Fully Homomorphic Encryption without Squashing Using Depth-3 Arithmetic Circuits

Craig Gentry and Shai Halevi
IBM T.J. Watson Research Center

September 14, 2011

Abstract

We describe a new approach for constructing fully homomorphic encryption (FHE) schemes. Previous FHE schemes all use the same blueprint from [Gentry 2009]: First construct a somewhat homomorphic encryption (SWHE) scheme, next "squash" the decryption circuit until it is simple enough to be handled within the homomorphic capacity of the SWHE scheme, and finally "bootstrap" to get a FHE scheme. In all existing schemes, the squashing technique induces an additional assumption: that the sparse subset sum problem (SSSP) is hard.

Our new approach constructs FHE as a hybrid of a SWHE and a multiplicatively homomorphic encryption (MHE) scheme, such as Elgamal. Our construction eliminates the need for the squashing step, and thereby also removes the need to assume the SSSP is hard. We describe a few concrete instantiations of the new method, including a "simple" FHE scheme where we replace SSSP with Decision Diffie-Hellman, an optimization of the simple scheme that let us "compress" the FHE ciphertext into a single Elgamal ciphertext(!), and a scheme whose security can be (quantumly) reduced to the approximate ideal-SIVP.

We stress that the new approach still relies on bootstrapping, but it shows how to bootstrap without having to "squash" the decryption circuit. The main technique is to express the decryption function of SWHE schemes as a depth-3 ($\sum\prod\sum$) arithmetic circuit of a particular form. When evaluating this circuit homomorphically (as needed for bootstrapping), we temporarily switch to a MHE scheme, such as Elgamal, to handle the $\prod$ part. Due to the special form of the circuit, the switch to the MHE scheme can be done without having to evaluate anything homomorphically. We then translate the result back to the SWHE scheme by homomorphically evaluating the decryption function of the MHE scheme. Using our method, the SWHE scheme only needs to be capable of evaluating the MHE scheme's decryption function, not its own decryption function. We thereby avoid the circularity that necessitated squashing in the original blueprint.

Key words. Arithmetic Circuits, Depth-3 Circuits, Homomorphic Encryption, Symmetric Polynomials

Contents

1 Introduction

Fully homomorphic encryption allows anyone to perform arbitrarily computations on encrypted data, despite not having the secret decryption key. Several fully homomorphic encryption (FHE) schemes appeared recently [Gen09b, vDGHV10, SV10, GH11], all following the same blueprint as Gentry's original construction [Gen09b, Gen09a]:

1. SWHE. Construct a somewhat homomorphic encryption (SWHE) scheme – roughly, a scheme that can evaluate low-degree polynomials homomorphically.

2. Squash. "Squash" the decryption function of the SWHE scheme, until decryption can be expressed as polynomial of degree low enough to be handled within the homomorphic capacity of the SWHE scheme, with enough capacity left over to evaluate a NAND gate. This is done by adding a "hint" to the public key – namely, a large set of elements that has a secret sparse subset that sums to the original secret key.

3. Bootstrap. Given a SWHE scheme that can evaluate its decryption function (plus a NAND), apply Gentry's transformation to get a "leveled"[1] FHE scheme.

In this work we construct leveled FHE by combining a SWHE scheme with a "compatible" multiplicatively homomorphic encryption (MHE) scheme (such as Elgamal) in a surprising way. Our construction still relies on bootstrapping, but it does not use squashing and does not rely on the assumed hardness of the sparse subset sum problem (SSSP). Using the new method, we construct a "simple" leveled FHE scheme where SSSP is replaced with Decision Diffie-Hellman. We also describe an optimization of this simple scheme where at one point during the bootstrapping process, the entire leveled FHE ciphertext consists of a single MHE (e.g., Elgamal) ciphertext! Finally, we show that it is possible to replace the MHE scheme by an additively homomorphic encryption (AHE) scheme that encrypts discrete logarithms. This allows us to construct a leveled FHE scheme whose security is based *entirely* on the worst-case hardness of the shortest independent vector problem over ideal lattices (ideal-SIVP) (compare [Gen10]). As in Gentry's original blueprint, we obtain a pure FHE scheme by assuming circular security. At present, our new approach does not improve efficiency, aside from the optimization that reduces the ciphertext length.

1.1 Our Main Technical Innovation

Our main technical innovation is a new way to evaluate homomorphically the decryption circuits of the underlying SWHE schemes. Decryption in these schemes involves computing a threshold function, that can be expressed as a multilinear symmetric polynomial. Previous works [Gen09b, vDGHV10, SV10, GH11] evaluated those polynomials in the "obvious way" using *boolean* circuits. Instead, here we use Ben-Or's observation (reported in [NW97]) that multilinear symmetric polynomials can be computed by depth-3 ($\sum\prod\sum$) *arithmetic* circuits over $\mathbb{Z}_p$ for large enough prime p. Let $e_k(\cdot)$ be the n-variable degree-k elementary symmetric polynomial, and consider a vector $\vec{x} = \langle x_1, \ldots, x_n \rangle \in \{0,1\}^n$. The value of $e_k(\vec{x})$ is simply the coefficient of z^{n-k} in the univariate polynomial $P(z) = \prod_{i=1}^{n}(z + x_i)$. This coefficient can be computed by fixing an arbitrary set $A = \{a_1, \ldots, a_{n+1}\} \subseteq \mathbb{Z}_p$, then evaluating the polynomial $P(z)$ at the points in A to obtain

[1]In a "leveled" FHE scheme, the size of the public key is linear in the depth of the circuits to evaluate. A "pure" FHE scheme (with a fixed-sized public key) can be obtained by assuming "circular security" – namely, that it is safe to encrypt the leveled FHE secret key under its own public key.

$t_j = P(a_j)$, and finally interpolating the coefficient of interest as a linear combination of the t_j's. The resulting circuit has the form

$$e_k(\vec{x}) \;=\; \sum_{j=1}^{n+1} \lambda_{jk} \prod_{i=1}^{n} (a_j + x_i) \pmod{p}, \tag{1}$$

where λ_{jk}'s are the interpolation coefficients, which are some known constants in $\mathbb{Z}_p$. Any multilinear symmetric polynomial over $\vec{x}$ can be computed as a linear combination of the $e_k(\vec{x})$'s, and thus has the same form (with different λ's).

