilon > 0$, i.e. $\epsilon = 2^{-128}$. In this setting we are able to generate a public key $h = g/f$, i.e. $g \in \chi_{\text{key}}$ and $f' \in \chi_{\text{key}}$ $(f = 2f + 1)$ in which χ_{key} is truncated $\mathbb{D}_{\mathbb{Z}^n,\sigma_{\text{key}}}$, that is indistinguishable from a uniformly random distribution.

For R-LWE security we follow the settings in [19]. The noise parameters s and e are sampled from the distribution χ_{err}, a truncated Gaussian distribution $\mathbb{D}_{\mathbb{Z}^n,\sigma_{\text{err}}}$. The standard deviation σ_{err} of the error distribution has the following bound $\sigma_{\text{err}} > \sqrt{n\log\left(n\right)}$. With this bound, we are able to add noise to our public key and keep it computationally indistinguishable from random uniform distribution.

As above mentioned, the distributions χ_{key} and χ_{err} are truncated Gaussian distributions. They are also defined as B-bounded[1] distributions which means

[1] We get rid of the subscripts *key* and *err* for simplicity. We use B_{key} and B_{err} if they need to be specified.

the samples are selected from $[-B, B]$. The bound value B is selected as a function of the advantage ϵ in a distinguishing attack, e.g. $\epsilon = 2^{-128}$. The function, e.g. see [36], gives for a sample $x \in \chi$ the probability of x being larger than $k \cdot \sigma$ for a factor k is:

$$\mathrm{Prob}_{x \leftarrow \mathbb{D}_{\mathbb{Z}^n,\sigma}}[|x| > k \cdot \sigma] = \mathrm{erf}(k/\sqrt{2}).$$

Using the equation above, we select the factor k as $k(\epsilon) = \min\{k \mid \mathrm{erf}(k/\sqrt{2}) < \epsilon\}$, and select the B-bound as $B = k(\epsilon) \cdot \sigma$.

Hermite Factor. The existing attacks rely on the lattice reduction algorithms and the best attack known in practice is BKZ 2.0 [37] by Chen and Nguyen. The quality of the security is measured by Hermite factor δ. In [38] Linder and Peikert derive a linear security estimate using the Hermite factor δ:

$$\log(T_{\mathrm{BKZ}}(\delta)) = 1.8/\log(\delta) - 110.$$

In order to have at least 128-bit security levels, i.e. $T_{\mathrm{BKZ}} = 2^{128}$, we need $\delta \leq 1.00525$. In [38] the Hermite factor δ is given as:

$$\delta(n, q, \sigma, \epsilon) = 2^{\log^2\left(q/\sigma \cdot \sqrt{2\log(1/\epsilon)}\right)/(4n\log q)}.$$

Using these equations and by fixing the Hermite factor, i.e. $\delta = 1.00525$, we achieve the same security level for various values of n by changing q and σ.

6 Noise Analysis

In this section we analyze the noise performance of our scheme with homomorphic evaluations. In case of a homomorphic addition, the noise increases only a small percent compared to homomorphic multiplication. In our analysis we want to determine the depth of the circuit we can evaluate and still decrypt correctly with growing noise. This analysis includes average case and worst case scenarios for homomorphic multiplication that estimates the number of possible multiplicative levels. Since additions contribute minimally to noise growth we focus only on homomorphic multiplications.

For an element $a \in \mathbb{R}$ we define the Euclidean norm $||a|| = \sqrt{\sum a_i^2}$ and the infinity norm $||a||_\infty = \max|a_i|$ for all possible values of i. In multiplication, we can bound the noise growth with the aid of the following Lemma.

Lemma 61 ([39,19]) *In a ring $R = \mathbb{Z}[x]/\langle x^n + 1\rangle$, for any two polynomials $a, b \in R$ we have the following norms $||ab|| \leq \sqrt{n}||a|| \cdot ||b||$ and $||ab||_\infty \leq n||a||_\infty \cdot ||b||_\infty$.*

Recursive Evaluation. We assume the evaluation circuit is arranged into a tree with levels of parallel multiplication gates that accept input the output ciphertexts from the previous level. Lets denote a ciphertext matrix that are at a certain multiplicative level i as $C^{(i)}$. Then, a ciphertext matrix at a multiplicative level $C^{(i)} = C^{(i-1)} \cdot \tilde{C}^{(i-1)}$. Lets recall that these ciphertext matrices are actually NTRU' ciphertext vectors with BitDecomp applied. We also denote the ciphertexts in the vector for multiplicative level i and row index j as $c_j^{(i)}$.

