Des. Codes Cryptogr. (2016) 80:333–358
DOI 10.1007/s10623-015-0095-1

Homomorphic AES evaluation using the modified LTV scheme

Yarkın Doröz[1] · **Yin Hu**[1] · **Berk Sunar**[1]

Received: 1 October 2014 / Revised: 25 April 2015 / Accepted: 6 May 2015 /
Published online: 28 May 2015

Abstract Since its introduction more than a decade ago the homomorphic properties of the NTRU encryption scheme have gone largely ignored. A variant of NTRU proposed by Stehlé and Steinfeld was recently extended into a full fledged multi-key fully homomorphic encryption scheme by López-Alt, Tromer and Vaikuntanathan (LTV). This NTRU based FHE presents a viable alternative to the currently dominant BGV style FHE schemes. While the scheme appears to be more efficient, a full implementation and comparison to BGV style implementations has been missing in the literature. In this work, we develop a customized implementation of the LTV. First parameters are selected to yield an efficient and yet secure LTV instantiation. We present an analysis of the noise growth that allows us to formulate a modulus cutting strategy for arbitrary circuits. Furthermore, we introduce a specialization of the ring structure that allows us to drastically reduce the public key size making evaluation of deep circuits such as the AES block cipher viable on a standard computer with a reasonable amount of memory. Moreover, with the modulus specialization the need for key switching is eliminated. Finally, we present a *generic* bit-sliced implementation of the LTV scheme that embodies a number of optimizations. To assess the performance of the scheme we homomorphically evaluate the full 10 round AES circuit in 29 h with 2048 message slots resulting in 51 s per AES block evaluation time.

Keywords Fully homomorphic encryption · NTRU · AES · Ring-LWE

Mathematics Subject Classification 94A60

Communicated by L. Perret.

✉ Yarkın Doröz
ydoroz@wpi.edu

Yin Hu
hhyy_best@alum.wpi.edu

Berk Sunar
sunar@wpi.edu

[1] Worcester Polytechnic Institute, 100 Institute Rd, Worcester, MA 01609, USA

1 Introduction

Fully homomorphic encryption has come a long way in a mere few years since the first plausibly secure construction was introduced by Gentry [1] in 2009. This advance settled an open problem posed by Rivest [2], and opened the door to many new applications. In a nutshell, by employing FHE one may perform an arbitrary number of computations directly on the encrypted data without revealing the secret key. This feature, makes FHE a powerful tool in multi-party computing and perfectly suited to protect sensitive data in distributed applications including those hosted on semi-trusted cloud servers.

The efficiency bottleneck that prevents FHE from being deployed in real-life applications is now being bridged with the introduction of numerous new optimizations and related proof-of-concept implementations. The first implementation of an FHE variant was proposed by Gentry and Halevi [3]. An impressive array of optimizations were proposed with the goals of reducing the size of the public-key and improving the performance of the primitives. Still, encryption of one bit takes more than a second on a high-end Intel Xeon based server, while the recrypt primitive takes nearly half a minute in the lowest security setting. In [4], a GPU based implementation of the same scheme was developed which managed to reduce the recryption time to a few seconds.

Recently more efficient schemes emerged based on the hardness of learning with errors (LWE) problem. In [5] Brakerski, Gentry, and Vaikuntanathan (BGV) introduced an LWE based scheme that reduces the need for bootstrapping. Instead the BGV scheme uses a new lightweight method, i.e. *modulus switching*, to mitigate noise growth in ciphertexts as homomorphic evaluation proceeds. While modulus switching cannot restore the original level of noise as bootstrapping does, it still manages to gain exponentially on depth of the circuits evaluated without affecting the depth of the decryption circuit. Therefore, as long as we can fix the depth of the circuit a priori, we can perform evaluations without bootstrapping using a *leveled implementation*.

In [6] Gentry, Halevi and Smart introduced the first evaluation of a complex circuit, i.e. a full AES block evaluation by using a BGV style scheme introduced earlier in [7] by the same authors. The scheme makes use of batching [7,8], key switching and modulus switching techniques to obtain an efficient leveled implementation. Three batching techniques are used to obtain bit–sliced, byte–sliced and SIMD implementations. With 5 min per block evaluation time the byte-sliced implementation is faster, but also requires less memory. The SIMD implementation takes about 40 min per block.

In [9], López-Alt, Tromer and Vaikuntanathan (LTV) presented a leveled FHE scheme based on the modified NTRU [10] scheme introduced earlier by Stehlé and Steinfeld [11]. A unique aspect of the LTV scheme is that it supports homomorphic evaluation of ciphertexts encrypted by using public keys assigned to different parties. The authors outline the scheme using a leveled implementation and introduce a technique called *relinearization* to facilitate key switching during the levels of the evaluation. Modulus switching is also performed after multiplication and addition operations. The security of LTV scheme relies on two assumptions: the ring LWE assumption, and the Decisional Small Polynomial Ratio (DSPR) assumption. The scheme also supports bootstrapping. While the scheme appears to be efficient, the analysis in [9] lacks concrete parameters.

Very recently Bos et al. [12] presented a leveled implementation based on LTV. The authors modify the proposed scheme LTV in a number of aspects to build up their own somewhat homomorphic encryption scheme. They also mention that one can implement the bootstrapping operation and turn their somewhat homomorphic scheme into a fully homomorphic

encryption scheme. The semantic security of LTV is based on uniformity of the public key which relies on the DSPR assumption. In [12], the LTV scheme is modified by adopting a tensor product technique introduced by Brakerski [13] such that the security depends only on standard lattice assumptions. Furthermore, they use the scale-invariant approach to achieve much more reduced noise growth. Lastly, the authors improve the flexibility of the scheme by splitting the message using the Chinese Remainder Theorem and then encrypting them into separate ciphertexts. This makes integer based arithmetic easier and more efficient with a cost of a reduction in the depth of circuits that can be evaluated with the scheme.

1.1 Our contributions

We introduce an implementation of the LTV FHE scheme along with a number of optimizations. More specifically we

- present a batched, bit-sliced implementation of the LTV scheme. The implementation is generic and is *not* customized to optimally evaluate any class of circuits (e.g. AES) more efficiently than others.
- resolve the parameter selection issue in the light of recent theoretical and experimental results in the field of lattice reduction.
- introduce a specialization of the rings that simplifies modulus switching and allows us to significantly reduce the size of the public key. We show that the impact of this specialization on the key space is negligibly small. Even further, with the specialization key switching is no longer needed.
- rigorously analyze the noise growth of the LTV scheme over the levels of computation, and develop a simple formula for estimating the number of bits one needs to cut during the modulus switching step.
- homomorphically evaluate the full 128-bit AES circuit in a bit-sliced implementation to demonstrate the scalability of the introduced technique. Our implementation is 5.8 times faster than the byte sliced implementation and 47 times faster than the SIMD implementation of [6].

2 Background

In [9], López-Alt, Tromer and Vaikuntanathan proposed a multi-key homomorphic encryption scheme (LTV) based on a modified NTRU [10] scheme previously introduced by Stehlé and Steinfeld [11]. In this section we adapt their presentation to derive a single-key formulation suitable for homomorphic evaluation by a single party.

2.1 Preliminaries

We work with polynomials in $\mathrm{R} = \mathbb{Z}_q[x]/\langle x^n + 1\rangle$ where n represents the lattice dimension. All operations are performed in $\mathrm{R}_q = \mathrm{R}/q\mathrm{R}$ where q is the prime modulus. Elements of $\mathbb{Z}_q$ are associated with elements of $\{\lfloor -\frac{q}{2} \rfloor, \ldots, \lfloor \frac{q}{2} \rfloor\}$. We require the ability to sample from a probability distribution χ, i.e. the truncated discrete Gaussian distribution $\mathbb{D}_{\mathbb{Z}^n,\sigma}$ with Gaussian parameter σ, deviation $r = \sigma/\sqrt{2\pi}$ and Gaussian function $e^{-\pi x^2/\sigma^2}$. For a detailed treatment of the discrete Gaussian distribution see [14]. A polynomial is B-bounded if all of its coefficients are in $[-B, B]$. For an element $a \in \mathrm{R}$ we define $||a||$ as the Euclidean norm and $||a||_\infty = \max|a_i|$ as the infinity norm where the following rules apply.

Lemma 1 ([9,15]) *Let $n \in \mathbb{N}$ and let $\phi(x) = x^n + 1$ and let $R = \mathbb{Z}[x]/\langle\phi(x)\rangle$. For any $a, b \in R$*

$$||ab|| \leq \sqrt{n}||a||||b||$$
$$||ab||_\infty \leq n||a||_\infty||b||_\infty .$$

Samples of the truncated discrete Gaussian distribution may be obtained from a discrete Gaussian distribution $\mathbb{D}_{\mathbb{Z}^n,\sigma}$ (see [14]) with the aid of the following lemma:

Lemma 2 ([14] Lemma 4.4) *Let $n \in \mathbb{N}$. For any real number $\sqrt{2\pi}r > \omega(\sqrt{\log(n)})$, we have*

$$\Pr_{x \leftarrow \mathbb{D}_{\mathbb{Z}^n,\sigma}}\left[||x|| > \sqrt{2\pi}r\sqrt{n}\right] \leq 2^{-n+1} .$$

Informally, the lemma tells us that by sampling from a discrete gaussian distribution $\mathbb{D}_{\mathbb{Z}^n,\sigma}$ we obtain a $\sigma\sqrt{n}$-bounded polynomial with very high probability.

2.2 The LTV scheme

We are now ready to introduce the LTV variant of the NTRU scheme. We would like to stress that we specialize the scheme to work in the single user setting for clarity. One (positive) side-effect is that we no longer need to relinearize after homomorphic addition operations. For a full description see [9]. The scheme sets its parameters by using the security parameter η as:

- an integer $n = n(\eta)$,
- a prime number $q = q(\eta)$,
- a degree-n polynomial $\phi(x) = \phi_\eta(x) = x^n + 1$,
- a $B(\eta)$- bounded error distribution $\chi = \chi(\eta)$ over the ring $R = \mathbb{Z}[x]/\langle\phi(x)\rangle$.

