Homomorphic AES Evaluation Using NTRU

Yarkin Doröz, Yin Hu, Berk Sunar
Worcester Polytechnic Institute

January 14, 2014

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 Alt-López, Tromer and Vaikuntanathan (ATV). 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 ATV scheme. First parameters are selected to yield an efficient and yet secure ATV 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 ATV scheme that embodies a number of optimizations. To assess the performance of the scheme we homomorphically evaluate the full 10 round AES circuit in 31 hours with 2048 message slots resulting in 55 sec per AES block evaluation time.

Keywords: Fully homomorphic encryption, NTRU, AES.

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 recrypt primitive takes nearly half a minute for the lowest security setting. In [7], 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 [11] 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. Smart and Vercauteren [9] presented a number of *batching* techniques for packing multiple data streams into a single ciphertext.

In [10] 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 [8] by the same authors. The scheme makes use of batching [9, 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 minutes per block evaluation time the byte-sliced implementation is faster, but also requires less memory. The SIMD implementation takes about 40 minutes per block.

In [12], Alt-López, Tromer and Vaikuntanathan (ATV) presented a leveled FHE scheme based on the modified NTRU [14] scheme introduced earlier by Stehle and Steinfeld [13]. A unique aspect of the ATV 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 ATV 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 is appears to be efficient, the analysis in [12] lacks concrete parameters.

Very recently Bos et al. [22] presented a leveled implementation based on ATV . The authors modify the proposed scheme in a number of aspects to build up their own fully homomorphic scheme. The semantic security of ATV is based on uniformity of the public key which relies on the DSPR assumption. In [22], the ATV scheme is modified by adopting a tensor product technique introduced by Brakerski [23] such that the security depends only on standard lattice assumptions. Furthermore, modulus–switching is no longer needed due to the 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.

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

- present a batched, bit-sliced implementation of the ATV scheme. The implementation is generic and is *not* customized to optimally evaluate any class of circuits (e.g. AES) more efficiently than other.
- 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 reduction 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 ATV scheme over the levels of computation, and develop a simple formula for estimating the number of bits one needs to cut during the modulus reduction 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 times faster than the byte sliced implementation and 43 times faster than the SIMD implementation of [10].

2 Background

In [12], Alt-López, Tromer and Vaikuntanathan proposed a multi-key homomorphic encryption scheme (LTV-FHE) based on a modified NTRU [14] scheme previously introduced by Stehlé and Steinfeld [13]. In this section we adapt their presentation to derive a single-key formulation suitable for homomorphic evaluation by a single party.

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 $\{\lfloor \frac{-q}{2} \rceil, \ldots, \lfloor \frac{q}{2} \rceil\}$. We require the ability to sample from a probability distribution χ, i.e. the truncated discrete Gaussian distribution $\mathbb{D}_{\mathbb{Z}^n,r}$ with standard deviation $r > 0$. A polynomial is B-bounded if all of its coefficients are in $[-B, B]$. A sample from this distribution is a $r\sqrt{n}$-bounded polynomial $e \in \mathrm{R}$. For a detailed treatment of the discrete Gaussian distribution see [6]. For an element $a \in \mathrm{R}$ we let $||a||_\infty = \max|a_i|$. The following rules apply on the infinity norm.

Lemma 1 ([24, 12]) *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 $\mathcal{D}_{\mathbb{Z}^n,r}$ (see [6]) with the aid of the following lemma:

Lemma 2 ([6] Lemma 4.4) *Let $n \in \mathbb{N}$. For any real number $r > \omega(\sqrt{\log(n)})$, we have*

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

Informally, the lemma tells us that by sampling from a discrete gaussian distribution $\mathcal{D}_{\mathbb{Z}^n,r}$ with deviation r we obtain a sample from a truncated gaussian distribution $\chi_{r\sqrt{n}}$ with very high probability.

The ATV-FHE Scheme. We are now ready to introduce the ATV 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 [12]. The primitives of the public key encryption scheme $\mathcal{E} = (\mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt}, \mathsf{Eval}, \mathsf{Relinearize})$ as as defined follows:

- KeyGen: We choose a decreasing sequence of primes $q_0 > q_1 > \cdots > q_d$ and a polynomial $\phi(x) = x^n + 1$. 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 key $\left\{\zeta_\tau^{(i)}(x)\right\}_{tau}^{i}$ where $\zeta_\tau^{(i)}(x) = h^{(i)}s_\tau^{(i)} + 2e_\tau^{(i)} + 2^\tau\left(f^{(i-1)}\right)^2$ in $R_{q_{i-1}}$.