In the first multiplicative level, i.e. $i = 0$, ciphertext vectors are fresh encryptions with security parameters explained in Section 5. Basically, we have samples $g, f' \in \chi_{\text{key}}$ and $s, e \in \chi_{\text{err}}$ and ciphertexts $c_j^{(0)} = hs_j + 2e_j + 2^j \mu$ where μ is the message, $f = 2f' + 1$ and $h = 2gf^{-1}$ is public key. Lets recall that for ciphertext matrix multiplications we have NTRU' ciphertexts that hold the form given in Equation 1. The equation can be rewritten with level index to show the result of a multiplicative level as:

$$c_j^{(i)} = \sum_{k=0}^{\ell-1} c_{(j,k)} \cdot \tilde{c}_k^{(i-1)} + c_j^{(i-1)} \cdot \tilde{\mu} + \tilde{c}_j^{(i-1)} \cdot \mu + 2^j(\mu \cdot \tilde{\mu}).$$

We can simplify the equation, for noise evaluation, by replacing radix size polynomials $c_{(j,k)}$ with y_τ, choosing ciphertext vector index $j = 0$, and substituting $y_{(i)}$ in place of all ciphertexts since they have the same noise level for the same multiplicative level i:

$$y_{(i)} = y_{(i-1)} y_\tau \ell + y_{(i-1)} \tilde{\mu} + y_{(i-1)} \mu + \mu \tilde{\mu}.$$

To be able to decrypt $y_{(i)}$ correctly, we need $||y_{(i)} f||_\infty < q/2$. Thus, we need

$$\begin{aligned} ||y_{(i)} f||_\infty \leq\ & ||y_{(i-1)} f y_\tau \ell||_\infty + ||y_{(i-1)} f \tilde{\mu}||_\infty \\ & + ||y_{(i-1)} f \mu||_\infty + ||\mu \tilde{\mu} f||_\infty. \end{aligned}$$

Later, we expand $||y_{(i-1)} f||_\infty$ in terms of $||y_{(i-2)} f||_\infty$ and continue the process recursively. At the lowest level we have:

$$||y_{(0)} f||_\infty = ||c^{(0)} f||_\infty \leq ||2gf^{(-1)} sf||_\infty + ||2ef||_\infty.$$

Since $||g||_\infty = ||f'||_\infty = B_{\text{key}}$, $||s||_\infty = ||e||_\infty = B_{\text{err}}$, the worst-case noise is equal to

$$\begin{aligned} ||y_{(0)} f||_\infty & \leq 2nB_{\text{key}} B_{\text{err}} + 2nB_{\text{err}}(2B_{\text{key}} + 1) \\ & \leq 2nB_{\text{err}}(3B_{\text{key}} + 1). \end{aligned}$$

For an arbitrary level i, the noise can be evaluated recursively by setting $B_i = ||y_{(i)} f||_\infty$, $||y_\tau||_\infty = 2^\omega - 1$ and $||\mu||_\infty \leq 1$. The message at level i is bounded as $||\mu_i||_\infty \leq n^{2^i - 1}$. Then, the noise evaluation is evaluated as

$$\begin{aligned} \underbrace{||y_{(i)} f||_\infty}_{B_i} \leq\ & \underbrace{||y_{(i-1)} f y_\tau \ell||_\infty}_{\ell n(2^\omega - 1) B_{(i-1)}} + \underbrace{||y_{(i-1)} f \tilde{\mu}_i||_\infty}_{n^{2^i} B_{(i-1)}} \\ & + \underbrace{||y_{(i-1)} f \mu_i||_\infty}_{n^{2^i} B_{(i-1)}} + \underbrace{||\mu_i \tilde{\mu}_i f||_\infty}_{n^{2^{i+1}}(2B_{\text{k}ey} + 1)} \end{aligned}$$

which is also summarized as

$$B_i \leq \ell n(2^\omega - 1)B_{(i-1)} + 2n^{2^i}B_{(i-1)} + n^{2^{i+1}}(2B_{\text{key}} + 1)$$

In the average case the noise accumulation will be much slower than what the bound given above predicts due to the Gaussian distribution. Using the worst case noise bound, we can simply obtain the average case noise by substituting $\sqrt{n}$ and $\sqrt{\ell}$ for n and ℓ, respectively,

Taking Advantage of Noise Asymmetry. As derived earlier the ciphertext c' obtained from the homomorphic product of ciphertexts c and $\tilde{c}$ can be expressed as

$$c'_j = \sum_{k=0}^{\ell-1} c_{(j,k)} \cdot \tilde{c}_k + c_j \cdot \tilde{\mu} + \tilde{c}_j \cdot \mu + 2^j(\mu \cdot \tilde{\mu}).$$