The primitives of the public key encryption scheme $\mathcal{E}$ = (KeyGen, Encrypt, Decrypt, Eval, Relinearize) are defined as follows:

- KeyGen: We choose a decreasing sequence of primes $q_0 > q_1 > \cdots > q_d$, a polynomial $\phi(x) = x^n + 1$ and set χ as a truncated discrete Gaussian distribution that is B-bounded. For each i, we sample $u^{(i)}$ and $g^{(i)}$ from distribution χ, set $f^{(i)} = 2u^{(i)} + 1$ and $h^{(i)} = 2g^{(i)}\left(f^{(i)}\right)^{-1}$ in ring $R_{q_i} = \mathbb{Z}_{q_i}[x]/\langle\phi(x)\rangle$. (If $f^{(i)}$ is not invertible in this ring, re-sample.) We then sample, for $i = 0, \ldots, d$ and for $\tau = 0, \ldots, \lfloor \log q_i \rfloor$, $s_\tau^{(i)}$ and $e_\tau^{(i)}$ from χ and publish evaluation keys $\left\{\zeta_\tau^{(i)}\right\}_\tau^i$ where $\zeta_\tau^{(i)} = h^{(i)}s_\tau^{(i)} + 2e_\tau^{(i)} + 2^\tau\left(f^{(i-1)}\right)^2$ in $R_{q_{i-1}}$. Then set; secret key $\mathsf{sk} = (f^{(i)})$ and public keys $\mathsf{pk} = (h^{(i)}, \zeta_\tau^{(i)})$.
- Encrypt: To encrypt a bit $b \in \{0, 1\}$ with a public key $(h^{(0)}, q_0)$, Encrypt first generates random samples s and e from χ and sets $c^{(0)} = h^{(0)}s + 2e + b$, a polynomial in R_{q_0}.
- Decrypt: To decrypt the ciphertext c with the corresponding private key $f^{(i)}$, Decrypt multiplies the ciphertext and the private key in R_{q_i} then compute the message by modulo two: $m = c^{(i)}f^{(i)} \pmod 2$.
- Eval: Arithmetic operations are performed directly on ciphertexts as follows: Suppose $c_1^{(0)} = \mathsf{Encrypt}(b_1)$ and $c_2^{(0)} = \mathsf{Encrypt}(b_2)$. Then XOR is effected by simply adding and AND is effected by simply multiplying the ciphertexts:

$$b_1 \oplus b_2 = \mathsf{Decrypt}\left(c_1^{(0)} + c_2^{(0)}\right) \quad \text{and} \quad b_1 \cdot b_2 = \mathsf{Decrypt}\left(c_1^{(0)} \times c_2^{(0)}\right).$$

Polynomial multiplication incurs a much greater growth in the noise, so each multiplication step is followed by a modulus switching. First, we compute

$$\tilde{c}^{(0)}(x) = c_1^{(0)} \cdot c_2^{(0)} \pmod{\phi(x)}$$

and then perform Relinearization, as described below, to obtain $\tilde{c}^{(1)}(x)$ followed by modulus switching $\mathsf{Encrypt}(b_1 \cdot b_2) = \left\lfloor \frac{q_1}{q_0} \tilde{c}^{(1)}(x) \right\rceil_2$ where the subscript 2 on the rounding operator indicates that we round up or down in order to make all coefficients equal modulo 2. The same process hold for evaluating with i^{th} level ciphertexts, e.g. computing $\tilde{c}^{(i)}(x)$ from $c_1^{(i-1)}$ and $c_2^{(i-1)}$.

– Relinearize: We will show the general process that computing $\tilde{c}^{(i)}(x)$ from $\tilde{c}^{(i-1)}(x)$. We expand $\tilde{c}^{(i-1)}(x)$ as an integer linear combination of 1-bounded polynomials $\tilde{c}^{(i-1)}(x) = \sum_\tau 2^\tau \tilde{c}_\tau^{(i-1)}(x)$ where $\tilde{c}_\tau^{(i-1)}(x)$ takes its coefficients from $\{0, 1\}$. We then define $\tilde{c}^{(i)}(x) = \sum_\tau \zeta_\tau^{(i)}(x) \tilde{c}_\tau^{(i-1)}(x)$ in R_{q_i}.
To see why relinearization works, observe that simple substitution gives us

$$\begin{aligned}
\tilde{c}^{(i)}(x) &= h^{(i)}(x) \left[\sum_{\tau=0}^{\lfloor \log q_i \rfloor} s_\tau^{(i)}(x) \tilde{c}_\tau^{(i-1)}(x) \right] + 2 \left[\sum_{\tau=0}^{\lfloor \log q_i \rfloor} e_\tau^{(i)}(x) \tilde{c}_\tau^{(i-1)}(x) \right] \\
&\quad + \left[f^{(i-1)} \right]^2 \sum_{\tau=0}^{\lfloor \log q_i \rfloor} 2^\tau \tilde{c}_\tau^{(i-1)}(x) \\
&= h^{(i)}(x) S(x) + 2E(x) + \left[f^{(i-1)} \right]^2 \tilde{c}^{(i-1)}(x) \\
&= h^{(i)}(x) S(x) + 2E(x) + \left[f^{(i-1)} c_1^{(i-1)}(x) \right] \left[f^{(i-1)} c_2^{(i-1)}(x) \right] \\
&= h^{(i)}(x) S(x) + 2E'(x) + m_1 m_2
\end{aligned}$$

modulo q_{i-1} for some pseudorandom polynomials $S(x)$ and $E'(x)$. This ensures that the output of each gate takes the form of a valid encryption of the product $m_1 m_2$ of plaintexts with reduced noise. Later in Sect. 5 we study the growth of noise under relinearization more carefully.

3 Parameter selection in the LTV

A significant challenge in implementing and improving LTV is parameter selection. In [9] and [10] the security analysis is mostly given in asymmptotics by reduction to the related learning with errors (LWE) problem [11]. In this section, after briefly reviewing the security of the LTV scheme we summarize the results of our preliminary work on parameter selection.

3.1 DSPR problem

The scheme proposed by Stehlé and Steinfeld [11] is a modification to NTRU [10], whose security can be reduced to the hardness of the Ring-LWE (RLWE) problem. The reduction relies on the hardness of the Decisional Small Polynomial Ratio ($\mathrm{DSPR}_{\phi,q,\chi}$) Problem defined as follows:

Definition 1 ([9,11] decisional small polynomial ratio (DSPR$_{\phi,q,\chi}$) Problem) Let $\phi(x) \in \mathbb{Z}[x]$ be a polynomial of degree n, $q \in \mathbb{Z}$ be a prime integer, and let χ denote a distribution over the ring $R = Z[x]/(\phi(x))$. The decisional small polynomial ratio problem DSPR$_{\phi,q,\chi}$ is to distinguish between the following two distributions:

- a polynomial $h = gf^{-1}$, where f and g are sampled from the distribution χ (with f invertible over R_q), and
- a polynomial h sampled uniformly at random over R_q.

Stehlé and Steinfeld have shown that the DSPR$_{\phi,q,\chi}$ problem is hard even for unbounded adversaries when n is a power of two, $\phi(x) = x^n + 1$ is the n-th cyclotomic polynomial, and χ is the discrete Gaussian $\mathbb{D}_{\mathbb{Z}^n,\sigma}$ for $\sigma > \sqrt{q} \cdot \text{poly}(n)$. Specifically, the security reduction is obtained through a hybrid argument as follows:

1. Recall that for the LTV scheme, the public key is of the form $h = 2gf^{-1}$ where g, f chosen from a Gaussian distribution χ where f is kept secret. If the DSPR problem is hard, we can replace $h = 2gf^{-1}$ by some uniformly sampled h'.
2. Once h is replaced by h', the encryption $c = h's + 2e + m$ takes the form of the RLWE problem and we can replace the challenge cipher by $c' = u + m$ with a uniformly sampled u, thereby ensuring security.

In this way we can reduce the LTV scheme into a RLWE problem. However, the RLWE problem is still relatively new and lacks thorough security analysis. A common approach is to *assume* that RLWE follows the same behavior as the LWE problem [6]. Then, the second part of the reduction immediately suggest a distinguishability criteria in the lattice setting. The matrix $\mathbf{H}$ is derived from the coefficients of the public key polynomial h as

$$\mathbf{H} = \begin{pmatrix} h_0 & h_1 & \cdots & h_{n-1} \\ -h_{n-1} & h_0 & \cdots & h_{n-2} \\ \vdots & \vdots & \ddots & \vdots \\ -h_1 & -h_2 & \cdots & h_0 \end{pmatrix}.$$

We can connect the q-ary lattice $\Lambda_{\mathbf{H}}$ definition to the distinguishability of the masks in the LTV scheme. For simplicity we consider only a single level instantiation of LTV. To encrypt a bit $b \in \{0, 1\}$ with a public key (h, q), we first generate random samples s and e from χ and compute the ciphertext $c = hs + 2e + b \pmod q$. Here we care about the *indistinguishability* of the mask $hs + 2e$ from a randomly selected element of R_q. We can cast the encryption procedure in terms of the q-ary lattice $\Lambda_{\mathbf{H}}$ as follows $\mathbf{c} = \mathbf{Hs} + 2\mathbf{e} + \mathbf{b}$ where we use boldface symbols to denote the vector representation of polynomials obtained trivially by listing the polynomial coefficients. Then the decisional LWE problem in the context of the LTV scheme is to distinguish between the following two distributions:

- a vector $\mathbf{v}$ sampled randomly from $\mathbb{Z}^n$, and
- a vector $\mathbf{v} = \mathbf{Hs} + 2\mathbf{e}$ where $\mathbf{e}$ and $\mathbf{s}$ are sampled from the distribution $\mathbb{D}_{\mathbb{Z}^n,\sigma}$, respectively.

Given a vector $\mathbf{v}$ we need to decide whether this is a randomly selected vector or close to an element of the q-ary lattice $\Lambda_{\mathbf{H}}$. The natural approach [16] to distinguishing between the two cases is to find a short vector w in the dual lattice $\Lambda^*_{\mathbf{H}}$ and then check whether $w \cdot v^T$ is close to an integer. If not then we decide that the sample was randomly chosen; otherwise we conclude that v is a noisy lattice sample. Here we can follow the work of Micciancio and Regev [16, Sect. 5.4] who considered the security of LWE distinguishability with the dual basis approach. They note that this method is effective as long as the perturbation of v from

a lattice point in the direction of w is not much bigger than $1/||w||$. Since our perturbation is Gaussian, its standard deviation in the direction of w is $r' = \sqrt{2}r$. Therefore we need $r \gg 1/(\sqrt{2}||w||)$. Micciancio and Regev note that restricting $r' > 1.5/(\sqrt{2}||w||)$ provides a sufficient security margin and derive the following criteria

$$r' \geq 1.5q \cdot \max\left(\frac{1}{q}, 2^{-2\sqrt{n\log(q)\log(\delta)}}\right).$$

This gives us another condition to satisfy once q, n and δ are selected.

3.2 Concrete parameters

Stehlé and Steinfeld reduced the security of their scheme to the hardness of the RLWE. Unfortunately, the reduction only works when a *wide distribution* $\mathbb{D}_{\mathbb{Z}^n,\sigma}$, i.e. $\sigma > \sqrt{q}\cdot\text{poly}(n)$ is used. Due to noise growth with such an instantiation the LTV scheme [9] will not be able to support even a single homomorphic multiplication. Therefore [9] *assumes* the hardness of $\text{DSPR}_{\phi,q,\chi}$ for smaller r values in order to support homomorphic evaluation. The impact of the new parameter settings to the security level is largely unknown and requires further research. However, even if we assume that the $\text{DSPR}_{\phi,q,\chi}$ problem is difficult, we still need to ensure the hardness of the RLWE problem. As we discussed above, a common approach is to assume that it follows the same behavior as the LWE problem. Under this assumption only, we can select parameters. If we omit the noise, given the prime number q and k-bit security level, the dimension is bounded as in [6] as $n \leq \log(q)(k+110)/7.2$. This bound is based on experiments run by Lindner and Peikert [17] with the NTL library. The bound is rather loose since it is not exactly clear how the experiments will scale to larger dimensions. For instance, [17] ran experiments with the NTL library but extrapolates a running time which grows as $2^{\mathcal{O}(k)}$ where k is the block size, whereas NTL's enumeration implementation grows as $2^{\mathcal{O}(k^2)}$. Another issue is the assumption of $\delta_0 = \mathcal{O}(2^{1/k})$ which should be $\delta_0 = \mathcal{O}(k^{1/k})$. On the positive side, these simplifications yield a loose upper bound and should not negatively affect the security.