By itself, Ben-Or's observation is not helpful to us, since (until now) we did not know how to bootstrap unless the polynomial *degree* of the decryption function is low. Ben-Or's observation does not help us lower the degree (it actually *increases* the degree).[2] Our insight is that we can evaluate the $\prod$ part by temporarily working with a MHE scheme, such as Elgamal [ElG85]. We first use a simple trick to get an encryption under the MHE scheme of all the $(a_j + x_i)$ terms in Ben-Or's circuit, then use the multiplicative homomorphism to multiply them, and finally convert them back to SWHE ciphertexts to do the final sum. Conversion back from MHE to SWHE is done by running the MHE scheme's decryption circuit homomorphically within the SWHE scheme, which may be expensive. However, the key point is that the degree of the translation depends only on the MHE scheme and *not on the SWHE scheme.* This breaks the self-referential requirement of being able to evaluate its *own* decryption circuit, hence obviating the need for the squashing step. Instead, we can now just increase the parameters of the SWHE scheme until it can handle the MHE scheme's decryption circuit.

1.2 An Illustration of an Elgamal-Based Instantiation

Perhaps the simplest illustration of our idea is using Elgamal encryption to do the multiplication. Let $p = 2q + 1$ be a safe prime. Elgamal messages and ciphertext components will live in $\mathrm{QR}(p)$, the group of quadratic residues modulo p. We also use a SWHE scheme with plaintext space $\mathbb{Z}_p$. (All previous SWHE schemes can be adapted to handle this large plaintext space). We also require the SWHE scheme to have a "simple" decryption function that can be expressed as a "restricted" depth-3 arithmetic circuit. These terms are defined later in Section 2, for now we just mention that all known SWHE schemes [Gen09b, vDGHV10, SV10, GH11] meet this condition

For simplicity of presentation here, imagine that the SWHE secret key is a bit vector $\vec{s} = (s_1, \ldots, s_n) \in \{0,1\}^n$, the ciphertext that we want to decrypt is also a bit vector $\vec{c} = (c_1, \ldots, c_n) \in \{0,1\}^n$, and that decryption works by first computing $x_i \leftarrow s_i \cdot c_i$ for all i, and then running the $\sum\prod\sum$ circuit, taking $\vec{x}$ as input. Imagine that decryption simply performs something like interpolation – namely, it computes $f(\vec{x}) = \sum_{j=1}^{n+1} \lambda_j \prod_{i=1}^{n} (a_j + x_i)$, where the a_j's and λ_j's are publicly known constants in $\mathbb{Z}_p$.

To enable bootstrapping, we provide (in the public key) the Elgamal secret key encrypted under the SWHE public key, namely we encrypt the bits of the secret Elgamal exponent e individually under the SWHE scheme. We also use a special form of encryption of the SWHE secret key under the Elgamal public key. Namely, for each secret-key bit s_i and each public constant a_j, we provide

[2]The degree of $P(z)$ is n, whereas in the previous blueprint Gentry's squashing technique is used to ensure that the Hamming weight of $\vec{x}$ is at most $m \ll n$, so that it suffices to compute $e_k(\vec{x})$ only for $k \leq m$.

an ElGamal encryption of the value $a_j + s_i \in \mathbb{Z}_p$. The public values a_j's are chosen so that both $a_j, a_j + 1 \in \mathrm{QR}(p)$, so that $a_j + s_i$ is always in the Elgamal plaintext space.[3]

Now let $\vec{c} \in \{0,1\}^n$ be a SWHE ciphertext that we want to decrypt *homomorphically.* First, for each (i,j), we obtain an Elgamal ciphertext that encrypts $a_j + (s_i \cdot c_i)$ as follows: if $c_i = 0$ then $a_j + (s_i \cdot c_i) = a_j$, so we simply generate a fresh encryption of the public value a_j. On the other hand, if $c_i = 1$ then $a_j + (s_i \cdot c_i) = a_j + s_i$, so we use the encryption of $a_j + s_i$ from the public key. (Note how the "restricted" form of these sums $a_j + x_i$ makes it possible to put in the public key all the Elgamal ciphertexts that are needed for these sums.)

Next we use Elgamal's multiplicative homomorphism for the $\prod$ part of the circuit, thus getting Elgamal encryptions of the values $\lambda_j \cdot P(a_j)$ (where $P(z) = \prod_i (z + x_i)$). We then convert these Elgamal encryptions into SWHE encryptions of the same values in $\mathbb{Z}_p$ by homomorphically evaluating the Elgamal decryption, using the SWHE encryption of the Elgamal secret exponent from the public key. Denote by e_i the i'th bit of the secret exponent e (so the public key includes an SWHE encryption of e_i), and let $(y,z) = (g^r, m \cdot g^{-er})$ be an Elgamal ciphertext to be converted. We compute $y^{2^i} - 1 \bmod p$ for all i, then compute SWHE ciphertexts that encrypt the powers

$$y^{e_i \cdot 2^i} = e_i y^{2^i} + (1 - e_i) y^0 = e_i (y^{2^i} - 1) + 1,$$

and then use multiplicative homomorphism of the SWHE scheme to multiply these powers and obtain an encryption of y^e. (This requires degree $\lceil \log q \rceil$). Finally, inside the SWHE scheme, we multiply the encryption of y^e by the known value z, thereby obtaining a SWHE ciphertext that encrypts m.

At this point, we have SWHE ciphertexts that encrypt the results of the $\prod$ operations – the values $\lambda_j \cdot P(a_j)$. We now use the SWHE scheme's additive homomorphism to finish off the depth-3 circuit, thus completing the homomorphic decryption. We can now compute another MULT or ADD operation, before running homomorphic decryption again to "refresh" the result, ad infinitum.

As explained above, using this approach the SWHE scheme only needs to evaluate polynomials that are slightly more complex than the MHE scheme's decryption circuit. Specifically, for Elgamal we need to evaluate polynomials of degree $2 \lceil \log q \rceil$. We can use any of the prior SWHE schemes from the literature, and set the parameters large enough to handle these polynomials. The security of the resulting leveled FHE scheme is based on the security of its component SWHE and MHE schemes.

We also show that by a careful choice of the constants a_j, we can set things up so that we always have $P(a_j) = w_j \cdot P(a_1)^{e_j}$ for some known constants $e_j, w_j \in \mathbb{Z}_p$. Hence we can compute all the Elgamal ciphertexts at the output of the Π layer given just the Elgamal ciphertext that encrypts $P(a_1)$, which yields a compact representation of the ciphertext.