It is important to note that roles of the input ciphertexts in the summation are not symmetric. The (left) input c is processed as binary polynomials breaking the structure of the ciphertext, while the (right) input $\tilde{c}$ is kept intact. Therefore the noise content in c becomes irrelevant in the summation. Since we have the freedom to switch the inputs, we may take advantage of this fact by by placing the noisier ciphertext to the left during our evaluations. In the extreme case we can always feed fresh ciphertexts from the right input. If we iterate i multiplications with fresh ciphertexts $\tilde{c}_i$ we obtain

$$\begin{aligned}
y_1 &= \tilde{y}_0 y_\tau \ell + \tilde{y}_0 \mu_0 + y_0 \tilde{\mu}_0 + \mu_0 \tilde{\mu}_0 \\
y_2 &= \tilde{y}_1 y_\tau \ell + \tilde{y}_1 \mu_1 + y_1 \tilde{\mu}_1 + \mu_1 \tilde{\mu}_1 \\
\vdots &= \vdots \\
y_i &= \tilde{y}_{i-1} y_\tau \ell + \tilde{y}_{i-1} \mu_i + y_{i-1} \tilde{\mu}_{i-1} + \mu_i \tilde{\mu}_{i-1}
\end{aligned}$$

The noise experienced in decrypting after iteration i will be

$$\begin{aligned}
||fy_i||_\infty &\leq ||f\tilde{y}_{i-1} y_\tau \ell||_\infty + ||f\tilde{y}_{i-1}\mu_i||_\infty \\
&\quad + ||fy_{i-1}\tilde{\mu}_{i-1}||_\infty + ||f\mu_i\tilde{\mu}_{i-1}||_\infty \\
&\leq ||(2\tilde{g}\tilde{s} + 2\tilde{e}f)y_\tau \ell||_\infty + ||(2\tilde{g}\tilde{s} + 2\tilde{e}f)\mu_i||_\infty \\
&\quad + ||fy_{i-1}\tilde{\mu}_{i-1}||_\infty + ||f\mu_i\tilde{\mu}_{i-1}||_\infty
\end{aligned}$$

Basically, we have samples $\tilde{g}, f' \in \chi_{\text{key}}$ and $\tilde{s}, \tilde{e} \in \chi_{\text{err}}$. For an arbitrary iteration i, the noise can be evaluated by setting $B_i = ||y_i f||_\infty$, $||y_\tau||_\infty = 2^\omega - 1$, $||\mu_0||_\infty \leq 1$ and for the messages $||\tilde{\mu}_i||_\infty \leq 1$, and $\mu_i = \mu_{i-1}\tilde{\mu}_{i-1}$, $||\mu_i||_\infty \leq n||\mu_{i-1}||_\infty$ and with $\tilde{\mu}_0 \leq 1$ we have $||\mu_i||_\infty \leq n^i$. We explicitly derive the noise bound as

$$\begin{aligned}
B_i = ||fy_i||_\infty \leq\ & [2n^2 B_{\text{key}} B_{\text{err}}(2^w - 1)\ell \\
& + 2n^2 B_{\text{err}}(2B_{\text{key}} + 1)(2^w - 1)\ell] \\
& + [2n^{i+2} B_{\text{err}} B_{\text{key}} + 2n^{i+2} B_{\text{err}}(2B_{\text{key}} + 1)] \\
& + [nB_{i-1}] + [n^{i+2}(2B_{\text{key}} + 1)]
\end{aligned}$$

Here for $B_0 = ||fy_0||_\infty = ||2gs + 2ef||_\infty \leq 5nB_{\text{key}}B_{\text{err}} + 2nB_{\text{err}}$. In the average case the noise growth can be captured by

$$\begin{aligned} B_{i,avg} \leq & [2nB_{\text{key}}B_{\text{err}}(2^w - 1)\sqrt{\ell} \\ & + 2nB_{\text{err}}(2B_{\text{key}} + 1)(2^w - 1)\sqrt{\ell}] \\ & + [2\sqrt{n^{i+2}}B_{\text{err}}B_{\text{key}} + 2\sqrt{n^{i+2}}B_{\text{err}}(2B_{\text{key}} + 1)] \\ & + [\sqrt{n}B_{i-1}] + [\sqrt{n^{i+2}}(2B_{\text{key}} + 1)] \end{aligned}$$

Improving Scalability with Single Bit Encryption. The analysis above shows that the noise growth scales with $\mathcal{O}(n)$ over the levels of encryption. In practice, during evaluations the noise term increases with the first term setting the noise floor, i.e. $\mathcal{O}(n^2B_{\text{key}}B_{\text{err}}\ell)$ and then with each additional level contributing by a factor of n (in the worst case). Therefore the number of levels supported is heavily determined by the dimension n.