For example, given a 256-bit prime q, an 80-bit security level will require dimension $n = 6756$. This large estimate is actually an upper bound and assumes that the LTV scheme can be reduced to the RLWE problem. It is not clear whether the reverse is true, i.e. whether attacks against the RLWE problem apply to the LTV scheme. For instance, the standard attack on the LWE problem requires many samples generated with the *same* secret s. However, in the LTV scheme, the corresponding samples are ciphertexts of the form $c = h's + 2e + m$, where the s polynomials are randomly generated and independent. This difference alone suggests that standard attacks against LWE problems cannot be directly applied to the LTV scheme.

3.3 NTRU lattice attacks

As a variant of NTRU, the LTV scheme suffers from the same attack as the original NTRU. We can follow a similar approach as in the original NTRU paper [10] (see also [18]) to find the secret f: Consider the following $2n$ by $2n$ NTRU lattice where the h_i are the coefficients of $h = 2gf^{-1}$. Let $\Lambda_{\mathcal{L}}$ be the lattice generated by the matrix.

$$\mathcal{L} = \begin{pmatrix} \mathbf{I} & \mathbf{H} \\ \mathbf{0} & q\mathbf{I} \end{pmatrix}$$

where $\mathbf{H}$ is derived from the public key polynomial h as defined above. Clearly, $\Lambda_{\mathcal{L}}$ contains the vector $a = (f, 2g)$ which is short, i.e. $||a||_\infty \leq 4B + 1$. Now the problem is transformed to searching for short lattice vectors. Quite naturally, to be able to select concrete parameters with a reasonable safety margin we need to have a clear estimate on the work factor of finding a short vector in $\Lambda_{\mathcal{L}}$.

In what follows, we present a Hermite (work) factor estimate, and experimental results that will allow us to choose safe parameters.

3.4 Hermite factor estimates

Gama and Nguyen [19] proposed a useful approach to estimate the hardness of the SVP in an n-dimensional lattice $\Lambda_{\mathcal{L}}$ using the Hermite factor δ defined as

$$\left(\prod_{i=1}^{d} \lambda_i(\Lambda_{\mathcal{L}})\right)^{1/d} \leq \sqrt{\delta}\,\mathrm{VOL}(\Lambda_{\mathcal{L}})^{1/n}\,, \tag{1}$$

where in the equation $\lambda_i(\Lambda_{\mathcal{L}})$ denotes the i-th minimum of the lattice $\Lambda_{\mathcal{L}}$ and d is any number between the range $1 \leq d \leq n$. More practically we can compute δ as

$$\delta^n = ||b_1||/\det(\Lambda_{\mathcal{L}})^{1/n}$$

where $||b_1||$ is the length of the shortest vector or the length of the vector for which we are searching. The authors also estimate that, for larger dimensional lattices, a factor $\delta^n \leq 1.01^n$ would be the feasibility limit for current lattice reduction algorithms. In [17], Lindner and Peikert gave further experimental results regarding the relation between the Hermite factor and the break time as $t(\delta) := \log(T(\delta)) = 1.8/\log(\delta) - 110$. For instance, for $\delta^n = 1.0066^n$, we need about 2^{80} s on the platform in [17].

For the LTV scheme, we can estimate the δ of the NTRU lattice and thus the time required to find the shortest vector. Clearly, the NTRU lattice has dimension $2n$ and volume q^n. However, the desired level of approximation, i.e. the desired $||b_1||$ is unclear. In [19], Gama and Nguyen use q as the desired level for the original NTRU. However, for the much larger q used in the LTV scheme, this estimate will not apply. In particular, Minkowski tells us that $\Lambda_{\mathcal{L}}$ has a nonzero vector of length at most $\det(L)^{1/t}\sqrt{t}$ where t is the dimension. There will be exponentially many (in t) vectors of length $\mathrm{poly}(t)\det(L)^{1/t}$.

To overcome this impasse we make use of an observation by Coppersmith and Shamir [20]: we do not need to find the precise secret key since most of the vectors of similar norm will correctly decrypt NTRU ciphertexts. Setting $||b_1||$ as the norm of the short vector we are searching for and volume as q^n, we can simplify the Hermite factor to

$$\delta^{2n} = \frac{||b_1||}{(q^n)^{1/2n}}\,.$$

Following the recommendation of [20], we set the norm of b_1 as $q/4$. Coppersmith and Shamir observed that $q/10$ can ensure a successful attack in the majority of cases for NTRU with dimension $n = 167$ while $q/4$ is enough to extract some information. With a much larger dimension used than in NTRU, we may need a $||b||$ even smaller than $q/10$ to fully recover a usable key. However, we choose $q/4$ here to provide a conservative estimate of the security parameters. Thus $\delta^{2n} = \frac{q/4}{q^{1/2}} = \sqrt{q}/4$. In Table 1 we compiled the Hermite factor for various choices of q and n values.

Table 1 Hermite factor estimates for various dimensions n and sizes of q

n	$\log_2(q)$		
	512	1024	2048
2^{13}	1.0108	1.0218	1.0441
2^{14}	**1.0053**	1.0108	1.0218
2^{15}	**1.0027**	**1.0054**	1.0108
2^{16}	**1.0013**	**1.0027**	**1.0054**

Bold values indicate the parameters that ensure security level 2^{80} or higher
According to [17] for $\delta^n = 1.0066^n$, we need about 2^{80} s computation on a current PC. Therefore, we need $\delta < 1.0066$ and the smaller δ the higher the security margin will be

Table 2 Estimated security level with BKZ

Dimension	28,340	28,940	30,140	31,300	32,768
Security	70	80	100	120	144

Running times were collected on an Intel Xeon 2.9 GHz machine and converted to bits by taking the logarithm

3.5 Experimental approach

As a secondary countermeasure we ran a large number of experiments to determine the time required to compromise the LTV scheme following the lattice formulation of Hoffstein, Pipher, and Silverman with the relaxation introduced by Coppersmith and Shamir [20]. We generated various LTV keys with coefficient size $\log(q) = 1024$ and various dimensions. To search the short vectors required in the attacks described as above we used the Block-Korkin-Zolotarev (BKZ) [21] lattice reduction functions in Shoup's NTL Library 6.0 [22] linked with the GNU Multiprecision (GMP) 5.1.3 package. We set the LLL constant to 0.99 and ran the program with the block size 2. The block size has exponential impact on the resulting vector size and the running time of the algorithm. For the dimensions covered by our experiments, even the lowest block size was enough to successfully carry out attacks. Experiment results show that with the same block size, the size of the recovered keys grows exponentially with the dimension and the time for the algorithm grows polynomially with the dimension. As discussed above, the recovered vectors are only useful if they are shorter than $q/4$. When the dimension is sufficiently large we end up with vectors longer than this limit, and we will need larger block sizes causing an exponential rise in the time required to recover a useful vector [21]. From the collected data, we estimated that the block size of 2 can be used until about dimension $n = 26{,}777$.

Clearly, we cannot run test on such large dimensions to examine the exponential effects and estimate the cost for higher dimensions. To investigate the detailed impact of larger block sizes, we ran the experiment on low dimensions with higher block sizes and checked the changes on the recovered key sizes and the running time. The result of the experiment follows the prediction of [21], i.e. the result vector size decreases exponentially while the running time grows exponentially with the block size. Assuming that the higher dimensions follow similar rates, we estimate the security level for higher dimensions in Table 2. The estimation assumes the relation between parameters follows a similar pattern for low dimension and high

Table 3 Hermite factor δ estimates for security level **sec** reported in [25]

Dimension	1000	5000	10,000	15,000	20,000	25,000	30,000	40,000	50,000	60,000
sec = 64	1.00851	1.00896	1.00918	1.00931	1.00940	1.00948	1.00954	1.00964	1.00972	1.00979
sec = 80	1.00763	1.00799	1.00811	1.00826	1.00833	1.00839	1.00846	1.00851	1.00857	1.00862
sec = 128	1.00592	1.00609	1.00619	1.00624	1.00628	1.00629	1.00634	1.00638	1.00641	1.00644

dimensions and ignores all sub-exponential terms.[1] Therefore the estimated security level is not very precise. However, the results are not far off from what the Hermite factor estimate predicts. For instance, our experiments predict a 80-bit security for dimension $n = 28{,}940$ with $\log(q) = 1024$. The Hermite work factor estimate for the same parameters yields $\delta = 1.0061$. This is slightly more conservative than [17] whose experiments found that $\delta = 1.0066$ for the same security level.

3.6 BKZ 2.0

In a recent work, van de Pol and Smart [23] demonstrated that it is possible to work with lattices of smaller dimensions while maintaining the same security level by utilizing the BKZ-2.0 simulator of Chen and Nguyen. They argue that general assumption of a secure Hermite factor $\delta_{\mathcal{B}}$, for a lattice basis $\mathcal{B}$, that works for every dimensional lattice is not true. Therefore, one should take into account the hardness of lattice basis reduction in higher dimensions during the parameter selection process. The authors use the following approach to determine the Hermite factor $\delta_{\mathcal{B}}$ and the dimensions (n, q) of the lattice. First, the security parameter **sec**, e.g. 80, 128 or 256, is selected which corresponds to visiting $2^{\mathbf{sec}}$ nodes in the BKZ algorithm. Then the lattice dimension d and $\delta_{\mathcal{B}}$ are chosen such that the reduction works by visiting $2^{\mathbf{sec}}$ nodes. The evaluation is carried out with the simulator of Chen and Nguyen [24] for various block sizes and number of rounds. This results in a Hermite factor $\delta_{\mathcal{B}}$ as a function of lattice dimension d and security parameter **sec**. Lastly, they use the Hermite factor $\delta_{\mathcal{B}}$ to obtain (n, q) using the distinguishing attack analysis of Micciancio and Regev [16].

This work was later revisited by Lepoint and Naehrig [25]. Pol and Smart [23] only computed the Hermite factor for powers of two and used linear interpolation to match the enumeration costs to compute the Hermite factors. On the other hand Lepoint and Naehrig performed the experiments for all dimensions from 1000 to 65,000 and further they used quadratic function interpolation to set the missing values of enumeration costs. This results in a more precise Hermite factor computation. Also, the authors rely on the more recent work of Chen and Nguyen [26] for enumeration costs to determine the Hermite factors. Their Hermite factor computation is given in Table 3.

4 Optimizations

4.1 Batching

Batching has become an indispensable tool for boosting the efficiency of homomorphic evaluations [8]. In a nutshell, batching allows us to evaluate a circuit, e.g. AES, on multiple

[1] Ignoring those terms will result in a more conservative estimation.

independent data inputs simultaneously by embedding them into the same ciphertext. With batching multiple message bits belonging to parallel data streams are packed into a single ciphertext all undergoing the same operation similarly as in the single instruction multiple data (SIMD) computing paradigm.