- 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_B 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 ciphertexts:

  $$\mathsf{Encrypt}(b_1 + b_2) = c_1^{(0)} + c_2^{(0)} \ .$$

  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

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

  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 fresh encryption of the product $m_1 m_2$ of plaintexts. Later in Section 5 we study the growth of noise under relinearization more carefully.

3 Parameter Selection in the NTRU-FHE

A significant challenge in implementing and improving NTRU-FHE is parameter selection. In [12] and [14] the security analysis is mostly given in asymmptotics by reduction to the related learning with error (LWE) problem [13]. In this section we summarize the results of our preliminary work on parameter selection. The NTRU-FHE scheme in [12] is developed from a modified version of NTRU [14] proposed by Stehle and Steinfeld [13], which can be reduced to the Ring-LWE (RLWE) problem. Specifically, the security reduction is obtained through a hybrid argument:

1. Recall that for the NTRU-FHE scheme, the public key is of the form $h = 2gf^{-1}$ where g, f chosen from a Gaussian distribution D where f is kept secret. The DSPR problem is to distinguish polynomials of the form $h = 2gf^{-1}$ from samples h' picked uniformly at random from the ring R_q. 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.

Stehlé and Steinfeld have shown that the DSPR problem is hard even for unbounded adversaries with their parameter selection. However, the new NTRU-FHE scheme will require different parameters to support homomorphic evaluation. The impact of the new parameter settings to the security level is largely unknown and requires careful research. However, even if we assume that the DSPR problem is hard for typical NTRU-FHE parameter selection, concrete parameters are still hard to chose. 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 [10]. 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 [10] as $n \leq \log(q)(k + 110)/7.2$.

For example, given a 256-bit prime q, an 80-bit security level will require dimension $n = 6756$. However, this large estimate is actually an upper bound and assumes that the NTRU-FHE 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 NTRU-FHE scheme. For instance, the standard attack on the LWE problem requires many samples generated with the *same* secret s. However, in the NTRU-FHE 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 NTRU-FHE scheme. However, as a modified version of NTRU, the NTRU-FHE scheme suffers from the same attack as the original NTRU.

We can follow a similar approach as in the original NTRU paper [14] (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 $\mathcal{L}$ be the lattice generated by the matrix.

$$\mathcal{L} = \left(\begin{array}{c|cccc} & h_0 & h_1 & \cdots & h_{N-1} \\ \mathbf{I} & -h_{N-1} & h_0 & \cdots & h_{N-2} \\ & \vdots & \vdots & \ddots & \vdots \\ & -h_1 & -h_2 & \cdots & h_0 \\ \hline \mathbf{0} & \multicolumn{4}{c}{q\mathbf{I}} \end{array}\right)$$

Clearly, $\mathcal{L}$ 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 $\mathcal{L}$. In what follows, we present a Hermite (work) factor estimate, and experimental results in that will allow us to chose safe parameters.

Hermite Factor Estimates. Gama and Nguyen [15] proposed a useful approach to estimate the hardness of the SVP in an N-dimensional lattice $\mathcal{L}$ using the Hermite factor δ defined as

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

More practically we can compute δ_n as

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

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 [16], 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} seconds on the platform in [16].

For the NTRU-FHE 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 [15], Gama and Nguyen use q as the desired level for the original NTRU. However, for the much larger q used in the NTRU-FHE scheme, this estimate will not apply. In particular, Minkowski tells us that $\mathcal{L}$ 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 [5]: 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 into:

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

Following the recommendation of [5], we set the norm of b_1 as $q/4$. Coppersmith and Shamir observed that $q/10$ can ensure a successful attack in majority cases for NTRU with dimension $n = 167$ while $q/4$ is enough to extract some information. With a much larger dimension used then 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 $\delta_n{}^{2n} = \frac{q/4}{q^{1/2}} = \sqrt{q}/4$. In Table 1 we compiled the Hermite factor for various choices of q and n values.

Experimental Approach. As a secondary countermeasure we ran a large number of experiments to determine the time required to compromise the ATV scheme following the lattice formulation of Hoffstein, Pipher, and Silverman with the relaxation introduced by Coppersmith and Shamir [5]. We generated various ATV 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) [19] lattice reduction functions in

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**

Table 1: Hermite Factor estimates for various dimensions n and sizes of q. According to [16] for $\delta^n = 1.0066^n$, we need about 2^{80} seconds computation on a current PC. Therefore, we need $\delta < 1.0066$ and the smaller δ the higher the security margin will be.