The factor n is due to message polynomials we are encrypting. Encrypting polynomials enables batching and yields better amortized times. However, if we are willing to trade off batching, and instead encrypt bits we may improve scalability, significantly. We may make up for the loss of batching by employing much smaller parameter sizes. If we assume all messages $\mu, \tilde{\mu} \in \{0, 1\}$, then the noise bound $B_i = ||fy_i||_\infty$ simplifies to

$$\begin{aligned} B_i \leq & [2n^2B_{\text{key}}B_{\text{err}}(2^w - 1)\ell \\ & + 2n^2B_{\text{err}}(2B_{\text{key}} + 1)(2^w - 1)\ell] \\ & + [2nB_{\text{err}}B_{\text{key}} + 2nB_{\text{err}}(2B_{\text{key}} + 1)] \\ & + [B_{i-1}] + [(2B_{\text{key}} + 1)] \end{aligned}$$

Supporting Large Integer Messages Our scheme is capable of supporting a large message space using the approach in [26]. The authors facilitate an integer to polynomial (and reverse) mapping by changing the message modulus from $p = 2$ into a polynomial $p(x) = x - 2$ in the encryption, decryption and key generation stages. Then we may simply encode the message bits into the message polynomial by distributing each message bit into the coefficients of a polynomial. For instance, a bit-decomposed message $m = \sum_i m_i 2^i$ is encoded as $m(x) = \sum_i m_i x^i$. With such an encoding we are able to support a very large integer message space, i.e. $m \in [-2^n, 2^n]$.

After decryption, a message can be evaluated by simply applying $x = 2$ into the message polynomial. This approach also changes to the noise analysis slightly:

$$\begin{aligned} B_i = ||fy_i||_\infty \leq & [4n^2B_{\text{key}}B_{\text{err}}(2^w - 1)\ell \\ & + 4n^2B_{\text{err}}(4B_{\text{key}} + 1)(2^w - 1)\ell] \\ & + [4nB_{\text{err}}B_{\text{key}} + 4nB_{\text{err}}(4B_{\text{key}} + 1)] \\ & + [B_{i-1}] + [(4B_{\text{key}} + 1)] \end{aligned}$$

7 The F-NTRU′ Variant

The F-NTRU construction manages to eliminate evaluation keys and relinearizations and the DSPR assumption. However, the ciphertext grows due to the matrix based construction compared to a standard NTRU based SHE constructions. The growth in the ciphertext size not only increases the memory footprint, but also increases the latency of homomorphic evaluations. With this motive we introduce a variant we call F-NTRU′ that uses less noise, by sampling the secret key f' and g from a slightly narrower distribution, i.e. $\mathcal{O}(\text{poly}(n) \cdot q^{1/3})$ instead of $\mathcal{O}(\text{poly}(n) \cdot q^{1/2})$. From Section 5 remember that for F-NTRU we require $\sigma_{\text{key}} > 2n\sqrt{\log(8nq)} \cdot q^{1/2+\epsilon}$. For a discussion on the effect of noise distribution on the security of NTRU based schemes, and possible attacks enabled by the use of very narrow key distributions see [35].

Under this modification we can select smaller parameters. The security and noise analysis presented above still applies with the exception of the re-introduced DSPR assumption. Therefore we can still use the noise bounds and parameter derivations by simply setting the noise to the appropriate values, i.e. for f' and g we use $\sigma_{\text{key}} > 2n\sqrt{\log(8nq)} \cdot q^{1/3+\epsilon}$ and for e and s we use $\sigma_{err} = \sqrt{n\log(n)}$.

8 Circuit Evaluation

To take advantage of the scalability gained with single bit encryption, we need to maintain the message values in the ciphertexts always as 0 or 1 not only in fresh ciphertexts, but also in ciphertexts obtained through homomorphic evaluation. In [24] this is achieved by restricting the circuit to only (universal) NAND gates. For input ciphertexts A and B, we can explicitly express the gate operations as follows

– NOT: $C = I_N - A$
– AND: $C = A \cdot B$
– NAND: $C = I_N - A \cdot B$
– XOR: $C = (I_N - A) \cdot B + A \cdot (I_N - B) = A + B - 2A \cdot B$
– OR: $C = I_N - ((I_N - A) \cdot (I_N - B)) = A + B - A \cdot B$