The LTV scheme we use here permits the encryption of binary polynomials as messages. However a simple encoding where each message polynomial coefficient holds a message bit is not very useful when it comes to the evaluation of multiplication operations. When we multiply two ciphertexts (evaluate an AND) the resulting ciphertext will contain the product of the two message polynomials. However, we will not be able to extract the parallel product of message bits packed in the original ciphertext operands. The cross product terms will overwrite the desired results. Therefore, a different type of encoding of the message polynomial is required so that AND and XOR operations can be performed on batched bits fully in parallel. We adopted the technique presented by Smart and Vercauteren [8]. Their technique is based on an elegant application of the Chinese Remainder Theorem on a cyclotomic polynomial $\Phi_m(x)$ where $\deg(\Phi_m(x)) = \phi(m)$. An important property of cyclotomic polynomials with m odd is that it factorizes into same degree factors over $\mathbb{F}_2$. In other words, Φ_m has the form

$$\Phi_m(x) = \prod_{i \in [\ell]} F_i(x),$$

where ℓ is the number of factors irreducible in $\mathbb{F}_2$ and $\deg(F_i(x)) = d$ and $d = N/\ell$. The parameter d is the smallest value satisfying $m|(2^d - 1)$. Each factor F_i defines a *message slot* in which we can embed message bits. Actually we can embed elements of $\mathbb{F}_2[x]/\langle F_i \rangle$ and perform batched arithmetic in the same domain. However, in this paper we will only embed elements of $\mathbb{F}_2$ in the message slots. To pack a vector of ℓ message bits $\mathbf{a} = (a_0, a_1, a_2, \ldots, a_{\ell-1})$ into a message polynomial $a(x)$ we compute the CRT inverse on the vector $\mathbf{a}$

$$a(x) = \mathrm{CRT}^{-1}(\mathbf{a}) = a_0 M_0 + a_1 M_1 + \cdots + a_{\ell-1} M_{\ell-1} \pmod{\Phi_m}.$$

The values M_i are precomputed values that are shown as:

$$M_i = \frac{\Phi_m}{F_i(x)} \left(\left(\frac{\Phi_m}{F_i(x)} \right)^{-1} \pmod{F_i(x)} \right) \pmod{\Phi_m}.$$

The batched message can be extracted easily by performing modular reduction on the polynomial, e.g. $a_i = a(x) \pmod{F_i(x)}$. Due to the Chinese Remainder Theorem multiplication and addition of the message polynomials carry through to the residues: $a_i \cdot b_i = a(x) \cdot b(x) \pmod{F_i(x)}$ and $a_i + b_i = a(x) + b(x) \pmod{F_i(x)}$.

4.2 Reducing the public key size

To cope with the growth of noise, following Brakerski et al [5] we introduce a series of decreasing moduli $q_0 > q_1 > \cdots > q_{t-1}$; one modulus per circuit level. Modulus switching is a powerful technique that exponentially reduces the growth of noise during computations. Here we introduce a mild optimization that allows us to reduce the public key size drastically. We require that $q_i = p^{t-i}$ for $i = 0, \ldots, t-1$ where $p \in \mathbb{Z}$ is a prime integer. Therefore, $\mathbb{Z}_{q_i} \supset \mathbb{Z}_{q_j}$ for any $i < j$. We also require the secret key $f \in \mathbb{Z}_{q_0}/\langle \Phi(x) \rangle$ to be invertible in all rings $\mathbb{Z}_{q_i}$. Luckily, the following lemma from [27] tells us that we only need to worry

about invertibility in $\mathbb{Z}_p = \mathbb{F}_p$. Note that the lemma is given for $R'_p = \mathbb{Z}_p[x]/\langle x^n - 1\rangle$ however the proof given in [27] is generic and also applies to the R_p setting.

Lemma 3 (Lemma 3.3 in [27]) *Let p be a prime, and let f be a polynomial. If f is a unit in R'_p, (or R_p) then f is a unit in R'_{p^k} (or R_{p^k}) for every $k \geq 1$.*

Under this condition the inverse $f^{-1} \in \mathbb{Z}_{q_0}/\langle \Phi(x)\rangle$ which is contained in the public key h will persist through the levels of computations, while implicitly being reduced to each new subring $\mathbb{Z}_{q_{i+1}}/\langle \Phi(x)\rangle$ when q_{i+1} is used in the computation. More precisely, let $f^{(i)}(x) = f(x)^{-1} \pmod{q_i}$. Then we claim $f^{(i)} \pmod{q_{i+1}} = f^{(i+1)}$ for $i = 0, \ldots, t-1$. To see why this works, note that by definition it holds that $f(x)f^{(t-1)}(x) = 1 \pmod{p}$ which allows us to write $f(x)f^{(t-1)}(x) = 1 - pu(x)$ for some $u(x)$ and form the geometric (Maclaurin series) expansion of $f(x)^{-1}$ w.r.t. modulus $q_{t-k} = p^{k-1}$ for any $k = 1, \ldots, t$ as follows:

$$f(x)^{-1} = f^{(t-1)}(x)(1 - pu(x))^{-1} = f^{(t-1)}(x)(1 + pu(x) + p^2u(x)^2 + \cdots + p^{k-2}u(x)^{k-2}) \pmod{p^{k-1}}.$$

Then it holds that $f^{(i)} \pmod{q_{i+1}} = f^{(i+1)}$ for $i = 0, \ldots, t-1$. This means that to switch to a new level (and modulus) during homomorphic evaluation the public key we simply compute via modular reduction. The secret key f remains the same for all levels. Therefore, key switching is no longer needed. Also we no longer need to store a secret-key/public-key for each level of the circuit. With this approach we can still take advantage of modulus switching without having to pay for storage or key switching.

In the original scheme, the public key size is quadratically dependent on the number of the levels the instantiation can support. Also the number of evaluation keys needed in a level is dependent to the bit size of the modulus at that level, i.e. $\log q_i$. Having a polynomial size $n \log q_i$ at each level, the public key size can be written as

$$|\mathrm{PK}| = \sum_{i=0}^{t-2} n(\log q_i)^2.$$

In our modified scheme we only need the key for the first level $q_0 = p^t$ which is progressively reduced as the evaluation proceeds through the levels, and therefore

$$|\mathrm{PK}'| = n(\log q_0)^2.$$

In Sect. 7 we calculate the public key size for two settings which show drastic savings in memory use.

To understand the impact of this restriction on key generation and on the size of the key space we invoke an earlier result by Silverman [27]. In this study, Silverman analyzed the probability of a randomly chosen $f \in \mathrm{R}'_q = \mathbb{Z}_q[x]/\langle x^n - 1\rangle$ to be invertible in R'_q.

Theorem 1 ([27]) *Let $q = p^k$ be a power of a prime p, and let $n \geq 2$ be an integer with $\gcd(q, n) = 1$. Define $w \geq 1$ to be the smallest positive integer such that $p^w = 1 \bmod n$ and for each integer $d|w$, let*

$$\nu_d = \frac{1}{d} \sum_{e|d} \mu\left(\frac{d}{e}\right) \gcd(n, p^e - 1)$$

then

$$\frac{|\mathrm{R}_q'^*|}{|\mathrm{R}_q'|} = \prod_{d|w} \left(1 - \frac{1}{p^d}\right)^{\nu_d}.$$

Silverman [27] noted that the large class of noninvertible polynomials $f \in \mathrm{R}_q'$ such that $f(1) = 0$ can be avoided by "intelligently choosing" f. He further restricts the selection $f \in \mathrm{R}_q'$ such that $f(1) = 1$ and derives an approximation on the probability of picking an invertible f which simplifies for large and prime n as follows

$$\frac{|\mathrm{R}_q'^*(1)|}{|\mathrm{R}_q'(1)|} \approx 1 - \frac{n-1}{wp^w}.$$

Here $\mathrm{R}_q'(1) = \{f \in \mathrm{R}_q' \text{ s.t. } f(1) = 1\}$ and $\mathrm{R}_q'^*(1) = \{f \in \mathrm{R}_q' \text{ s.t. } f(1) = 1 \text{ and } \exists g \in \mathrm{R}_q' \text{ s.t. } gf = 1\}$. Note that the failure probability may be made negligibly small by picking appropriate p and n values.

In this paper we are working in a much simpler setting, i.e. $\mathrm{R}_q = \mathbb{Z}_q[z]/\langle \Phi_m(x)\rangle$. The uniform factorization of the cyclotomic polynomial $\Phi(x)$ allows us to adapt Silverman's analysis [27] and obtain a much simpler result. Assuming $\gcd(n, p) = 1$, the cyclotomic polynomial factors into equal degree irreducible polynomials $\Phi_m(x) = \prod_{i=1}^{\ell} F_i(x)$ over $\mathbb{Z}_p$, where $\deg(F_i(x)) = w$, $\ell = \phi(m)/w$ and $w \geq 1 \in \mathbb{Z}$ is the smallest integer satisfying $p^w = 1 \pmod{\phi(m)}$. Therefore

$$\mathbb{F}_p[x]/\langle \Phi_m(x)\rangle \cong \mathbb{F}_p[x]/\langle F_1(x)\rangle \times \cdots \times \mathbb{F}_p[x]/\langle F_\ell(x)\rangle \cong (\mathbb{F}_{p^w})^\ell$$

and for R_q we have $\nu_d = \ell$. With this simplification the probability of randomly picking an invertible $f \in \mathrm{R}_q$ given in Theorem 1 simplifies to

$$\frac{|\mathrm{R}_q^*|}{|\mathrm{R}_q|} = \frac{|\mathrm{R}_p^*|}{|\mathrm{R}_p|} = \left(1 - \frac{1}{p^w}\right)^\ell.$$

When p is large $|\mathrm{R}_q^*|/|\mathrm{R}_q| \approx 1 - \ell p^{-w}$.

With the new restriction imposed by the selection of the moduli we introduce a modified KeyGen procedure as follows.

4.3 Modified KeyGen

We use the chosen decreasing moduli $q_0 > q_1 > \cdots > q_{t-1}$ where $q_i = p^{t-i}$ for $i = 0, \ldots, t-1$. We further set the m^{th} cyclotomic polynomial $\Phi_m(x)$ as our polynomial modulus and set χ as a truncated discrete Gaussian distribution that is B-bounded. We sample u and g from distribution χ, set $f = 2u + 1$ and $h = 2gf^{-1}$ in ring $R_{q_0} = \mathbb{Z}_{q_0}[x]/\langle \Phi_m(x)\rangle$. We then sample, for $\tau = 0, \ldots, \lfloor \log q_0 \rfloor$, s_τ and e_τ from χ and publish evaluation key $\{\zeta_\tau\}_\tau^{(0)}$ where $\zeta_\tau^{(0)} = hs_\tau + 2e_\tau + 2^\tau f$ in R_{q_0}. Then using Lemma 3, we can evaluate rest of the evaluation keys for a level i by simply computing $\zeta_\tau^{(i)} = \zeta_\tau^{(0)} \bmod q_i$. Then set; secret key $\mathsf{sk} = (f)$ and public key $\mathsf{pk} = (h, \zeta_\tau)$.