Shoup's NTL Library 6.0 [17] linked with the GNU Multiprecision (GMP) 5.1.3 package. More specifically, we set Schnorr's pruning constant to 0 as experiments did not show that it would improve the running time with the setting of our tests. Then 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 is 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 [19]. From the collected data, we estimated that the block size of 2 can be used until about dimension $n = 26777$.

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 [19], 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 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 $n = 28940$ with $\log(q) = 1024$. The Hermite work factor estimate for the same parameters yields $\delta = 1.0061$. This is slightly more conservative than [16] whose experiments found that $\delta = 1.0066$ for the same security level.

Dimension	28340	28940	30140	31300	32768
Security	70	80	100	120	144

Table 2: Estimated security level with BKZ. Running times were collected on an Intel Xeon 2.9 GHz machine and converted to bits by taking the logarithm.

4 Optimizations

Batching. *Batching* has become an indispensable tool for boosting the efficiency of homomorphic evaluations [9]. In a nutshell, batching allows us to evaluate a circuit, e.g. AES, on multiple 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.

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

The ATV 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 [9]. 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 $a_i + b_i = a(x) + b(x) \pmod{F_i(x)}$.

Reducing the Public Key Size. To cope with the growth of noise, following Brakerski el al [11] we introduce a series of decreasing moduli $q_0 > q_1 > \ldots > q_{t-1}$; one modulus per circuit level. Modulus reduction 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 [21] 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 [21] is generic and also applies to the R_p setting.

Lemma 3 (Lemma 3.3 in [21]) *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 expansion of $f(x)^{-1}$ w.r.t. modulus $q_{t-k} = p^{k-1}$ for any $k = 1, \ldots, t$ as follows

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

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 reduction without having to pay for storage or key switching.

In the original scheme, the public key size is quadratically dependent on the 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 Section 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 [21]. 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 ([21]) *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

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

Furthermore, it is shown [21] that if n is a prime and large the following approximation shows that the probability of picking a noninvertible $f \in \mathrm{R}_q$ such that $f(1) \neq 0$ may be made negligibly small by picking appropriate p and n:

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

Here $\mathrm{R}_q(1) = \{\forall f \in \mathrm{R}_q \ \text{ s.t } \ f(1) \neq 0\}$. 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 [21] 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 $|\mathrm{R}^*_q|/|\mathrm{R}_q| \approx 1 - \ell p^{-w}$.

Optimizing Relinearization. In homomorphic circuit evaluation using ATV 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 reduction. In contrast $\tilde{c}_\tau(x) \in \mathbb{F}_2[x]/\langle\Phi(x)\rangle$. Also 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 [4]. 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 $m_1, m_2 \in \chi_1$ and parameters $g, s \in \chi_B$ encrypted using a single user (single key) with no modulus switching specialization of the LTV scheme. The secret key is $f = 2f' + 1$ where $f' \in \chi_B$. 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:

$$c_1c_2 \;=\; h^2s_1s_2 + h(s_1m_2 + s_2m_1) + 2[h(s_1e_2 + s_2e_1) + e_1m_2 + e_2m_1 + 2e_1e_2] + m_1m_2$$

To decrypt the resulting ciphertext we compute

$$\begin{aligned} f^2c_1c_2 \;=\;& 4g^2s_1s_2 + 2gf(s_1m_2 + s_2m_1) + 2[2gf(s_1e_2 + s_2e_1) + \\ &+ f^2e_1m_2 + f^2e_2m_1 + 2f^2e_1e_2] + f^2m_1m_2 \end{aligned}$$

If the following condition is satisfied no wraparound on the ciphertext coefficients will occur during decryption:

$$\begin{aligned} q/2 \;&>\; 4n^3B^4 + 4n^3B^3(2B+1) + 8n^3B^3(2B+1) + 8n^3B^2(2B+1)^2 + n^3B^2(2B+1)^2 \\ &>\; n^3(64B^4 + 48B^3 + 9B^2) \end{aligned}$$

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

Modulus Reduction. 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 reduction* as introduced in [11]. For this, we make use of a series of decreasing moduli $q_0 > q_1 > \ldots > q_t$; one modulus per level. Modulus reduction is a powerful technique that exponentially reduces the growth of noise during computations. For simplicity assume $q_{i+1}/q_i \approx \kappa$. 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 reduction, i.e. multiplication by $q_1/q_0 \approx \kappa$ and correction of parities symbolized by $p_i \in \mathcal{D}_{\mathbb{Z}^n,r}$ we obtain

$$c_1 c_2 \kappa + p_1 = \left[h^2 s_1 s_2 + h(s_1 m_2 + s_2 m_1) + 2[h(s_1 e_2 + s_2 e_1) + e_1 m_2 + e_2 m_1 + 2e_1 e_2 + m_1 m_2\right] \kappa + p_1$$