1.3 Leveled FHE Based on Worst-Case Hardness

We use similar ideas to get a leveled FHE scheme whose security is based entirely on the (quantum) worst-case hardness of ideal-SIVP. At first glance this may seem surprising: how can we use a lattice-based scheme as our MHE scheme when current lattice-based schemes do not handle multiplication very well? (This was the entire reason the old blueprint required squashing!) We get around this

[3]An amusing exercise: Prove that the number of a_j's with $a_j, a_j + 1 \in \mathrm{QR}(p)$ is $(p-3)/4$ when $p = 3 \bmod 4$ and $(p-5)/4$ when $p = 1 \bmod 4$. See Lemma 5 for the answer.

apparent problem by replacing the MHE scheme with an *additively* homomorphic encryption (AHE) scheme, applied to *discrete logs*.

In more detail, as in the Elgamal-based instantiation, the SWHE scheme uses plaintext space $\mathbb{Z}_p$ for prime $p = 2q+1$. But p is chosen to be a *small prime*, polynomial in the security parameter, so it is easy to compute discrete logs modulo p. The plaintext space of the AHE scheme is $\mathbb{Z}_q$, corresponding to the space of exponents of a generator g of $\mathbb{Z}_p^*$. Rather than encrypting in the public key the values $a_j + s_i$ (as in the Elgamal instantiation), we provide AHE ciphertexts that encrypt the values $\mathrm{DL}_g(a_j + s_i) \in \mathbb{Z}_q$, and use the same trick as above to get AHE ciphertexts that encrypt the values $\mathrm{DL}_g(a_j + (s_i \cdot c_i))$. We homomorphically add these values, getting an AHE encryption of $\mathrm{DL}_g(\lambda_j \cdot P(a_j))$. Finally, we use the SWHE scheme to homomorphically compute the AHE decryption followed by exponentiation, getting SWHE encryption of the values $\lambda_j \cdot P(a_j)$, which we add within the SWHE scheme to complete the bootstrapping.

As before, the SWHE scheme only needs to support the AHE decryption (and exponentiation modulo the small prime p), thus we don't have the self-reference problem that requires squashing. We note, however, that lattice-based additively-homomorphic schemes are not completely error free, so once must set the parameters so that it supports sufficient number of summands. Since the dependence of the AHE noise on the number of summands is very weak (only logarithmic), this can be done without the need for squashing. See Section A.3 for more details on this construction.

2 Decryption as a Depth-3 Arithmetic Circuit

Recall that, in Gentry's FHE, we "refresh" a ciphertext c by expressing decryption of this ciphertext as a function $D_c(s)$ in the secret key s, and evaluating that function homomorphically. Below, we describe "restricted" depth-3 circuits, sketch a "generic" lattice based construction that encompasses known SWHE schemes (up to minor modifications), and show how to express its decryption function $D_c(s)$ as a restricted depth-3 circuit over a large enough ring $\mathbb{Z}_p$. We note that Klivans and Sherstov [KS06] have already shown that the decryption functions of Regev's cryptosystems [Reg04, Reg09] can be computed using depth-3 circuits.

2.1 Restricted Depth-3 Arithmetic Circuits

In our construction, the circuit that computes $D_c(s)$ depends on the ciphertext c only in a very restricted manner. By "restricted" we roughly mean that the bottom sums in the depth-3 circuit must come from a fixed (polynomial-size) set $\mathcal{L}$ of polynomials, where $\mathcal{L}$ itself is independent of the ciphertext. Thus, the bottom sums used in the circuit can depend on the ciphertext only to the extent that the ciphertext is used to *select* which and how many of the polynomials in $\mathcal{L}$ are used as bottom sums in the circuit.

Definition 1 (Restricted Depth-3 Circuit)**.** *Let $\mathcal{L} = \{L_j(x_1, \ldots, x_n)\}$ be a set of polynomials, all in the same n variables. An arithmetic circuit C is an $\mathcal{L}$-*restricted *depth-3 circuit over $(x_1, \ldots, x_n)$ if there exists multisets $S_1, \ldots, S_t \subseteq \mathcal{L}$ and constants $\lambda_0, \lambda_1, \ldots, \lambda_t$ such that*

$$C(\vec{x}) \;=\; \lambda_0 + \sum_{i=1}^{t} \lambda_i \cdot \prod_{L_j \in S_i} L_j(x_1, \ldots, x_n),$$

The degree of C with respect to $\mathcal{L}$ is $d = \max_i |S_i|$ (we also call it the $\mathcal{L}$-degree of C).

Remark 1. *In all our instantiations of decryption circuits for known SWHE schemes, the L_j's happen to be linear. However, our generic construction in Section 3 does not require that they be linear (or even low degree).*

To express decryption as restricted circuit as above, we use Ben-Or's observation that multilinear symmetric polynomials can be computed by restricted depth-3 arithmetic circuits that perform interpolation. Recall that a *multilinear symmetric polynomial* $M(\vec{x})$ is a symmetric polynomial where, for each i, every monomial is of degree at most 1 in x_i; there are no high powers of x_i. A simple fact is that every multilinear symmetric polynomial $M(\vec{x})$ is a linear combination of the elementary symmetric polynomials: $M(\vec{x}) = \sum_{i=0}^{n} \ell_i \cdot e_i(\vec{x})$, where $e_i(\vec{x})$ is the sum of all degree-i monomials that are the product of i distinct variables. Also, for every symmetric polynomial $S(\vec{x})$, there is a multilinear symmetric polynomial $M(\vec{x})$ that agrees with $S(\vec{x})$ on all binary vectors $\vec{x} \in \{0,1\}$. The reason is that $x_i^k = x_i$ for $x_i \in \{0,1\}$, and therefore all higher powers in $S(\vec{x})$ can be "flattened"; the end result is multilinear symmetric. The following lemma states Ben-Or's observation formally.