4.4 Optimizing relinearization

In homomorphic circuit evaluation using LTV by far the most expensive operation is relinearization. Therefore, it becomes essential to optimize relinearization as much as possible. Recall that the relinearization operation computes a sum of encrypted shifted versions

of a secret key $f(x)$ and polynomials $\tilde{c}_\tau(x)$ with coefficients in $\mathbb{F}_2$ extracted from the ciphertext c:

$$\tilde{c}(x) = \sum_\tau \zeta_\tau(x) \cdot \tilde{c}_\tau(x).$$

For simplicity we dropped the level indices in superscripts. The ciphertext $\zeta_\tau(x) \in \mathrm{R}_q[x]/\langle \Phi(x) \rangle$ values are full size polynomials with coefficients in R_q and do shrink in size over the levels of evaluation after each modulus switching operation. In contrast $\tilde{c}_\tau(x) \in \mathbb{F}_2[x]/\langle \Phi(x) \rangle$ where τ ranges $\log(q)$. We may evaluate the summation, by scanning the coefficients of the current $\tilde{c}_\tau(x)$ and conditionally shifting and adding $\zeta_\tau(x)$ to the current sum depending on the value of the coefficient. With this approach the computational complexity of relinearization becomes $\mathcal{O}(n \log(q))$ polynomial summations or $\mathcal{O}(n^2 \log(q))$ coefficient, i.e. $\mathbb{Z}_q$, summations. This approach is useful only for small n.

In contrast, if we directly compute the sum after we compute the products we obtain a more efficient algorithm. The number of polynomial multiplications is $\mathcal{O}(\log(q))$ each having a complexity of $\mathcal{O}(n \log(n) \log\log(n))$ with the Schönhage Strassen algorithm [28]. The algorithm simply uses Number Theoretic Transform (NTT) and completes the polynomial multiplication in three steps; conversion of the polynomials to NTT form, digit-wise multiplications, conversion from NTT to polynomial form. After the multiplications, coefficient additions require $\mathcal{O}(n \log(q))$ operations. The total complexity of relinearization becomes $\mathcal{O}(n \log(n) \log\log(n) \log(q))$ coefficient operations.

Another optimization technique is to store the polynomials $\zeta_\tau(x)$ in NTT form. This eliminates the time needed for the conversions of $\zeta_\tau(x)$ at beginning of each multiplication operation. Furthermore, polynomial additions are also performed in NTT form to eliminate NTT^{-1} conversions to polynomial form. Representing the precomputed NTT form of $\zeta_\tau(x)$ as $\zeta'_\tau(x)$ we can rewrite the relinearization operations as follows:

$$\tilde{c}(x) = \mathrm{NTT}^{-1}\left[\sum_\tau \zeta'_\tau(x) \cdot \mathrm{NTT}[\tilde{c}_\tau(x)]\right].$$

With this final optimization, we eliminate $2/3^{rds}$ of the conversions in each relinearization and obtain nearly 3 times speedup.

5 Coping with noise

In this section, we describe our approach in managing the growth of noise over the homomorphic evaluation of levels of the circuit. The accumulation of noise from the evaluations of additions adds very little noise compared to that contributed by multiplication. Therefore, as long as we have a reasonably balanced circuit we can focus only on multiplications. Furthermore, in our analysis we focus on noise growth with regards to its effect on the correctness of the scheme. Our goal is to minimize the computational burden, i.e. minimize parameters q and n, such that the scheme still correctly decrypts with very high probability.

Consider two plaintexts $m_1, m_2 \in \{0, 1\}$ and parameters $g, s \in \chi$ encrypted using a single user (single key) with no modulus switching specialization of the LTV scheme. The secret key is $f = 2f' + 1$ where $f' \in \chi$. the product of two given ciphertexts $c_1 = E(m_1) = hs_1 + 2e_1 + m_1$ and $c_2 = E(m_2) = hs_2 + 2e_2 + m_2$ yields:

$$c_1 c_2 = h^2 s_1 s_2 + h(s_1 m_2 + s_2 m_1) + 2[h(s_1 e_2 + s_2 e_1) + e_1 m_2 + e_2 m_1 + 2e_1 e_2] + m_1 m_2.$$

To decrypt the resulting ciphertext we compute:

$$f^2 c_1 c_2 = 4g^2 s_1 s_2 + 2gf(s_1 m_2 + s_2 m_1) + 2\big[2gf(s_1 e_2 + s_2 e_1) + f^2 e_1 m_2 + f^2 e_2 m_1 + 2f^2 e_1 e_2\big] + f^2 m_1 m_2.$$

The accumulative noise in the ciphertext should satisfy the following condition and avoid any wraparound to prevent corruption in the message coefficients during decryption:

$$q/2 > 4n^3B^4 + 4n^3B^3(2B+1) + 8n^3B^3(2B+1) + 8n^3B^2(2B+1)^2 + n^3B^2(2B+1)^2 > n^3(64B^4 + 48B^3 + 9B^2).$$

Note that this is the *worst case* behavior of the norm and therefore decryption will work for most ciphertexts even with a somewhat smaller q.

5.1 Modulus switching

It will be impractical to evaluate a deep circuit, e.g. AES, using this approach since the norm grows exponentially with the depth of the circuit. To cope with the growth of noise, we employ *modulus switching* as introduced in [5]. For this, we make use of a series of decreasing moduli $q_0 > q_1 > \cdots > q_t$; one modulus per level. Modulus switching is a powerful technique that exponentially reduces the growth of noise during computations. The modulus switching operation is done for a ciphertext c is shown as $c_{new} = \lceil c \cdot q_{i+1}/q_i \rfloor_2$. The ceil-floor represents the rounding operation and subscript 2 represents matching the parities of c and c_{new} in modulus 2. The modulus switching operation is performed by first multiplying the coefficients by $q_{i+1}/q_i \approx \kappa$ and rounding them to the nearest integer. Later, a parity correction operation performed by adding a parity polynomial P_i. As before, for $c_1 = E(m_1) = hs_1 + 2e_1 + m_1$ and $c_2 = E(m_2) = hs_2 + 2e_2 + m_2$ the product of the two ciphertexts gives

$$c_1 c_2 = h^2 s_1 s_2 + h(s_1 m_2 + s_2 m_1) + 2[h(s_1 e_2 + s_2 e_1) + e_1 m_2 + e_2 m_1 + 2e_1 e_2] + m_1 m_2.$$

After modulus switching, i.e. multiplication by $q_1/q_0 \approx \kappa$ and correction of parities symbolized by $P_i \in \mathbb{D}_{\mathbb{Z}^n,\sigma}$ we obtain

$$c_1 c_2 \kappa + P_1 = \big[h^2 s_1 s_2 + h(s_1 m_2 + s_2 m_1) + 2[h(s_1 e_2 + s_2 e_1) + e_1 m_2 + e_2 m_1 + 2e_1 e_2 + m_1 m_2\big]\kappa + P_1.$$

After i levels the ciphertext products (for simplicity assume $c = c_1 = \ldots = c_{2^i}$) where each multiplication is followed by modulus switching and parity corrections (symbolized by the P_i) will be

$$c^{2^i} = \big(\ldots\big(\big(c^2\kappa + P_1\big)^2\kappa + P_2\big)^2 \ldots \kappa + P_{2^i}\big).$$

We may decrypt the result as follows:

$$\begin{aligned} c^{2^i} f^{2^i} &= \big(\ldots\big(\big(c^2\kappa + P_1\big)^2\kappa + P_2\big)^2 \ldots \kappa + P_{2^i}\big) f^{2^i} \\ &= \big(\ldots\big(\big(c^2\kappa + P_1\big)^2\kappa + P_2\big)^2 \ldots \kappa + P_{2^i}\big) f^{2^i}. \end{aligned}$$

The correctness condition becomes $||c^{2^i} f^{2^i}||_\infty < q/2$. Note that due to the final multiplication with the f^{2^i} term we still have exponential growth in the norm with the circuit depth. Therefore, we need one more ingredient, i.e. *relinearization* [9], to force the growth into a linear function of the circuit depth. Intuitively, relinearization achieves to linearize the growth by homomorphically multiplying the current ciphertext by f right *before* modulus switching.

5.2 Relinearization and modulus switching

After each multiplication level we implement a relinearization operation which keeps the power of f in the ciphertext under control and reduces the chances of wraparound before decryption. Assume we homomorphically evaluate a simple d-level circuit $\mathcal{C}(m) = m^{2^d}$ by computing repeated squaring, relinearization and modulus switching operations on a ciphertext c where $||c||_\infty = B_i$. Recall that for relinearization we compute

$$\tilde{c}^{(i)}(x) = \sum_\tau \zeta_\tau^{(i)}(x)\tilde{c}_\tau^{(i-1)}(x),$$

where each $\zeta_\tau^{(i)}(x)$ is of the form $\zeta_\tau^{(i)}(x) = h^{(i)}s_\tau^{(i)} + 2e_\tau^{(i)} + 2^\tau f^{(i-1)}$ in $R_{q_{i-1}}$. Substituting this value we obtain

$$\begin{aligned}\tilde{c}^{(i)}(x) &= \sum_\tau \left[h^{(i)}s_\tau^{(i)} + 2e_\tau^{(i)} + 2^\tau\left(f^{(i-1)}\right)\right]\tilde{c}_\tau^{(i-1)}(x)\\ &= \sum_\tau \left[h^{(i)}s_\tau^{(i)} + 2e_\tau^{(i)}\right]\tilde{c}_\tau^{(i-1)}(x) + \sum_\tau 2^\tau\left(f^{(i-1)}\right)\tilde{c}_\tau^{(i-1)}(x).\end{aligned}$$

Since we are only interested in bounding the growth of noise we assume $s_\tau^{(i)} = s \in \chi$, $g_\tau^{(i)} = g \in \chi$ and $e_\tau^{(i)} = e \in \chi$ and drop unnecessary indices from here on:

$$\begin{aligned}\tilde{c}^{(i)} &= \sum_\tau (hs + 2e + 2^\tau f)\tilde{c}_\tau^{(i-1)}\\ &= \sum_\tau (hs + 2e)\tilde{c}_\tau^{(i-1)} + \sum_\tau 2^\tau f\tilde{c}_\tau^{(i-1)}\\ &= \sum_\tau (2gf^{-1}s + 2e)\tilde{c}_\tau^{(i-1)} + f\tilde{c}^{(i-1)}\\ &= \sum_{\tau\in[\log(q)]} (2gf^{-1}s + 2e)\tilde{c}_\tau^{(i-1)} + \tilde{c}^{(i-1)}f.\end{aligned}$$