The evaluation process requires costly matrix multiplications. However note that for decryption we only need the first cipherext vector element to recover the result. Thus, during circuit evaluation we multiply all the elements of the ciphertext vector with the first element of the left operand. The remaining elements in the ciphertext vector are simply discarded. With this approach we achieve ℓ times speedup in circuit evaluations. This is achieved by simply evaluating the boolean circuit from the left-hand side by only using the first element of the ciphertext vector. The fresh ciphertext is always given as input from the right and the accumulated ciphertext is kept on the left. In multiplications, bit decomposition is only applied to the first element and its sum of product is computed with the ciphertext vector on the right-hand side. For instance, we may compute $C' = C \cdot \tilde{C}$ as follows

- $C = [c_{\ell-1}, c_{\ell-2}, \dots, c_0]$ and $\tilde{C} = [\tilde{c}_{\ell-1}, \tilde{c}_{\ell-2}, \dots, \tilde{c}_0]$
- Discard the unused ciphertexts: $C = [0, 0, \dots, 0, c_0]$
- Take the bit decomposition of C: $\{c_{(0,\ell-1)} c_{(0,\ell-2)}, \dots, c_{(0,0)}\}$.
- Compute the sum of product: $C' = \sum_{i=0}^{\ell-1} c_{(0,i)} \cdot \tilde{c}_i$

Using this method, we are able to complete the circuit evaluation in ℓ operations rather than ℓ^2.

9 Complexity

In Table 3 we summarize the asymptotic complexities of F-NTRU and YASHE schemes. We use $\ell = (\log q)/\omega$ for radix size 2^ω. Note that our scheme does not require evaluation keys, but the ciphertexts consist of ℓ polynomials. After circuit evaluations the ℓ polynomials are reduced to a single polynomial ciphertext. On the other hand YASHE requires ℓ^3 polynomials for evaluation keys. For homomorphic AND evaluation our scheme requires ℓ^2 polynomial multiplications or using the one sided AND evaluation it uses ℓ polynomial multiplications. In case of YASHE, the scheme uses ℓ^2 polynomial multiplications followed by a costly key switching operation computed via ℓ^3 polynomial multiplications.

	F-NTRU/F-NTRU'	YASHE
Eval. Key Size	-	$\mathcal{O}(\ell^3 n \log q)$
Ciphertext Size	$\mathcal{O}(\ell n \log q)$	$\mathcal{O}(n \log q)$
Final Ciphertext Size	$\mathcal{O}(n \log q)$	$\mathcal{O}(n \log q)$
AND Eval.	$\mathcal{O}(\ell^2)$	$\mathcal{O}(\ell^2)$
One Sided AND Eval.	$\mathcal{O}(\ell^2)$	$\mathcal{O}(\ell)$
Key-Switching	-	$\mathcal{O}(\ell^3)$

Table 3. Comparison of F-NTRU and YASHE: homomorphic AND evaluation and Key Switching complexities are in terms of polynomial multiplications with $\ell = (\log q)/\omega$.

10 Parameter Selection

In our choice of parameters we aimed for a 128-bit security level. For comparison, we also tabulated parameter choices for selected depths for F-NTRU, F-NTRU' and YASHE as shown in Table 4. These parameters are for the single bit encryption case and for an implementation that takes advantage of the noise asymmetry. Note that for similar sizes of q, the dimension n doubles for F-NTRU' to keep the Hermite factor at $\delta(n) = 1.00525$, however we can accommodate roughly 3 times as many evaluation levels in F-NTRU' for the same choice of parameters.

In Table 5, we summarize the maximum possible number of multiplications, i.e. 2^L, for 128-bit security level. Note that it is advantageous to select large radix sizes ω for faster evaluation. This decreases the run-time significantly, however,

L	F-NTRU	F-NTRU'	YASHE
5	(10,134)	(11,103)	(11,359)
10	(10,145)	(11,111)	(13,840)
20	(10,162)	(11,126)	(14,1705)
30	(11,189)	(11,141)	(14,2538)

Table 4. Parameters $(\log(n), \log(q))$ to support depth L evaluations with $\omega = 16$. The distinguishing advantage is set to 2^{-128}.

the depth is reduced somewhat since the noise is increased. As for ciphertext sizes, we are able to decrease it by a factor of ω.

	$\omega = 16$		$\omega = 8$		$\omega = 4$	
$(N, \log q)$	Worst	Average	Worst	Average	Worst	Average
(512, 95)	0	0	0	0	0	1
(1024, 163)	6	20	14	28	18	32
(2048, 294)	67	82	75	90	79	94

Table 5. Entries give the equivalent multiplicative depth L, i.e. parameters support 2^L multiplications. The Hermite factor δ is chosen as 1.00525 for 128-bit security.