Lemma 1 (Ben-Or, reported in [NW97])**.** *Let $p \geq n+1$ be a prime, let $A \subseteq \mathbb{Z}_p$ have cardinality $n+1$, let $\vec{x} = (x_1, \ldots, x_n)$ be variables, and denote $\mathcal{L}_A \stackrel{\text{def}}{=} \{(a + x_i) : a \in A,\ 1 \leq i \leq n\}$. For every multilinear symmetric polynomial $M(\vec{x})$ over $\mathbb{Z}_p$, there is a circuit $C(\vec{x})$ such that:*

- *C is a $\mathcal{L}_A$-restricted depth-3 circuit over $\mathbb{Z}_p$ such that $C(\vec{x}) \equiv M(\vec{x})$ (in $\mathbb{Z}_p$).*
- *C has $n+1$ product gates of $\mathcal{L}_A$-degree n, one gate for each value $a_j \in A$, with the j'th gate computing the value $\lambda_j \cdot P(a_j) = \prod_i (a_j + x_i)$ for some scalar λ_j.*
- *A description of C can be computed efficiently given the values $M(\vec{x})$ at all $\vec{x} = 1^i 0^{n-i}$.*

The final bullet clarifies that Ben-Or's observation is constructive – we can compute the restricted depth-3 representation from any initial representation that lets us evaluate M. For completeness, we prove Lemma 1 in Appendix B.

In some cases, it is easier to work with univariate polynomials. The following fact, captured in Lemma 2, will be useful for us: Suppose $f(x)$ is an arbitrary univariate function and we want to compute $f(\sum b_i \cdot t_i)$, where the b_i's are bits and the t_i's are small (polynomial). Then, we can actually express this computation as a multilinear symmetric polynomial, and hence a restricted depth-3 circuit in the b_i's.

Lemma 2. *Let T, n be positive integers, and $f(x)$ a univariate polynomial over Z_p (for p prime, $p \geq Tn+1$). Then there is a multilinear symmetric polynomial $M_f(\cdot)$ on Tn variables such that for all $t_1, \ldots, t_n \in \{0, \ldots, T\}$,*

$$f(b_1 \cdot t_1 + \cdots + b_n \cdot t_n) = M_f(\underbrace{b_1, \ldots, b_1}_{t_1 \text{ times}}, \underbrace{0, \ldots, 0}_{T-t_1 \text{ times}}, \underbrace{b_2, \ldots, b_2}_{t_2 \text{ times}}, \underbrace{0, \ldots, 0}_{T-t_2 \text{ times}}, \ldots, \underbrace{b_n, \ldots, b_n}_{t_n \text{ times}}, \underbrace{0, \ldots, 0}_{T-t_n \text{ times}})$$

for all $\vec{b} \in \{0,1\}^n$. Moreover, a representation of M_f as a $\mathcal{L}_A$-restricted depth-3 circuit can be computed in time poly(Tn) given oracle access to f.

Proof. Define a Tn-variate polynomial $g : \mathbb{Z}_p^{Tn} \to \mathbb{Z}_p$ as $g(\vec{x}) = f(\sum x_i)$, then g is symmetric and we have

$$f(b_1 \cdot t_1 + \cdots + b_n \cdot t_n) = g(\underbrace{b_1, \ldots, b_1}_{t_1 \text{ times}}, \underbrace{0, \ldots, 0}_{T-t_1 \text{ times}}, \underbrace{b_2, \ldots, b_2}_{t_2 \text{ times}}, \underbrace{0, \ldots, 0}_{T-t_2 \text{ times}}, \ldots, \underbrace{b_n, \ldots, b_n}_{t_n \text{ times}}, \underbrace{0, \ldots, 0}_{T-t_n \text{ times}}).$$

As noted above, there is a multilinear symmetric polynomial $M_f(\vec{x})$ that agrees with $g(\vec{x})$ on all 0-1 inputs, By Lemma 1, for any $A \subseteq \mathbb{Z}_q$ of size $Tn+1$ we can compute an $\mathcal{L}_A$-restricted depth-3 circuit representation of $M_f(\vec{x})$ by evaluating $g(\vec{x})$ over the vectors $\vec{x} = 1^i 0^{Tn-i}$, which can be done using the f-oracle. □

2.2 Lattice-Based Somewhat-Homomorphic Cryptosystems

In GGH-type [GGH97] lattice-based encryption schemes, the public key describes some lattice $L \subset \mathbb{R}^n$ and the secret key is a rational matrix $S \in \mathbb{Q}^{n \times n}$ (related to the dual lattice L^*). In the schemes that we consider, the plaintext space is $\mathbb{Z}_p$ for a prime p, and an encryption of m is a vector $\vec{c} = \vec{v} + \vec{e} \in \mathbb{Z}^n$, where $\vec{v} \in L$ and $\vec{e}$ is a short noise vector satisfying $\vec{e} \equiv \vec{m} \pmod{p}$. It was shown in [Gen09a] that decryption can be implemented by computing $\vec{m} \leftarrow \vec{c} - \lceil \vec{c} \cdot S \rfloor \bmod p$, where $\lceil \cdot \rfloor$ means rounding to the nearest integer. Moreover the parameters can be set to ensure that ciphertexts are close enough to the lattice so that the vector $\vec{c} \cdot S$ is less than $1/2(N+1)$ away from $\mathbb{Z}^n$.

Somewhat similarly to [Gen09b], such schemes can be modified to make the secret key a bit vector $\vec{s} \in \{0,1\}^N$, such that $S = \sum_{i=1}^{N} s_i \cdot T_i$ with the T_i's public matrices. For example, the s_i's could be the bit description of S itself, and then each T_i's has only a single nonzero entry, of the form 2^j or 2^{-j} (for as many different values of j as needed to describe S with sufficient precision). Differently from [Gen09b], the T_i's in our setting contain no secret information – in particular we do not require a sparse subset that sums up to S. The ciphertext $\vec{c}$ from the original scheme is post-processed to yield $(\vec{c}, \{\vec{u}_i\}_{i=1}^N)$ where $\vec{u}_i = \vec{c} \cdot T_i$, and the decryption formula becomes $\vec{m} \leftarrow \vec{c} - \left\lceil \sum_{i=1}^N s_i \cdot \vec{u}_i \right\rfloor \bmod p$.

Importantly, the coefficients of the $\vec{u}$'s are output with only $\kappa = \lceil \log(N+1) \rceil$ bits of precision to the right of the binary point, just enough to ensure that the rounding remains correct in the decryption formula. For simplicity hereafter, we will assume that the plaintext vector is $\vec{m} = \langle 0, \ldots, 0, m \rangle$ – i.e., it has only one nonzero coefficient. Thus, the post-processed ciphertext becomes $(c, \{u_i\})$ (numbers rather than vectors).