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

$$c^{2^i} = (\ldots((c^2\kappa + p_1)^2\kappa + p_2)^2 \ldots \kappa + p_{2^i})$$

We may decrypt the result as follows:

$$\begin{aligned} c^{2^i} f^{2^i} &= (\ldots((c^2\kappa + p_1)^2\kappa + p_2)^2 \ldots \kappa + p_{2^i}) f^{2^i} \\ &= (\ldots((c^2\kappa + p_1)^2\kappa + p_2)^2 \ldots \kappa + p_{2^i}) f^{2^i} \end{aligned}$$

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* [12], 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 reduction.

Relinearization and Modulus Reduction. 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 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 $s_\tau^{(i)} = s \in \chi_B$, $g_\tau^{(i)} = g \in \chi_B$ and $e_\tau^{(i)} = e \in \chi_B$ and drop unnecessary indices from here on:

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

Also factoring in the modulus reduction and parity correction steps and substituting $\tilde{c} = c^2$ we obtain

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

where $p \in \chi_1$ 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

$$Y_0 = \left(fc^2 + X_0 \right) \kappa + p_0 \ .$$

Assume we continue this process, i.e. squaring, relinearization, modulus reduction 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

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

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

$$\begin{aligned} ||fX_i||_\infty &= || \sum_\tau (2gs_\tau + 2e_\tau f)\tilde{c}_\tau ||_\infty \\ &\leq n || \sum_\tau (2gs_\tau + 2e_\tau f) ||_\infty || \sum_\tau \tilde{c}_\tau ||_\infty \\ &\leq n(2nB^2 + n2B(2B+1)) \log(q_i) \\ &\leq n^2(6B^2 + 2B) \log(q_i) \end{aligned}$$

Now let B_i denote an upper bound on the norm of a decrypted ciphertext entering level i of leveled circuit, i.e. $B_i \geq ||fY_i||_\infty$. The norm of the output grows from one level to the next including multiplication (squaring with our simplification), relinearization and modulus reduction as follows

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

Notice the level independent (fixed) noise growth term on the left summand of the recursion. In practice, the summand on the right dominates right-hand-side and therefore κ needs to be chosen so as to *stabilize* the the norm over the levels of computation, i.e. $B_1 \approx B_2 \approx \ldots \approx B_{d-1} < q_{d-1}/2$. Finally, we can make the accounting a bit more generic by defining circuit parameters a_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 a_i as follows.

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

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 $\sigma = B$. 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 $a_i b_{n-1-i} = \frac{1}{4}(a_i + b_{n-1-1})^2 - \frac{1}{4}(a_i - b_{n-1-1})^2$. As n becomes large $\chi^2(2n)$ becomes close to an ideal Gaussian distribution

with variance $4n$. Thus $\sigma((ab)_i) \approx \sqrt{n}B^2$ for large n. Therefore, a sufficiently good approximation of the expected norm may be obtained by replacing n with $\sqrt{n}$ in Equation 3 as follows.

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

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

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

Assuming nearly fixed $a_i \approx a$, if we set $1/\kappa = \epsilon\left[an(6B^2+2B)(\log(q_0)+\log(a)) + n^{3/2}(2B+1)^2B^2\right]$ 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. Values of B_i and $B_{i,avg}$ for n relevant to our implementation are tabulated in Figure 1.

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 $q_0 = p^t$. Also since with our $q = p^t$ specialization $1/\kappa \approx p$ and since $p \gg a$ neglecting $\log(a)$ we can simplify our noise estimate as follows:

$$p \approx 28at\log(p)n + 100n^{3/2} \ .$$

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 $q = p^t$ are tied through the Hermite factor $\delta = (\sqrt{q}/4)^{1/(2n)} = (\sqrt{p^t}/4)^{\frac{1}{2n}}$ and $p = (4\delta^{2n})^{2/t}$. Substituting p yields

$$\begin{aligned}(4\delta^{2n})^{2/t} &\approx 28at[4n/t\log(\delta)]n + 100n^{1.5} \ , \\ (4\delta^{2n})^{2/t} &\approx 112a\log(\delta)n^2 + 100n^{1.5} \ .\end{aligned}$$

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

		Worst Case		Average Case	
$\log(n)$	$\log(q)$	$\log(1/K)$	$\#L$	$\log(1/K)$	$\#L$
12	155	43	2	27	4
13	311	46	5	29	9
14	622	49	11	30	19
15	1244	52	22	32	37
16	2488	55	44	33	74
17	4976	58	84	35	141

Figure 1: 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 $\delta(n, q) = 1.0066$.