11 Implementation Results

We implemented the F-NTRU and F-NTRU' schemes using Shoup's NTL library version 9.6.4 [40] compiled with the GMP 6.1 package. We ran our experiments on a 125 GBs of RAM and Intel Xeon E5-2637v2 64-bit CPU server clocked at 3.5 Ghz. The timing results for homomorphic multiplication are summarized in Table 6 for the parameters choices in Table 4. Although the algorithm is suitable for parallelization, NTL's threading capabilities are limited. Therefore, when we use 4 threads, i.e. C=4, we only achieve 2 times speedup. Furthermore, it is clear from the table that using 8 threads does not change the timings significantly.

	F-NTRU			F-NTRU'		
L	C=1	C=4	C=8	C=1	C=4	C=8
5	11.0	6.2	5.8	17.1	10.3	9.0
10	12.1	6.6	6.9	17.4	9.4	9.1
20	15.3	7.5	7.8	19.5	10.6	9.9
30	32.9	21.9	17.6	22.0	12.9	12.1

Table 6. Homomorphic multiplication times (msec) for radix selection $\omega = 16$. C denotes the number of threads.

The main advantage of the proposed F-NTRU and F-NTRU' schemes is that it eliminates costly evaluation keys and the slow relinearization operations.In addition the homomorphic evaluation is immensely simplified as we do no longer care about keeping track of the evaluation levels per ciphertext. Finally, we summarize the evaluation key and ciphertext sizes in Table 7.

	Evaluation Key	**Ciphertext**		
	YASHE	F-NTRU	F-NTRU'	YASHE
L	$\omega = 2$	$\omega = 16$	$\omega = 16$	$\omega = 2$
5	3.86 TB	150 KB	180 KB	87 KB
10	478 TB	181 KB	194 KB	820 KB
20	n/a	222 KB	252 KB	3.3 MB
30	n/a	567 KB	317 KB	4.9 MB

Table 7. Evaluation key and ciphertext sizes to support depth L evaluation, i.e. 2^L multiplications.

12 Conclusion

We presented a new leveled FHE scheme F-NTRU based on a variant of NTRU and by making use of a recently proposed noise management technique. Our scheme manages to eliminate costly evaluation keys, and any need for key switching and relinearization operations. We rigorously analyzed the security and noise performance of the proposed scheme, and determined parameters needed for correct evaluation of circuits of arbitrary depth. Our scheme makes use of wide key distributions and does not suffer from the Subfield Lattice Attack as LTV and YASHE'. Unlike these schemes, our proposed scheme does not rely on the DSPR assumption.

We analyzed and compared our scheme to the only other NTRU based FHE scheme immune to the attack, i.e. YASHE. Unlike YASHE, our scheme does not suffer from costly evaluation keys and relinearization operations. More significantly is supports deep homomorphic evaluation at reasonable ciphertext sizes, i.e. 567 KB for 30 multiplicative levels. Our implementation achieves competitive speeds: multiplication in 5.8 msec to support 5 levels and 17.6 msec for 30 levels. With such speeds our scheme becomes relevant for use in restricted real-life applications. Finally, via a simple polynomial to integer mapping, our scheme supports homomorphic integer arithmetic with a very large message space as desired by many applications.

13 Acknowledgements

We would like to thank Leo Ducas for clarifying the Subfield Lattice Attack.