Also factoring in the modulus switching and parity correction steps and substituting $\tilde{c}^{(i-1)} = c^2$ we obtain the reduced noise ciphertext $\tilde{c}'$ as

$$\tilde{c}' = \left(\sum_{\tau\in[\log(q)]} (2gf^{-1}s + 2e)\tilde{c}_\tau + c^2 f\right)\kappa + P,$$

where $P \in \{0, 1\}$ represents the parity polynomial. The distribution (and norm) of the left summand in the inner parenthesis is constant over the levels. To simplify the equation we use the shorthand $X_0 = \sum_{\tau\in[\log(q_i)]}(2gf^{-1}s + 2e)\tilde{c}_\tau$ where the index is used to indicate the level. Assume we use Y_i to label the ciphertext (output) of evaluation level i then

$$Y_1 = \left(fc^2 + X_0\right)\kappa + P_1\ .$$

Assume we continue this process, i.e. squaring, relinearization, modulus switching and parity correction for d levels and then decrypt by multiplying the resulting ciphertext by f we obtain:

$$Y_i = \left(fY_{i-1}^2 + X_{i-1}\right)\kappa + P_i \quad , \quad \text{for } i = 1, \ldots, d-1.$$

To decrypt Y_{d-1} we need $||Y_{d-1}f||_\infty < q/2$. Now first note that

$$\begin{aligned} Y_i f &= \left[\left(f Y_{i-1}^2 + X_{i-1}\right)\kappa + P_i\right] f \\ &= \left((Y_{i-1} f)^2 + X_{i-1} f\right)\kappa + P_i f. \end{aligned}$$

Therefore, $||Y_i f||_\infty \leq ||(Y_{i-1} f)^2||_\infty \kappa + ||X_{i-1} f||_\infty \kappa + ||P_i f||_\infty$. Also note that

$$\begin{aligned} ||f X_i||_\infty &= ||\sum_\tau (2g s_\tau + 2e_\tau f)\tilde{c}_\tau||_\infty \\ &\leq ||\sum_\tau 2g s_\tau \tilde{c}_\tau||_\infty + ||\sum_\tau 2e_\tau f \tilde{c}_\tau||_\infty. \end{aligned}$$

Since $||f||_\infty = 2B + 1$, $||\tilde{c}_\tau||_\infty = ||s_\tau||_\infty = ||e_\tau||_\infty = ||g||_\infty \leq B$ and all polynomials have at most degree n it follows that

$$\begin{aligned} ||f X_i||_\infty &\leq (2n^2 B^3 + 2n^2 B^2 (2B + 1)) \log(q_i) \\ &\leq n^2 (6B^3 + 2B^2) \log(q_i). \end{aligned}$$

Now let B_i denote an upper bound on the norm of a decrypted ciphertext entering level i of leveled circuit, i.e. $B_i \geq ||f Y_i||_\infty$. Using the equation $B_i \geq ||f Y_i||_\infty$ we can set the norm $||(Y_{i-1} f)^2||_\infty$ as

$$\begin{aligned} ||(Y_{i-1} f)^2||_\infty &\leq n||(Y_{i-1} f)||_\infty \cdot ||(Y_{i-1} f)||_\infty \\ &\leq n B_{i-1}^2. \end{aligned}$$

The norm of the output grows from one level to the next including multiplication (squaring with our simplification), relinearization and modulus switching as follows:

$$B_i \leq \left[n B_{i-1}^2 + n^2 (6B^3 + 2B^2) \log(q_i)\right]\kappa + n(2B + 1)\ . \tag{2}$$

Notice the level independent (fixed) noise growth term on the right summand of the recursion. In practice, κ needs to be chosen so as to *stabilize* the norm over the levels of computation, i.e. $B_1 \approx B_2 \approx \cdots \approx B_{d-1} < q_{d-1}/2$. Finally, we can make the accounting a bit more generic by defining circuit parameters v_i which denote the maximum number of ciphertext additions that take place in evaluation level i. With this parameter we can bound the worst case growth simply by multiplying any ciphertext that goes into level $i + 1$ by v_i as follows

$$B_i \leq \left[v_i^2 n B_{i-1}^2 + n^2 (6B^3 + 2B^2) \log(q_i)\right]\kappa + n(2B + 1)\ . \tag{3}$$

5.3 Average case behavior

In our analysis so far we have considered worst case behavior. When viewed as a distribution, the product norm $||ab||_\infty$ will grow much more slowly and the probability that the norm will reach the worst case has exponentially small probability. To take advantage of the slow growth we can instead focus on the growth of the standard deviation by modeling each coefficient of a and b as a scaled continuous Gaussian distribution with zero mean and deviation $r = B$. The coefficients of the product $(ab)_i = \sum_{i=0,\ldots,n-1} a_i b_{n-1-i}$, behave as drawn from a scaled chi-square distribution with $2n$ degrees of freedom, i.e. $\chi^2(2n)$. To see this just note each coefficient product can be rewritten as $a_i b_{n-1-i} = \frac{1}{4}(a_i + b_{n-1-i})^2 - \frac{1}{4}(a_i - b_{n-1-i})^2$. As n becomes large $\chi^2(2n)$ becomes close to an ideal Gaussian distribution with variance $4n$. Thus $r((ab)_i) \approx \sqrt{n} B^2$ for large n. Therefore, a sufficiently good approximation of the expected norm may be obtained by replacing n with $\sqrt{n}$ in Eq. 3 as follows

$$B_{i,\text{avg}} \approx \left[v_i \sqrt{n} B_{i-1,\text{avg}}^2 + n(6B^3 + 2B^2) \log(q_i)\right]\kappa + \sqrt{n}(2B + 1)\ .$$

Table 4 Worst case and average case number of bits $\log(1/K)$ required to cut to correctly evaluate a pure multiplication circuit of depth L with $B = 2$ and $\alpha = 6$ for n and q chosen such that $\delta(n, q) = 1.0066$

$\log(n)$	$\log(q)$	Worst case		Average case	
		$\log(1/K)$	$\#L$	$\log(1/K)$	$\#L$
12	155	36	3	33	3
13	311	37	7	33	8
14	622	39	14	34	17
15	1244	40	30	34	35
16	2488	42	58	35	70
17	4976	44	112	35	141

For practical values of n and small B the left-hand-side dominates the other terms in the equation. Further simplifying we obtain

$$B_{i,\text{avg}} \approx \left[\nu_i \sqrt{n} B_{i-1,\text{avg}}^2 + n(6B^3 + 2B^2)\log(q_i)\right]\kappa \ . \tag{4}$$

Assuming nearly fixed $\nu_i \approx \nu$, if we set $1/\kappa = \epsilon\left[\nu\sqrt{n}B_{i-1,\text{avg}}^2 + n(6B^3+2B^2)\log(q_0)\right]$ for a small constant $\epsilon > 1$ we can stabilize the growth of the norm and keep it nearly constant over the levels of the evaluation. Initially recursive computation will start with $B_{0,\text{avg}} = 2$ and the noise will grow in a few steps until it is stable. Our experiments showed that after 5–6 levels of computation the noise stabilizes between $2^{10} < B_{i,\text{avg}} < 2^{15}$ for latticec of dimensions $2^{12} < n < 2^{17}$. By taking $B_{i,\text{avg}} = 2^{12}$ and $\nu = 1$ we tabulated the cutting sizes and attainable levels of homomorphic evaluation in Table 4.

We can simplify the noise equation further to gain more insight on the noise growth and subsequently on how the modulus q and the dimension n will be affected. Fix $B = 2$ and assume we are interested in evaluating a depth t circuit and therefore $q_0 = p^{t+1}$. Also since with our $q_0 = p^{t+1}$ specialization $1/\kappa \approx p$ and since $p \gg \nu$ neglecting $\log(\nu)$ we can simplify our noise estimate as follows:

$$p \approx \nu\sqrt{n}B_{i-1,\text{avg}}^2 + 56n(t+1)\log p \ .$$

This nonlinear equation displays the relationship between the chosen dimension n and depth of circuit we wish to support and the number of bits we need to cut in each level. However, p and n are not independent since n and $q = p^{t+1}$ are tied through the Hermite factor $\delta = (\sqrt{q}/4)^{1/(2n)} = (\sqrt{p^{t+1}}/4)^{\frac{1}{2n}}$ and $p = (4\delta^{2n})^{2/(t+1)}$. Substituting p yields

$$\begin{aligned}(4\delta^{2n})^{2/(t+1)} &\approx \nu\sqrt{n}B_{i-1,\text{avg}}^2 + 56n(t+1)\log(4\delta^{2n})^{2/(t+1)} \\ &\approx \nu\sqrt{n}B_{i-1,\text{avg}}^2 + 56n(t+1)2/(t+1)(\log 4 + \log(\delta^{2n})) \\ &\approx \nu\sqrt{n}B_{i-1,\text{avg}}^2 + 112n(2 + 2n\log(\delta)).\end{aligned}$$

By taking the logarithm and fixing ν and the security level δ we see that $t \sim \mathcal{O}(n/\log(n))$.

5.4 Failure probability

Equation 4 tells us that we can use a much smaller q than that determined by the worst case bound in Eq. 3 if are willing to accept a small decryption failure probability at the expense of a small margin. The failure probability is easily approximated. If we set $q/2 > \alpha B_{\text{avg}}$ where $\alpha > 1$ captures the margin, then $\alpha B_{\text{avg}}/\sigma$ determines how much of the probability space we cover in a Gaussian distribution $\mathcal{N}(\mu = 0, \sigma)$. The probability for the norm

of a single coefficient to exceed a preset margin $\alpha\sigma$ becomes $\mathsf{Prob}\left[||(ab)_i||_\infty > \alpha\sigma\right] \approx 1 - \mathsf{erf}\left(\alpha/\sqrt{2}\right)$ where erf denotes the error function. For the entire product polynomial we can approximate the worst case probability by assuming independent product coefficients as $\mathsf{Prob}\left[||ab||_\infty > \alpha\sigma\right] \approx 1 - \mathsf{erf}\left(\alpha/\sqrt{2}\right)^n$. Having dependent coefficients (as they really are) will only improve the success probability. For instance, assuming $n = 2^{14}$ and $\sigma = B$ with a modest margin of $\alpha = 7$ we obtain a reasonably small failure probability of 2^{-60}.

6 Evaluating AES using LTV-FHE

Here we briefly summarize the AES circuit we use during evaluation. The homomorphic evaluation function takes as input the encrypted AES evaluation keys, and the description of the AES circuit as input. All input bits are individually encrypted into separate ciphertexts. We do *not* use byte-slicing in our implementation. Our description follows the standard definition of AES with 128-bit keys where each of the 10 rounds are divided into four steps: AddRoundKey, ShiftRows, MixColumns and SubBytes:

6.1 AddRoundKey

The round keys are derived from the key through an expansion algorithm and encrypted to be given alongside the message beforehand. The first round key is added right after the computation starts and the remaining round keys are added at the end of each of their respective rounds during evaluation. Therefore, each round key is prepared for the level during which it will be used. As we will shortly show each AES level requires 4 multiplication levels. Therefore the round key for level i is computed in $\mathrm{R}_{q_{4i}}$ for $0 \leq i \leq 10$. Adding a round key is a simple XOR operation performed by addition of the ciphertexts. Since round keys are fresh ciphertexts, the added noise is limited to a single bit.

6.2 ShiftRows

The shifting of rows is a simple operation that only requires swapping of indices trivially handled in the code. This operation has no effect on the noise.