Failure Probability. Equation 4 tells us that we can use a much smaller q than that determined by the worst case bound in Equation 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 $q/2 > \alpha B_{avg}$ where $\alpha > 1$ captures the margin, then $\alpha B_{avg}/\sigma$ determines how much of the probability space we cover in a Gaussian distribution $N(\mu = 0, \sigma)$. 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 in 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:

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.

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.

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.

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 [20], 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.

$$
X^{-1} = \begin{pmatrix}
1 & 1 & 1 & 0 & 0 & 1 & 1 & 1 \\
0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\
0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\
1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \\
1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \\
0 & 1 & 0 & 0 & 1 & 1 & 1 & 1
\end{pmatrix}
\qquad
\mathrm{MX} = \begin{pmatrix}
0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\
1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\
1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\
0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 \\
0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\
0 & 1 & 0 & 1 & 0 & 0 & 1 & 0
\end{pmatrix}
\tag{5}
$$

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 ATV scheme with the optimizations summarized in Section 4 using Shoup's NTL library version 6.0 [17] compiled with the GMP 5.1.3 package. The implementation batches bits into ciphertexts using CRT applied using a cyclotomic modulus polynomial $\Psi_m(x)$ where $\deg(\Psi) = \text{n}$. 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.

Homomorphic AES evaluation. Using the ATV primitives we implemented the depth 40 AES circuit described in Section 6. The AES S–Box evaluation, is completed using 18 Relinearization operations and thus 2,880 Relinearizations are needed for the full AES.

We ran the AES evaluation for two choices of parameters:

- Polynomial degree of $n = 27000$ 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 $a \approx 100$ if we cut $\log(p) \approx \log(1/K) = 30$ bits at each level Equation 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 27 hours. Since we batched with 1800 message slots we obtain 54 seconds evaluation time per block.
- Polynomial degree set as $n = 32768$ 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 $a \approx 100$ if we cut $\log(1/K) = 31$ bits at each level the noise will stabilize around 12.6 bits. The total running time is 31 hours resulting in 55 seconds per block encryption with 2048 message slots.

Table 3 summarizes the parameters for the two settings and the timing results.

n	$\log(q_0)$	δ	$\log(1/K)$	Message Slots	Total Time	Time/Block
27000	1230	1.0078	30	1800	27 hours	54 sec
32768	1271	1.0067	31	2048	31 hours	55 sec

Table 3: The two settings under which we evaluated AES and timing results on Intel Xeon @ 2.9 GHz.

Memory requirements. In the implementation we are taking advantage of the reduced public key size as described in Section 4. To support a 40 level AES circuit evaluation with the original scheme in [12] for the two settings outlined above we would need to store public keys of size 67 GBytes and 87 GBytes, respectively. The optimized scheme reduces the public keys to 4.75 Gbytes 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 4 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. 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.

Brief comparison to GHS AES. When compared to the BGV style leveled AES implementation by Gentry, Smart, Halevi (GHS) [10]; our implementation runs 48 times faster than the bit-sliced and 6 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 reduction, 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 reduction, 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.

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

Table 4: Sizes of public-key in various representations with and without optimization for the two selected parameter settings.

8 Conclusion

In this work, we presented a customized leveled implementation of the NTRU based ATV 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 NTRU based FHE scheme, we homomorphically evaluated the full 10 round AES circuit in 31 hours with 2048 message slots yielding a 55 sec per AES block evaluation making it 43 times faster than the generic bit-sliced implementation, 5 times faster than the AES customized byte sliced BGV implementation by Gentry, Halevi and Smart. This suggest that the NTRU based evaluation does indeed yield more efficient homomorphic evaluation than the ones based on the BGV scheme.

9 Acknowledgements

We would like to thank Jeffrey Hoffstein for pointing us to Coppersmith and Shamir's paper [5] and to William J. Martin for helpful discussions on the ATV FHE. This work was in part supported by the NSF-CNS Awards #1117590 and #1319130.