2.3 Decryption Using a Restricted Depth-3 Circuit

For the rest of this section, the details of the particular encryption scheme $\mathcal{E}$ are irrelevant except insofar as it has the following decryption formula: The secret key is $\vec{s} \in \{0,1\}^N$, and the ciphertext is post-processed to the form $(c, \{u_i\})$, and each u_i is split into an integer part and a fractional part, $u_i = u'_i{}_\bullet u''_i$, such that the fractional part has only $\kappa = \lceil \log(N+1) \rceil$ bits of precision (namely, u''_i is a κ-bit integer). The plaintext is recovered as:

$$m \quad \leftarrow \quad \underbrace{c - \sum s_i \cdot u'_i}_{\text{“simple part”}} - \underbrace{\left\lceil 2^{-\kappa} \cdot \sum s_i \cdot u''_i \right\rfloor}_{\text{“complicated part”}} \bmod p. \tag{2}$$

We now show that we can compute Equation (2) using a $\mathcal{L}_A$-restricted circuit.

Lemma 3. *Let p be a prime $p > 2N^2$. Regarding the “complicated part” of Equation (2), there is a univariate polynomial $f(x)$ of degree $\leq 2N^2$ such that $f(\sum s_i \cdot u''_i) = \lceil 2^{-\kappa} \cdot \sum s_i \cdot u''_i \rfloor \bmod p$.*

Proof. Since $p > 2N^2$, there is a polynomial f of degree at most $2N^2$ such that $f(x) = \lceil 2^{-\kappa} \cdot x \rfloor \bmod p$ for all $x \in [0, 2N^2]$. The lemma follows from the fact that $\sum s_i \cdot u''_i \in [0, N \cdot (2^\kappa - 1)] \subseteq [0, 2N^2]$. □

Theorem 1. *Let p be a prime $p > 2N^2$. For any $A \subseteq \mathbb{Z}_p$ of cardinality at least $2N^2 + 1$, $\mathcal{E}$'s decryption function (Equation (2)) can be efficiently expressed as and computed using a $\mathcal{L}_A$-restricted depth-3 circuit C of $\mathcal{L}_A$-degree at most $2N^2$ having at most $2N^2 + N + 1$ product gates.*

Proof. First, consider the "complicated part". By Lemma 3, there is a univariate polynomial $f(x)$ of degree $2N^2$ such that $f(\sum s_i \cdot u_i'') = \lceil 2^{-\kappa} \cdot \sum s_i \cdot u_i'' \rfloor \bmod p$. Since all $u_i'' \in \{0, \ldots, 2N\}$, by Lemma 2, there is a multilinear symmetric polynomial $M_f(\vec{x})$ taking $2N^2$ inputs such that

$$f\big(\sum_i s_i \cdot u_i''\big) \;=\; M_f\big(s_1^{u_1''} 0^{2N-u_1''}, \ldots, s_N^{u_N''} 0^{2N-u_N''}\big)$$

for all $\vec{s} \in \{0,1\}^N$, and moreover we can efficiently compute M_f's representation as a $\mathcal{L}_A$-restricted depth-3 circuit C. By Lemma 1, C has $\mathcal{L}_A$-degree at most $2N^2$ and has at most $2N^2 + 1$ product gates. We have proved the theorem for the complicated part. To handle the "simple part" as an $\mathcal{L}_A$-restricted circuit, we can re-write it as $(c + a_1 \cdot \sum u_i') - \sum (a_1 + s_i) \cdot u_i' \bmod p$ with the constant term $\lambda_0 = (c + a_1 \cdot \sum u_i')$. The circuit for the simple part has $\mathcal{L}_A$-degree 1 and N "product" gates. □

In Section 4.2, we show how to tweak the "generic" lattice-based decryption further to allow a *purely* multilinear symmetric decryption formula. (Above, only the complicated part is multilinear symmetric.) While not essential to construct leveled FHE schemes, this tweak enables interesting optimizations. For example, in 4.1 we show that we can get a very compact leveled FHE ciphertext – specifically, at one point, it consists of a *single MHE ciphertext* – e.g., a single Elgamal ciphertext! This single MHE ciphertext encrypts the value $P(a_1)$, and we show how (through a clever choice of a_i's) to derive MHE ciphertexts that encrypt $P(a_i)$ for the other i's.

3 Leveled FHE from SWHE and MHE

Here, we show how to take a SWHE scheme that has restricted depth-3 decryption and a MHE scheme, and combine them together into a "monstrous chimera" [Wik11] to obtain leveled FHE. The construction works much like the Elgamal-based example given in the Introduction. That is, given a SWHE ciphertext, we "recrypt" it by homomorphically evaluating its depth-3 decryption circuit, pre-processing the first level of linear polynomials $L_j(\vec{s})$ (where $\vec{s}$ is the secret key) by encrypting them under the MHE scheme, evaluating the products under the MHE scheme, converting MHE ciphertexts into SWHE ciphertexts of the same values by evaluating the MHE's scheme's decryption function under the SWHE scheme using the encrypted MHE secret key, and finally performing the final sum (an interpolation) under the SWHE scheme. The SWHE scheme only needs to be capable of evaluating the MHE scheme's decryption circuit, followed by a quadratic polynomial. Contrary to the old blueprint, the required "homomorphic capacity" of the SWHE scheme is largely independent of the SWHE scheme's decryption function.

3.1 Notations

Recall that an encryption scheme $\mathcal{E} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$ with plaintext space $\mathcal{P}$ is *somewhat-homomorphic* (SWHE) with respect to a class $\mathcal{F}$ of multivariate functions[4] over $\mathcal{P}$, if for every

[4]The class $\mathcal{F}$ may depend on the security parameter λ.

$f(x_1, \ldots, x_n) \in \mathcal{F}$ and every $m_1, \ldots, m_n \in \mathcal{P}$, it holds (with probability one) that

$$\mathsf{Dec}(sk, \mathsf{Eval}(pk, f,\ c_1, \ldots, c_n)) = f(m_1, \ldots, m_n),$$

where (sk, pk) are generated by $\mathsf{KeyGen}(1^\lambda)$ and the c_i's are generated as $c_i \leftarrow \mathsf{Enc}(pk, m_i)$. We refer to $\mathcal{F}$ as the "homomorphic capacity" of $\mathcal{E}$. We say that $\mathcal{E}$ is *multiplicatively* (resp. *additively*) homomorphic if all the functions in $\mathcal{F}$ are naturally described as multiplication (resp. addition).