6.3 MixColumns

The Mix Column operation is a 4×4 matrix multiplication with constant terms in $GF(2^8)$. The multiplication is between a byte and one of the constant terms of $\{x+1, x, 1\}$ with modulo $(x^8 + x^4 + x^3 + x + 1)$. These products are evaluated by simple additions and shifts as follows:

$$
\begin{aligned}
(b_7b_6b_5b_4b_3b_2b_1b_0) &\xrightarrow{\times 1} (b_7b_6b_5b_4b_3b_2b_1b_0)\\
(b_7b_6b_5b_4b_3b_2b_1b_0) &\xrightarrow{\times x} (b_6b_5b_4b_3b_2b_1b_0b_7) \oplus (000b_7b_70b_70)\\
(b_7b_6b_5b_4b_3b_2b_1b_0) &\xrightarrow{\times (x+1)} (b_7b_6b_5b_4b_3b_2b_1b_0) \oplus (b_6b_5b_4b_3b_2b_1b_0b_7) \oplus (000b_7b_70b_70)
\end{aligned}
$$

Once the multiplication of the rows are finished, 4 values are added to each other. The addition operations add a few bits of noise.

6.4 SubBytes

The SubBytes step or the S–Box is the only place where we require homomorphic multiplications and Relinearization operations. An S–Box lookup in AES corresponds to a finite field inverse computation followed by the application of an affine transformation; i.e., $s = Mb^{-1} \oplus B$. M is a $\{0, 1\}$ matrix and B is constant vector for the affine transformation which may are simply realized using addition operations between ciphertexts. The time consuming part of the S–Box is the evaluation of inversion operation. In [29], the authors introduced a compact design for computing the inverse. The input byte in $GF(2^8)$ is converted using an isomorphism into a tower field representation, i.e. $GF(((2^2)^2)^2)$, which allows much more efficient inversion. This conversion to/from tower field representation is achieved by simply multiplying with a conversion matrix with $\{0, 1\}$ coefficients. The inversion operation can be written as: $b^{-1} = X(X^{-1}b)^{-1}$. With this modificaiton the operations in the SubBytes step can be expressed as $s = M(X(X^{-1}b)^{-1}) \oplus B$. The conversion matrices X^{-1} and the matrix product MX are given as follows:

$$X^{-1} = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 \end{pmatrix} \quad \text{MX} = \begin{pmatrix} 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \end{pmatrix} \quad (5)$$

With tower field representation, the 8-bit S–Box substitution requires 4 (multiplication) levels of circuit evaluation. The full 10 round 128 bit-AES block homomorphic evaluation requires the evaluation of a depth 40 circuit.

7 Implementation results

We implemented the customized LTV scheme with the optimizations summarized in Sect. 4 using Shoup's NTL library version 6.0 [22] compiled with the GMP 5.1.3 package. The implementation batches bits into ciphertexts using CRT applied using a cyclotomic modulus polynomial $\Phi_m(x)$ where $\deg(\Phi) = n$. Also note that we fix χ with $B = 2$ and $r' = 1.55$. The evaluation functions supports homomorphic additions and multiplication operations. Each multiplication is followed by a Relinearization operation and modulus switching in our implementation. After homomorphic evaluation the results may be recovered from the message slots using remainder computations as usual.

7.1 Homomorphic AES evaluation

Using the LTV primitives we implemented the depth 40 AES circuit described in Sect. 6. The AES S-Box evaluation, is completed using 18 Relinearization operations and thus 2880 Relinearizations are needed for the full AES.

We ran the AES evaluation for two choices of parameters:

– Polynomial degree of $n = 27{,}000$ with a modulus of size $\log(q) = 1230$ and Hermite factor $\delta = 1.0078$ (low security setting). For a error margin of $\alpha \approx 8$ and number of additions per AES level of $\nu \approx 100$ if we cut $\log(p) \approx \log(1/K) = 30$ bits at each level

Table 5 The two settings under which we evaluated AES and timing results on Intel Xeon @ 2.9 GHz

n	$\log(q_0)$	δ	$\log(1/K)$	Message slots	Total time	Time/block
27,000	1230	1.0078	30	1800	25 h	50 s
32,768	1271	1.0067	31	2048	29 h	51 s

Eq. 4 tells us that the noise will stabilize around 12.8 bits. For $\alpha \approx 8$ we obtain an error probability of 2^{-41} per ciphertext. Under these parameters the total running time of AES is 25 h. Since we batched with 1800 message slots we obtain 50 s evaluation time per block.

– Polynomial degree set as $n = 32{,}768$ with modulus size $\log(q) = 1271$ and Hermite factor $\delta = 1.0067$. For a error margin of $\alpha \approx 8$ and number of additions of $\nu \approx 100$ if we cut $\log(1/K) = 31$ bits at each level the noise will stabilize around 12.6 bits. The total running time is 29 h resulting in 51 s per block encryption with 2048 message slots.

Table 5 summarizes the parameters for the two settings and the timing results.

When we utilize the two Hermite factor for the security analysis using the formula in [17], i.e. $1.8/\log\delta - 110$, we compute 77 and 50-bits security for $\delta = 1.0067$ and $\delta = 1.0078$ respectively. If we use the Hermite factor parameters from the Table 3 we have a security level of larger than 128- bits for $\delta = 1.0067$ and we have security level of between 80 and 128-bits for $\delta = 1.0078$.

7.2 Memory requirements

In the implementation we are taking advantage of the reduced public key size as described in Sect. 4. To support a 40 level AES circuit evaluation with the original scheme in [9] for the two settings outlined above we would need to store public keys of size 67 and 87 GBytes, respectively. The optimized scheme reduces the public keys to 4.75 and 6.15 GBytes. This demonstrates the effectiveness of the optimization. We can perform the evaluation on common machines with less than 16 Gbytes memory. Table 6 summarizes the public key sizes for the two chosen parameter settings with and without the public key optimization.

Since our server has more memory, to speed up the relinearization operations we keep the public keys in the NTT domain requiring 12.2 Gbytes and 13.1 Gbytes, respectively. Also we keep all the keys for a round (4 levels at a time) in memory. Keeping the public keys in the NTT domain improved the speed of relinearizations by about 3 times. Since relinearizations amount to about 70 % of the time we gained an overall speedup of 2.5 times in AES evaluation.

Table 6 Sizes of public-key in various representations with and without optimization for the two selected parameter settings

Representation	Original (GBytes)		Optimized (GBytes)		AES speedup
Polynomial	67	87	4.75	6.13	1
NTT	172	184	12.2	13.1	2.5

8 Comparison

In the following, we will briefly compare our implementation with other homomorphic encryption libraries and implementations that have appeared in the literature.

8.1 GHS-AES

When compared to the BGV style leveled AES implementation by Gentry, Smart, Halevi (GHS) [6]; our implementation runs 47 times faster than the bit-sliced and 5.8 times faster than the byte-sliced implementation. Our implementation is more comparable to the bit-sliced version since we did not customize our software library to more efficiently evaluate AES in order to keep it generic. While we also use optimizations such as modulus switching, and batching the two implementations differ in the way they handle noise. In the GHS FHE implementation take a more fine grain approach to modulus switching, by cutting the noise even after constant multiplications, additions and shifting operations. Depending on the implementation is bit-sliced or byte-sliced, the number of levels ranges between 50 to 100 where in each level 18–20 bits are cut. In the presented work we only cut the modulus after multiplications and therefore we have a fixed 40 levels with 30–31 bits cut per level.

8.2 GHS-AES (updated implementation)

In the final revision of this manuscript, Gentry, Smart, Halevi (GHS) [30] published significantly improved runtime results. Compared to the earlier implementation, the authors used the latest version of the HElib library. They managed to decrease the number of levels in the AES circuit but had to increase the number of bits cut in each level to manage the additional noise growth. The new result only reports a SIMD version. Two variations of the implementation are reported: one with bootstrapping and one without bootstrapping. In the bootstrapping version, 180 blocks are processed at a time which achieves a 6 s amortized runtime using only 23 circuit levels. In the non-bootstrapping implementation, they process 120 blocks at a time and achieve 2 s per block runtime for 40 circuit levels. Also the design only requires around 3.5GB of memory.

8.3 MS-AES

Very recently Mella and Susella (MS) revisited the homomorphic AES computation of GHS AES with some optimizations in [31]. They used the homomorphic encryption library HElib [32] that is based on BGV style homomorphic encryption. The authors managed to reduce the number of levels to 4 per AES round to a total of 40 levels for the entire AES circuit evaluation. They implemented two versions of AES; byte-sliced and packed. In byte-sliced version they were able to pack 12 AES evaluations with 16 ciphertexts, whereas in the packed version they use 1 ciphertext and were not able to pack multiple AES evaluations. The byte-sliced implementation has an execution time of 2 h 47 min with an amortized time of 14 min. For the packed implementation the total execution time is only 22 min. In terms of execution time the MS implementation is the fastest with 22 min, and 2 h 47 min runtime compared to ours with 29 h, and to GHS with 36 and 65 h runtime. However in the amortized case, we are the fastest with 51 s runtime compared to MS with 22 and 14 min, and to GHS with 40 and 5 min runtimes.

Table 7 Number of multiplications and evaluation key sizes for constructions in [12] and ours

	YASHE	YASHE'	Ours
# of Multiplications	ℓ	ℓ^3	$\log q$
Eval. key size	$\ell n \log q$	$\ell^3 n \log q$	$n(\log q)^2$

8.4 YASHE

In another recent work, Bos et al. introduced a scale-invariant implementation of LTV called YASHE [12]. The authors select a word size w for the homomorphic computations, and create $\ell = \lfloor \log_w q \rfloor + 2$ vectors to support the radix w operations. The construction has evaluation key that is element of R^{ℓ^3} for a polynomial ring R. This means that, any increase in the evaluation depth and size of q will cause cubic growth in the evaluation key size. In order to overcome large growth the authors modified the scheme called YASHE' to reduce the key size and achieved it to be an element of R^{ℓ}. Also they are able to reduce the complexity of the key-switching operation from ℓ^3 multiplications to ℓ.

Our variant of LTV is closer to YASHE' in terms of complexity. The computations and memory requirements grow linearly with the evaluation depth, i.e. $\log q$. Furthermore, as we progress through the levels with modulus switching technique, the homomorphic evaluation accelerates since the operands shrink which is not the case in scale–invariant with fixed run time. The YASHE scheme overcomes the negative effect by selecting a larger word size w, e.g. they can set $w = 32$ and achieve a high performance boost. In our case, our scheme is constructed with bit operations. Thus it is harder to eliminate the negative effects of bit size growth. The number of multiplications and evaluation key sizes are given in Table 7. The YASHE authors give implementation results only for the small case. They set the ring $R = \mathbb{Z}[x]/(X^{4096} + 1)$, the prime q as a 127-bit number and the word size w as 2^{32}. In the implementation key-switching takes 31 ms. Using the same parameters our relinearization takes about 139 ms when q equals to 125-bit number. At the smallest bit size, q is equal to 25-bit number and it takes 20 ms to complete relinearization. For the given case, our scheme seem to be slower with an average of 80 ms run time. Of course this is a single setting and more experiments with various settings should be studied to see how the timings are affected.