References

[1] Craig Gentry, Fully homomorphic encryption using ideal lattices, Symposium on the Theory of Computing (STOC), 2009, pp. 169-178.

[2] R.L. Rivest, L. Adleman, and M.L. Dertouzos. On data banks and privacy homomorphisms. In Foundations of Secure Computation, 1978.

[3] Craig Gentry and Shai Halevi, Implementing Gentry's fully-homomorphic encryption scheme, *Advances in Cryptology–EUROCRYPT 2011*, pp. 129–148, 2011.

[4] A. Schönhage and V. Strassen, Schnelle Multiplikation großer Zahlen, Computing 7, pp. 281–292. 1971.

[5] Don Coppersmith, Adi Shamir, Lattice Attacks on NTRU Advances in Cryptology EUROCRYPT 97 Lecture Notes in Computer Science Volume 1233, 1997, pp 52–61.

[6] D. Micciancio and O. Regev. Worst-case to average-case reductions based on gaussian measures. SIAM J. Comput., 37(1):267–302, 2007.

[7] Wei Wang, Yin Hu, Lianmu Chen, Xinming Huang and Berk Sunar, Accelerating Fully Homomorphic Encryption on GPUs. Proceeding of 2012 IEEE HPEC, 2012.

[8] Craig Gentry, Shai Halevi, and Nigel P. Smart. Fully homomorphic encryption with polylog overhead. Advances in Cryptology EUROCRYPT 2012 Lecture Notes in Computer Science Volume 7237, 2012, pp 465–482.

[9] Nigel P. Smart and Frederik Vercauteren. Fully homomorphic SIMD operations. Designs, Codes and Cryptography, Springer, 2012, pp 1–25.

[10] Craig Gentry, Shai Halevi, and Nigel Smart. Homomorphic evaluation of the AES circuit. Advances in Cryptology – CRYPTO 2012 (2012): 850-8

[11] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. Fully homomorphic encryption without bootstrapping. Innovations in Theoretical Computer Science, ITCS (2012): 309-325.

[12] Adriana Lopez-Alt, Eran Tromer, and Vinod Vaikuntanathan. On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption. In Proceedings of the 44th symposium on Theory of Computing, pp. 1219-1234. ACM, 2012.

[13] Damien Stehlé and Ron Steinfeld. Making NTRU as secure as worst-case problems over ideal lattices. Advances in Cryptology EUROCRYPT 2011 (2011): 27-4

[14] Jeffrey Hoffstein, Jill Pipher, and Joseph Silverman. NTRU: A ring-based public key cryptosystem. Algorithmic number theory (1998): 267-288.

[15] Nicolas Gama and Phong Nguyen. Predicting lattice reduction. Advances in Cryptology-EUROCRYPT 2008 (2008): 31-5

[16] Richard Lindner and Chris Peikert. Better key sizes (and attacks) for LWE-based encryption. Topics in Cryptology CT-RSA 2011 (2011): 319-339

[17] Victor Shoup, NTL: A Library for doing Number Theory, http://www.shoup.net/ntl

[18] Jeffrey Hoffstein, Joseph H. Silverman, and William Whyte. Estimated breaking times for NTRU lattices. In version 2, NTRU Cryptosystems (2003) http://www.ntru.com/cryptolab/tech_notes.html, 1999.

[19] Claus-Peter Schnorr and Martin Euchner. Lattice basis reduction: Improved practical algorithms and solving subset sum problems. Mathematical programming 66, no. 1 (1994): 181-199.

[20] David Canright, A very compact S-box for AES. Cryptographic Hardware and Embedded Systems-CHES 2005 (2005): 441–45

[21] Joseph H. Silverman, Invertibility in Truncated Polynomial Rings URL: https://www.securityinnovation.com/uploads/Crypto/NTRUTech009.pdf October 1, 1998.

[22] Joppe W. Bos, Kristin Lauter, Jake Loftus and Michael Naehrig, Improved Security for a Ring-Based Fully Homomorphic Encryption Scheme In Lecture Notes in Computer Science PQCrypto 2013. pp. 45–64. Springer, 2013.

[23] Zvika, Brakerski. Fully Homomorphic Encryption without Modulus Switching from Classical GapSVP. In Lecture Notes in Computer Science. pp. 868–886. Springer, 2012.

[24] V. Lyubashevsky, C. Peikert, and O. Regev. On ideal lattices and learning with errors over rings. H. Gilbert, editor. Advances in Cryptology - EUROCRYPT 2010, LNCS volume 6110, Springer, 2010. pages 1-23.