Given the encryption scheme $\mathcal{E}$, we denote by $\mathcal{C}_\mathcal{E}(pk)$ the space of "freshly-encrypted ciphertexts" for the public key pk, namely the range of the encryption function for this public key. We also denote by $\mathcal{C}_\mathcal{E}$ the set of freshly-encrypted ciphertexts with respect to all valid public keys, and by $\mathcal{C}_{\mathcal{E},\mathcal{F}}$ the set of "evaluated ciphertexts" for a class of functions $\mathcal{F}$ (i.e. those that are obtained by evaluating homomorphically a function from $\mathcal{F}$ on ciphertexts from $\mathcal{C}_\mathcal{E}$). That is (for implicit security parameter λ),

$$\mathcal{C}_\mathcal{E} \overset{\text{def}}{=} \bigcup_{pk \in \mathsf{KeyGen}} \mathcal{C}_\mathcal{E}(pk), \quad \mathcal{C}_{\mathcal{E},\mathcal{F}} \overset{\text{def}}{=} \left\{\mathsf{Eval}(pk, f, \vec{c}) \ : \ pk \in \mathsf{KeyGen},\ f \in \mathcal{F},\ \vec{c} \in \mathcal{C}_\mathcal{E}(pk)\right\}$$

3.2 Compatible SWHE and MHE Schemes

To construct "chimeric" leveled FHE, the component SWHE and MHE schemes must be *compatible*:

Definition 2 (Chimerically Compatible SWHE and MHE)**.** *Let* SWHE *be an encryption scheme with plaintext space* $\mathbb{Z}_p$*, which is somewhat homomorphic with respect to some class* $\mathcal{F}$*. Let* MHE *be a scheme with plaintext space* $\mathcal{P} \subseteq \mathbb{Z}_p$*, which is multiplicatively homomorphic with respect to another* $\mathcal{F}'$*.*

We say that SWHE *and* MHE *are chimerically compatible if there exists a polynomial-size set* $\mathcal{L} = \{L_j\}$ *of polynomials and polynomial bounds* D *and* B *such that the following hold:*

- *For every ciphertext* $c \in \mathcal{C}_{\mathsf{SWHE},\mathcal{F}}$*, the function* $\mathcal{D}_c(sk) = \mathsf{SWHE.Dec}(sk, c)$ *can be evaluated by an* $\mathcal{L}$*-restricted circuit over* $\mathbb{Z}_p$ *with* $\mathcal{L}$*-degree* D*. Moreover, an explicit description of this circuit can be computed efficiently given* c*.*
- *For any secret key* $sk \in \mathsf{SWHE.KeyGen}$ *and any polynomial* $L_j \in \mathcal{L}$ *we have* $L_j(sk) \in \mathcal{P}$*. I.e., evaluating* L_j *on the secret key* sk *lands us in the plaintext space of* MHE*.*
- *The homomorphic capacity* $\mathcal{F}'$ *of* MHE *includes all products of* D *or less variables.*
- *The homomorphic capacity of* SWHE *is sufficient to evaluate the decryption of* MHE *followed by a quadratic polynomial (with polynomially many terms) over* $\mathbb{Z}_p$*. Formally, the number of product gates in all the* $\mathcal{L}$*-restricted circuits from the first bullet above is at most the bound* B*, and for any two vectors of MHE ciphertexts* $\vec{c} = \langle c_1, \ldots c_b \rangle$ *and* $\vec{c}' = \langle c'_1, \ldots c'_{b'} \rangle \in \mathcal{C}^{\le B}_{\mathsf{MHE},\mathcal{F}'}$*, the two functions*

$$\mathsf{DAdd}_{\vec{c},\vec{c}'}(sk) \overset{\text{def}}{=} \sum_{i=1}^{b} \mathsf{MHE.Dec}(sk, c_i) + \sum_{i=1}^{b'} \mathsf{MHE.Dec}(sk, c'_i) \bmod p$$

$$\mathsf{DMul}_{\vec{c},\vec{c}'}(sk) \overset{\text{def}}{=} \Big(\sum_{i=1}^{b} \mathsf{MHE.Dec}(sk, c_i)\Big) \cdot \Big(\sum_{i=1}^{b'} \mathsf{MHE.Dec}(sk, c'_i)\Big) \bmod p$$

are within the homomorphic capacity of SWHE *– i.e.,* $\mathsf{DAdd}_{\vec{c},\vec{c}'}(sk), \mathsf{DMul}_{\vec{c},\vec{c}'}(sk) \in \mathcal{F}$*.*

We note that the question of whether two schemes are compatible may depend crucially on the exact representation of the secret keys and ciphertexts in both. Consider for example our Elgamal instantiation from the introduction. While a naive implementation of exponentiation would have exponential degree, certainly too high to be evaluated by any known SWHE scheme, we were able to post-process the Elgamal ciphertext so as to make the degree of decryption more manageable.

We also note that we can view "additively-homomorphic encryption of discrete logarithms" as a particular type of multiplicative-homomorphic scheme, where encryption include taking discrete-logarithm (assuming that it can be done efficiently) and decryption includes exponentiation.

3.3 Chimeric Leveled FHE: The Construction

Let SWHE and MHE be chimerically compatible schemes. We construct a leveled FHE scheme as follows:

$\underline{\mathsf{FHE.KeyGen}(\lambda, \ell)}$: Takes as input the security parameter λ and the number of circuit levels ℓ that the composed scheme should be capable of evaluating. For $i \in [1, \ell]$, run

$$\left(pk_{SW}^{(i)},\ sk_{SW}^{(i)}\right) \xleftarrow{\mathrm{R}} \mathsf{SWHE.KeyGen}\ , \qquad \left(pk_{MH}^{(i)},\ sk_{MH}^{(i)}\right) \xleftarrow{\mathrm{R}} \mathsf{MHE.KeyGen}\ .$$

Encrypt the i'th MHE secret key under the $(i+1)$'st SWHE public key, $\overline{sk}_{MH}^{(i)} \leftarrow \mathsf{SWHE.Enc}(pk_{SW}^{(i+1)}, sk_{MH}^{(i)})$. Also encrypt the i'th SWHE secret key under the i'th MHE public key, but in a particular format as follows: Recall that there is a polynomial-size set of polynomials $\mathcal{L}$ such that SWHE decryption can be computed by $\mathcal{L}$-restricted circuits. To encrypt $sk_{SW}^{(i)}$ under $pk_{MH}^{(i)}$, compute $m_{ij} \leftarrow L_j(sk_{SW}^{(i)})$ for all $L_j \in \mathcal{L}$, and then encrypt it $c_{ij} \leftarrow \mathsf{MHE.Enc}(pk_{MH}^{(i)}, m_{ij})$. Let $\overline{sk}_{SW}^{(i)}$ denote the collection of all the c_{ij}'s. The public key pk_{FH} consists of $(pk_{SW}^{(i)}, pk_{MH}^{(i)})$ and the encrypted secret keys $(\overline{sk}_{SW}^{(i)}, \overline{sk}_{MH}^{(i)})$ for all i. The secret key sk_{FH} consists of $sk_{SW}^{(i)}$ for all i.