8.5 GPU implementation of LTV

Dai et al. recently reported a GPU implementation of our modified LTV scheme on an NVIDIA GeForce GTX 690 graphics cards in [33]. The authors developed a custom GPU CUDA library to support fast discrete Fourier transform based arithmetic for large degree polynomials. This fast arithmetic library later is used to accelerate the homomorphic multiplication and relinearization operations. The authors achieved a 4.15 h batched AES evaluation resulting in a 7.3 s per block amortized running time. This implementation achieves a 7.6 times speedup over our CPU implementation.

8.6 Windowed implementation of LTV

Recently, Öztürk et al. reported [34] a windowed implementation of the LTV implementation presented in this paper. In the design, the evaluation keys are constructed using powers of a word w instead of powers of 2. This decreases the number of evaluation keys from $\log q$ to $\log q/w$ where $\log q$ is the bit length of the modulus. With this additional optimization, the

authors achieve a 4.4 times speedup and reduce the per AES block amortized running time to 12.6 s.

9 Conclusion

In this work, we presented a customized leveled implementation of the NTRU based LTV homomorphic encryption scheme. We introduced a number of optimizations to obtain an efficient bit-sliced and batched implementation. We analyzed noise growth for increasing circuit depths and developed a simple formula that allows one to determine parameter sizes to support arbitrary depth circuits efficiently. Furthermore, we specialized the modulus in a way that allows us to drastically reduce the public key size while retaining the ability to apply modulus reduction and switching through the levels of evaluation. The reduced public key size makes it possible to evaluate deep circuits such as the AES block cipher on common (non-server) computing platforms with a reasonable amount of memory.

To expose the performance of the LTV, we homomorphically evaluated the full 10 round AES circuit in 29 h with 2048 message slots yielding a 51 s per AES block evaluation making it 47 times faster than the generic bit-sliced implementation, 5.8 times faster than the AES customized byte sliced BGV implementation by Gentry, Halevi and Smart. In the final stages of the revision of this manuscript Gentry, Halevi and Smart introduced an updated version of their implementation with a significant speedup achieved via a highly optimized custom SIMD implementation. This suggests that our implementation can be further sped up via the SIMD evaluation strategy.

Acknowledgments We would like to thank Jeffrey Hoffstein for pointing us to Coppersmith and Shamir's paper [20], and for helpful discussions to William J. Martin on the LTV scheme and to Joppe W. Bos and Michael Naehrig for clarifying the YASHE scheme. This work was in part supported by the NSF-CNS Awards #1117590 and #1319130.

References

1. Gentry C.: Fully homomorphic encryption using ideal lattices. In: Proceedings of the Forty-First Annual ACM Symposium on Theory of Computing, Ser. STOC '09, pp. 169–178. ACM, New York (2009).
2. Rivest R., Adleman L., Dertouzos M.: On Data Banks and Privacy Homomorphisms, pp. 169–177. Academic Press, New York (1978).
3. Gentry C., Halevi S.: Implementing gentrys fully-homomorphic encryption scheme. In: Paterson K. (ed.) Advances in Cryptology (EUROCRYPT 2011). Lecture Notes in Computer Science, vol. 6632, pp. 129–148. Springer, Berlin (2011).
4. Wang W., Hu Y., Chen L., Huang X., Sunar B.: Accelerating fully homomorphic encryption using GPU. In: High Performance Extreme Computing (HPEC), Sept 2012, pp. 1–5 (2012).
5. Brakerski Z., Gentry C., Vaikuntanathan V.: (leveled) fully homomorphic encryption without bootstrapping. In: Proceedings of the 3rd Innovations in Theoretical Computer Science Conference (ITCS '12), pp. 309–325. ACM, New York (2012).
6. Gentry C., Halevi S., Smart N.: Homomorphic evaluation of the AES circuit. In: Safavi-Naini R., Canetti R. (eds.) Advances in Cryptology (CRYPTO 2012). Lecture Notes in Computer Science, vol. 7417, pp. 850–867. Springer, Berlin (2012). doi:10.1007/978-3-642-32009-5_49.
7. Gentry C., Halevi S., Smart N.: Fully homomorphic encryption with polylog overhead. In: Pointcheval D., Johansson T. (eds.) Advances in Cryptology (EUROCRYPT 2012). Lecture Notes in Computer Science, vol. 7237, pp. 465–482. Springer, Berlin (2012). doi:10.1007/978-3-642-29011-4_28.
8. Smart N., Vercauteren F.: Fully homomorphic SIMD operations. Des. Codes Cryptogr. **71**(1), 57–81, (2014). doi:10.1007/s10623-012-9720-4.

9. López-Alt A., Tromer E., Vaikuntanathan V.: On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption. In: Proceedings of the 44th Annual ACM Symposium on Theory of Computing (STOC '12), pp. 1219–1234. ACM, New York (2012).
10. Hoffstein J., Pipher J., Silverman J.: NTRU: a ring-based public key cryptosystem. In: Buhler J. (ed.) Algorithmic Number Theory. Lecture Notes in Computer Science, vol. 1423, pp. 267–288. Springer, Berlin. doi:10.1007/BFb0054868.
11. Stehl D., Steinfeld R.: Making NTRU as secure as worst-case problems over ideal lattices. In: Paterson K. (ed.) Advances in Cryptology (EUROCRYPT 2011). Lecture Notes in Computer Science, vol. 6632, pp. 27–47. Springer, Berlin (2011). doi:10.1007/978-3-642-20465-4_4.
12. Bos J., Lauter K., Loftus J., Naehrig M.: Improved security for a ring-based fully homomorphic encryption scheme. In: Stam M. (ed.) Cryptography and Coding. Lecture Notes in Computer Science, vol. 8308, pp. 45–64. Springer, Berlin (2013). doi:10.1007/978-3-642-45239-0_4.
13. Brakerski Z.: Fully homomorphic encryption without modulus switching from classical gapSVP. In: Safavi-Naini R., Canetti R. (eds.) Advances in Cryptology (CRYPTO 2012). Lecture Notes in Computer Science, vol. 7417, pp. 868–886. Springer, Berlin (2012). doi:10.1007/978-3-642-32009-5_50.
14. Micciancio D., Regev O.: Worst-case to average-case reductions based on gaussian measures. SIAM J. Comput. **37**(1), 267–302 (2007). doi:10.1137/S0097539705447360.
15. Lyubashevsky V., Peikert C., Regev O.: On ideal lattices and learning with errors over rings. In: Gilbert H. (ed.) Advances in Cryptology (EUROCRYPT 2010). Lecture Notes in Computer Science, vol. 6110, pp. 1–23. Springer, Berlin (2010). doi:10.1007/978-3-642-13190-5_1.
16. Micciancio D., Regev O.: Lattice-based cryptography. In: Bernstein D., Buchmann J., Dahmen E. (eds.) Post-quantum Cryptography, pp. 147–191. Springer, Berlin (2009). doi:10.1007/978-3-540-88702-7_5.
17. Lindner R., Peikert C.: Better key sizes (and attacks) for LWE-based encryption. In: Kiayias A. (ed.) Topics in Cryptology (CT-RSA 2011). Lecture Notes in Computer Science, vol. 6558, pp. 319–339. Springer, Berlin (2011). doi:10.1007/978-3-642-19074-2_21.
18. Hoffstein J., Silverman J.H., Whyte W.: Estimated breaking times for NTRU lattices. version 2, NTRU Cryptosystems, Technical Report (2003).
19. Gama N., Nguyen P.: Predicting lattice reduction. In: Smart N. (ed.) Advances in Cryptology (EUROCRYPT 2008). Lecture Notes in Computer Science, vol. 4965, pp. 31–51. Springer, Berlin (2008). doi:10.1007/978-3-540-78967-3_3.
20. Coppersmith D., Shamir A.: Lattice attacks on NTRU. In: Fumy W. (ed.) Advances in Cryptology (EUROCRYPT 97). Lecture Notes in Computer Science, vol. 1233, pp. 52–61. Springer, Berlin (1997). doi:10.1007/3-540-69053-0_5.
21. Schnorr C., Euchner M.: Lattice basis reduction: improved practical algorithms and solving subset sum problems. Math. Program., **66**(1–3), 181–199 (1994). doi:10.1007/BF01581144.
22. Shoup V.: NTL: A Library for Doing Number Theory. http://www.shoup.net/ntl
23. van de Pol J., Smart N.: Estimating key sizes for high dimensional lattice-based systems. In: Stam M. (ed.) Cryptography and Coding. Lecture Notes in Computer Science, vol. 8308, pp. 290–303. Springer, Berlin (2013). doi:10.1007/978-3-642-45239-0_17.
24. Chen Y., Nguyen P.: BKZ 2.0: better lattice security estimates. In: Lee D., Wang X. (eds.) Advances in Cryptology (ASIACRYPT 2011). Lecture Notes in Computer Science, vol. 7073, pp. 1–20. Springer, Berlin (2011). doi:10.1007/978-3-642-25385-0_1.
25. Lepoint T., Naehrig M.: A comparison of the homomorphic encryption schemes FV and YASHE. In: Pointcheval D., Vergnaud D. (eds.) Progress in Cryptology (AFRICACRYPT 2014). Lecture Notes in Computer Science, vol. 8469, pp. 318–335. Springer, Berlin (2014). doi:10.1007/978-3-319-06734-6_20.
26. Chen Y., Nguyen P.: BKZ 2.0: Better Lattice Security Estimates. (2013). http://www.di.ens.fr/ychen/research/Full_BKZ.pdf.
27. Silverman J.H.: Invertibility in Truncated Polynomial Rings. Technical report, NTRU Cryptosystems (1998).
28. Schnhage A., Strassen V.: Schnelle multiplikation großer zahlen. Computing **7**(3–4), 281–292 (1971).
29. Canright D.: A very compact S-Box for AES. In: Rao J., Sunar B. (eds.) Cryptographic Hardware and Embedded Systems (CHES 2005). Lecture Notes in Computer Science, vol. 3659, pp. 441–455. Springer, Berlin (2005). doi:10.1007/11545262_32.
30. Gentry C., Halevi S., Smart N.: Homomorphic evaluation of the AES circuit (updated implementation). (2015). https://eprint.iacr.org/2012/099.pdf.
31. Mella S., Susella R.: On the homomorphic computation of symmetric cryptographic primitives. In: Stam M. (ed.) Cryptography and Coding. Lecture Notes in Computer Science, vol. 8308, pp. 28–44. Springer, Berlin (2013). doi:10.1007/978-3-642-45239-0_3.

32. Helib: A Software Library that Implements Homomorphic Encryption (HE). https://github.com/shaih/HElib.
33. Dai W., Doröz Y., Sunar B.: Accelerating NTRU based homomorphic encryption using GPUs. IACR Cryptology ePrint Archive, vol. 389 (2014). http://eprint.iacr.org/2014/389.
34. Öztürk E., Doröz Y., Sunar B., Savaş E.: Accelerating somewhat homomorphic evaluation using FPGAs. Cryptology ePrint Archive, Report 2015/294 (2015). http://eprint.iacr.org/.