References

1. Bos, J.W., Lauter, K., Loftus, J., Naehrig, M.: Cryptography and Coding: 14th IMA International Conference, IMACC 2013, Oxford, UK, December 17-19, 2013. Proceedings, chap. Improved Security for a Ring-Based Fully Homomorphic Encryption Scheme, pp. 45–64. Springer Berlin Heidelberg, Berlin, Heidelberg (2013), http://dx.doi.org/10.1007/978-3-642-45239-0_4
2. Bos, J.W., Lauter, K., Naehrig, M.: Private predictive analysis on encrypted medical data. Tech. Rep. MSR-TR-2013-81 (September 2013), http://research.microsoft.com/apps/pubs/default.aspx?id=200652
3. Brakerski, Z.: Fully Homomorphic Encryption without Modulus Switching from Classical GapSVP, pp. 868–886. Springer Berlin Heidelberg, Berlin, Heidelberg (2012), http://dx.doi.org/10.1007/978-3-642-32009-5_50
4. Brakerski, Z., Gentry, C., Vaikuntanathan, V.: Fully homomorphic encryption without bootstrapping. Electronic Colloquium on Computational Complexity (ECCC) 18, 111 (2011)
5. Brakerski, Z., Vaikuntanathan, V.: Efficient fully homomorphic encryption from (standard) LWE. In: FOCS. pp. 97–106 (2011)
6. Çetin, G.S., Dai, W., Doröz, Y., Sunar, B.: Homomorphic autocomplete. IACR Cryptology ePrint Archive 2015, 1194 (2015), http://eprint.iacr.org/2015/1194
7. Çetin, G.S., Doröz, Y., Sunar, B., Martin, W.J.: An investigation of complex operations with word-size homomorphic encryption. IACR Cryptology ePrint Archive 2015, 1195 (2015), http://eprint.iacr.org/2015/1195
8. Çetin, G.S., Doröz, Y., Sunar, B., Savas, E.: Depth optimized efficient homomorphic sorting. In: Progress in Cryptology - LATINCRYPT 2015 - 4th International Conference on Cryptology and Information Security in Latin America, Guadalajara, Mexico, August 23-26, 2015, Proceedings. pp. 61–80 (2015), http://dx.doi.org/10.1007/978-3-319-22174-8_4
9. Chen, Y., Nguyen, P.Q.: BKZ 2.0: Better lattice security estimates. In: ASIACRYPT. pp. 1–20 (2011)
10. Coron, J.S., Mandal, A., Naccache, D., Tibouchi, M.: Fully homomorphic encryption over the integers with shorter public keys. In: CRYPTO. pp. 487–504 (2011)
11. Coron, J.S., Naccache, D., Tibouchi, M.: Public key compression and modulus switching for fully homomorphic encryption over the integers. In: EUROCRYPT. pp. 446–464 (2012)
12. Dai, W., Doroz, Y., Sunar, B.: Accelerating ntru based homomorphic encryption using gpus. In: High Performance Extreme Computing Conference (HPEC), 2014 IEEE. pp. 1–6 (Sept 2014)
13. Dai, W., Doröz, Y., Sunar, B.: Accelerating swhe based pirs using gpus. In: Brenner, M., Christin, N., Johnson, B., Rohloff, K. (eds.) Financial Cryptography and Data Security, Lecture Notes in Computer Science, vol. 8976, pp. 160–171. Springer Berlin Heidelberg (2015), http://dx.doi.org/10.1007/978-3-662-48051-9_12
14. Dai, W., Sunar, B.: cuhe: A homomorphic encryption accelerator library. In: Pasalic, E., Knudsen, L.R. (eds.) Cryptography and Information Security in the Balkans, Lecture Notes in Computer Science, vol. 9540, pp. 169–186. Springer International Publishing (2016), http://dx.doi.org/10.1007/978-3-319-29172-7_11
15. van Dijk, M., Gentry, C., Halevi, S., Vaikuntanathan, V.: Fully homomorphic encryption over the integers. In: EUROCRYPT. pp. 24–43 (2010)