$\underline{\mathsf{FHE.Enc}(pk_{FH}, m)}$: Takes as input the public key pk_{FH} and a message in the plaintext space of the SWHE scheme. It outputs $\mathsf{SWHE.Enc}(pk_{SW}^{(1)}, m)$.

$\underline{\mathsf{FHE.Dec}(sk_{FH}, c)}$: Takes as input the secret key sk_{FH} and a SWHE ciphertext. Suppose the ciphertext is encrypted under $pk_{SW}^{(i)}$. It is decrypted directly using $\mathsf{SWHE.Dec}(sk_{SW}^{(i)}, c)$.

$\underline{\mathsf{FHE.Recrypt}(pk_{FH}, c)}$: Takes as input the public key and a ciphertext c that is a valid "evaluated SWHE ciphertext" under $pk_{SW}^{(i)}$, and outputs a "refreshed" SWHE ciphertext c', encrypting the same plaintext but under $pk_{SW}^{(i+1)}$. It works as follows:

Circuit-generation. For a SWHE ciphertext c, generate a description of the $\mathcal{L}$-restricted circuit C over $\mathbb{Z}_p$ that computes the decryption of c. Denote it by

$$C_c(sk) = \lambda_0 + \sum_{k=1}^{t} \lambda_k \prod_{L_j \in S_k} L_j(sk) \mod p \quad (= \mathsf{SWHE.Dec}(sk, c))$$

Products. Pick up from the public key the encryptions under the MHE public key $pk_{MH}^{(i)}$ of the values $L_j(sk_{SW}^{(i)})$. Use the homomorphism of MHE to compute MHE encryptions of all the terms $\lambda_k \cdot \prod_{L_j \in S_k} L_j(sk_{SW}^{(i)})$. Denote the set of resulting MHE ciphertexts by $c_1, \ldots, c_t$.

Translation. Pick up from the public key the encryption under the SWHE public key $pk_{SW}^{(i+1)}$ of the MHE secret key $sk_{MH}^{(i)}$. For each MHE ciphertext c_i from the Products step, use the homomorphism of SWHE to evaluate the function $D_{c_i}(sk) = \mathsf{MHE.Dec}(sk, c_i)$ on the encrypted secret key. The results are SWHE ciphertexts $c'_1, \ldots c'_t$, where c'_j encrypts the value $\lambda_k \cdot \prod_{L_j \in S_k} L_j(sk_{SW}^{(i)})$ under $pk_{SW}^{(i+1)}$.

Summation. Use the homomorphism of SWHE to sum up all the c'_j's and add λ_0 to get a ciphertext c^* that encrypts under $pk_{SW}^{(i+1)}$ the value

$$\lambda_0 + \sum_{k=1}^{t} \lambda_k \prod_{L_j \in S_k} L_j(sk_{SW}^{(i)}) \mod p \ = \mathsf{SWHE.Dec}(sk_{SW}^{(i)}, c)$$

Namely, c^* encrypts under $pk_{SW}^{(i+1)}$ the same value that was encrypted in c under $pk_{SW}^{(i)}$.

$\underline{\mathsf{FHE.Add}(pk_{FH}, c_1, c_2)}$ and $\underline{\mathsf{FHE.Mult}(pk_{FH}, c_1, c_2)}$: Take as input the public key and two ciphertexts that are valid evaluated SWHE ciphertexts under $pk_{SW}^{(i)}$. Ciphertexts within the SWHE scheme (at any level) may be added and multiplied within the homomorphic capacity of the SWHE scheme. Once the capacity is reached, they can be recrypted and then at least one more operation can be applied.

Theorem 2. *If* SWHE *and* MHE *are chimerically compatible schemes, then the above scheme* FHE *is a leveled FHE scheme. Also, if both* SWHE *and* MHE *are semantically secure, then so is* FHE*.*

Correctness follows in a straightforward manner from the definition of chimerically compatible schemes. Security follows by a standard hybrid argument similar to Theorem 4.2.3 in [Gen09a]. We omit the details.

4 Optimizations

In the $\mathsf{Products}$ step of the $\mathsf{Recrypt}$ process (see Section 3), we compute multiple products homomorphically within the MHE scheme. In Section 4.1, we provide an optimization that allows us to compute only *a single product* in the $\mathsf{Products}$ step. In Section 4.2, we extend this optimization so that the entire leveled FHE ciphertext after the $\mathsf{Products}$ step can consist of a *single MHE ciphertext.*

4.1 Computing Only One Product

For now, let us ignore the "simple part" of our decryption function (Equation 2), which is linear and therefore does not involve any "real products".

The products in the "complicated part" all have a special form. Specifically, by Theorem 1 and the preceding lemmas, for secret key $\vec{s} \in \{0,1\}^N$, ciphertext $(c, \{u_i\})$, set $A \subset \mathbb{Z}_p$ with $|A| > 2N^2$, and fixed scalars $\{\lambda_j\}$ associated to a multilinear symmetric polynomial M_f, the products are all of the form $\lambda_j \cdot P(a_j)$ for all $a \in A$, where

$$P(z) = \prod_i (z + s_i)^{u''_i} \cdot (z + 0)^{2N - u''_i} \ .$$

We will show how to choose the a_j's so that we can compute $P(a_j)$ for all j given only $P(a_1)$. This may seem surprising, but observe that the $P(a_j)$'s are highly redundant. Namely, if we consider the integer $v = \sum_{s_i=1} u''_i$ (which is at most $2N^2$), then we have

$$P(a_j) = (a_j + 1)^v \cdot (a_j + 0)^{2N^2 - v}.$$

Knowing a_1, the value of $P(a_1)$ contains enough information to deduce v, and then knowing a_j we can get $P(a_j)$ for all j. To be able to compute the $P(a_j)$'s efficiently from $P(a_1)$, we choose the a_j's so that for all $j > 1$ we know integers (w_j, e_j) such that:

$$a_j = w_j \cdot a_1^{e_j} \quad \text{and} \quad a_j + 1 = w_j \cdot (a_1 + 1)^{e_j}.$$

We store (w_j, e_j) in the public key, and then compute $P(a_j) = w_j^{2N^2} \cdot P(a_1)^{e_j}$.