16. Doröz, Y., Hu, Y., Sunar, B.: Homomorphic aes evaluation using the modified ltv scheme. Designs, Codes and Cryptography pp. 1–26 (2015), http://dx.doi.org/10.1007/s10623-015-0095-1
17. Doröz, Y., Sunar, B., Hammouri, G.: Financial Cryptography and Data Security: FC 2014 Workshops, BITCOIN and WAHC 2014, Christ Church, Barbados, March 7, 2014, Revised Selected Papers, chap. Bandwidth Efficient PIR from NTRU, pp. 195–207. Springer Berlin Heidelberg, Berlin, Heidelberg (2014), http://dx.doi.org/10.1007/978-3-662-44774-1_16
18. ElGamal, T.: Advances in Cryptology: Proceedings of CRYPTO 84, chap. A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms, pp. 10–18. Springer Berlin Heidelberg, Berlin, Heidelberg (1985), http://dx.doi.org/10.1007/3-540-39568-7_2
19. Fan, J., Vercauteren, F.: Somewhat practical fully homomorphic encryption. IACR Cryptology ePrint Archive 2012, 144 (2012)
20. Gentry, C.: A Fully Homomorphic Encryption Scheme. Ph.D. thesis, Stanford University (2009)
21. Gentry, C., Halevi, S.: Implementing Gentry's fully-homomorphic encryption scheme. In: EUROCRYPT. pp. 129–148 (2011)
22. Gentry, C., Halevi, S., Smart, N.P.: Better Bootstrapping in Fully Homomorphic Encryption, pp. 1–16. Springer Berlin Heidelberg, Berlin, Heidelberg (2012), http://dx.doi.org/10.1007/978-3-642-30057-8_1
23. Gentry, C., Halevi, S., Smart, N.P.: Fully Homomorphic Encryption with Polylog Overhead, pp. 465–482. Springer Berlin Heidelberg, Berlin, Heidelberg (2012), http://dx.doi.org/10.1007/978-3-642-29011-4_28
24. Gentry, C., Halevi, S., Smart, N.P.: Homomorphic Evaluation of the AES Circuit, pp. 850–867. Springer Berlin Heidelberg, Berlin, Heidelberg (2012), http://dx.doi.org/10.1007/978-3-642-32009-5_49
25. Gentry, C., Sahai, A., Waters, B.: Advances in Cryptology – CRYPTO 2013: 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part I, chap. Homomorphic Encryption from Learning with Errors: Conceptually-Simpler, Asymptotically-Faster, Attribute-Based, pp. 75–92. Springer Berlin Heidelberg, Berlin, Heidelberg (2013), http://dx.doi.org/10.1007/978-3-642-40041-4_5
26. Goldwasser, S., Micali, S.: Probabilistic encryption &amp; how to play mental poker keeping secret all partial information. In: Proceedings of the Fourteenth Annual ACM Symposium on Theory of Computing. pp. 365–377. STOC '82, ACM, New York, NY, USA (1982), http://doi.acm.org/10.1145/800070.802212
27. Halevi, S., Shoup, V.: HElib, homomorphic encryption library. Internet Source (2012)
28. Hoffstein, J., Pipher, J., Silverman, J.H.: Algorithmic Number Theory: Third International Symposiun, ANTS-III Portland, Oregon, USA, June 21–25, 1998 Proceedings, chap. NTRU: A ring-based public key cryptosystem, pp. 267–288. Springer Berlin Heidelberg, Berlin, Heidelberg (1998), http://dx.doi.org/10.1007/BFb0054868
29. Lauter, K., López-Alt, A., Naehrig, M.: Progress in Cryptology - LATINCRYPT 2014: Third International Conference on Cryptology and Information Security in Latin America Florianópolis, Brazil, September 17–19, 2014 Revised Selected Papers, chap. Private Computation on Encrypted Genomic Data, pp. 3–27. Springer International Publishing, Cham (2015), http://dx.doi.org/10.1007/978-3-319-16295-9_1

30. Lindner, R., Peikert, C.: Better key sizes (and attacks) for lwe-based encryption. In: CT-RSA. pp. 319–339 (2011)
31. López-Alt, A., Tromer, E., Vaikuntanathan, V.: 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. pp. 1219–1234. STOC '12, ACM, New York, NY, USA (2012), http://doi.acm.org/10.1145/2213977.2214086
32. Lyubashevsky, V., Peikert, C., Regev, O.: On Ideal Lattices and Learning with Errors over Rings, pp. 1–23. Springer Berlin Heidelberg, Berlin, Heidelberg (2010), http://dx.doi.org/10.1007/978-3-642-13190-5_1
33. Martin Albrecht, Shi Bai, L.D.: 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/
34. Naehrig, M., Lauter, K., Vaikuntanathan, V.: Can homomorphic encryption be practical? In: Proceedings of the 3rd ACM Workshop on Cloud Computing Security Workshop. pp. 113–124. CCSW '11, ACM, New York, NY, USA (2011), http://doi.acm.org/10.1145/2046660.2046682
35. Paillier, P.: Advances in Cryptology — EUROCRYPT '99: International Conference on the Theory and Application of Cryptographic Techniques Prague, Czech Republic, May 2–6, 1999 Proceedings, chap. Public-Key Cryptosystems Based on Composite Degree Residuosity Classes, pp. 223–238. Springer Berlin Heidelberg, Berlin, Heidelberg (1999), http://dx.doi.org/10.1007/3-540-48910-X_16
36. Rivest, R.L., Adleman, L., Dertouzos, M.L.: On data banks and privacy homomorphisms. Foundations of Secure Computation pp. 169–180 (1978)
37. Rivest, R.L., Shamir, A., Adleman, L.M.: A method for obtaining digital signatures and public-key cryptosystems. Commun. ACM 21(2), 120–126 (1978)
38. Shoup, V.: http://www.shoup.net/ntl/, NTL: A Library for doing Number Theory
39. Smart, N.P., Vercauteren, F.: Fully homomorphic simd operations. Designs, Codes and Cryptography 71(1), 57–81 (2014), http://dx.doi.org/10.1007/s10623-012-9720-4
40. Stehlé, D., Steinfeld, R.: Making NTRU as secure as worst-case problems over ideal lattices. Advances in Cryptology – EUROCRYPT '11 pp. 27–4 (2011)