Importantly for our application to chimeric FHE, we can compute an encryption of $P(a_j)$ from an encryption of $P(a_1)$ within the MHE scheme – simply use the multiplicative homomorphism to exponentiate by e_j (using repeated squaring as necessary) and then multiply the result by $w_j^{2N^2}$.

Generating suitable tuples (a_j, w_j, e_j) for $j > 1$ from an initial value a_1 is straightforward: We choose the e_j's arbitrarily and then solve for the rest. Namely, we generate distinct e_j's, different from 0,1, then set $a_j \leftarrow a_1^{e_j}/((a_1 + 1)^{e_j} - a_1^{e_j})$ and $w_j = a_j/a_1^{e_j}$. Observe that $a_j + 1 = (a_1 + 1)^{e_j}/((a_1 + 1)^{e_j} - a_1^{e_j})$ – i.e., the ratio $(a_j + 1)/a_j = ((a_1 + 1)/a_1)^{e_j}$, as required.

Some care must be taken to ensure that the values $a_j, a_j + 1$ are in plaintext space of the MHE scheme – e.g., for Elgamal they need to be quadratic residues. Recall the basic fact that for a safe prime p there are $(p - 3)/4$ values a for which $a, a + 1 \in \mathrm{QR}(p)$ (see Lemma 5). Therefore, finding suitable $a_1, a_1 + 1 \in \mathrm{QR}(p)$ is straightforward. Since $a_1^{e_j}, (a_1 + 1)^{e_j} \in \mathrm{QR}(p)$, we have

$$a_j, a_j + 1 \in \mathrm{QR}(p) \;\Leftrightarrow\; (a_1 + 1)^{e_j} - a_1^{e_j} \in \mathrm{QR}(p) \;\Leftrightarrow\; ((a_1 + 1)/a_1)^{e_j} - 1 \in \mathrm{QR}(p).$$

If $(a_1 + 1)/a_1$ generates $\mathrm{QR}(p)$ (which is certainly true if p is a safe prime), then (re-using the basic fact above) we conclude that $a_j, a_j + 1 \in \mathrm{QR}(p)$ with probability approximately 1/2 over the choices of e_j.

Observe that the amount of extra information needed in the public key is small. The e_j's need not be truly random – indeed, by an averaging argument over the choice of a_1, one will quickly find an a_1 for which suitable e_j's are $O(1)$-dense among very small integers. Hence it is sufficient to add to the public key only $O(\log p)$ bits to specify a_1.

4.2 Short FHE Ciphertexts: Decryption as a Pure Symmetric Polynomial

Here we provide an optimization that allows us to compress the *entire* leveled FHE ciphertext down to a *single MHE ciphertext* – e.g., a single Elgamal ciphertext! (The optimization above only compresses only representation of the "complicated part" of Equation 2, not the "simple part".) Typically, a MHE ciphertext will be much much shorter than a SWHE ciphertext: a few thousand bits vs. millions of bits.

The main idea is that we do not need the full ciphertext $(c, \{u'_i\}, \{u''_i\})$ to recover m if we know *a priori* that m is in a small interval – e.g., $m \in \{0, 1\}$. Rather, we can choose a "large-enough" polynomial-size prime r, so that we can recover m just from $([c]_r, \{[u'_i]_r\}, \{[u''_i]_r\})$, where $[x]_r$ denotes $x \bmod r \in \{0, \ldots, r - 1\}$. Moreover, after reducing the ciphertext components modulo r, we can invoke Lemma 2 to represent decryption as a *purely* multilinear symmetric polynomial, whose output after the product step can be represented by a *single* product $P(a_1)$ (like the complicated part in the optimization of Section 4.1).

Lemma 4. *Let prime $p = \omega(N^2)$. There is a prime $r = O(N)$ and a univariate polynomial $f(x)$ of degree $O(N^2)$ such that, for all ciphertexts $(c, \{u'_i\}, \{u''_i\})$ that encrypt $m \in \{0,1\}$, we have $m = f(t_r) \bmod p$ where*

$$t_r \overset{\text{def}}{=} [2^\kappa \cdot c]_r + \textstyle\sum_i s_i \cdot [-2^\kappa \cdot u'_i - u''_i]_r. \tag{3}$$

Proof. Let $t = 2^\kappa\big(c - \sum s_i \cdot u'_i\big) - \sum s_i \cdot u''_i$. The original decryption formula (Equation 2) is

$$m = c - \sum s_i \cdot u'_i - \lfloor 2^{-\kappa} \cdot \sum s_i \cdot u''_i \rceil = \lfloor 2^{-\kappa} \cdot t \rceil \bmod p$$

Thus, m can be recovered from t. Since there are only 2 possibilities for m, the (consecutive) support of t has size $2^{\kappa+1} = O(N)$. Set r to be a prime $\geq 2^{\kappa+1}$. Since the mapping $x \mapsto [x]_r$ has no collisions over the support of t, t can be recovered from $[t]_r$. Note that $[t]_r = [t_r]_r$. Thus m can be recovered from t_r (via $[t_r]_r = [t]_r$, then t). Since there are $O(N \cdot r) = O(N^2)$ possibilities for t_r, the lemma follows. □

Theorem 3. *Let prime $p = \omega(N^2)$. There is a prime $r = O(N)$ and a multilinear symmetric polynomial M such that, for all "hashed" ciphertexts $([2^\kappa \cdot c]_r, \{[-2^\kappa \cdot u'_i - u''_i]_r\})$ that encrypt $m \in \{0,1\}$, we have*

$$m = M(\underbrace{1,\ldots,1}_{[2^\kappa \cdot c]_r}, \underbrace{0,\ldots,0}_{r-[2^\kappa \cdot c]_r}, \ldots \underbrace{s_1,\ldots,s_1}_{[-2^\kappa \cdot u'_1 - u''_1]_r}, \underbrace{0,\ldots,0}_{r-[-2^\kappa \cdot u'_1 - u''_1]_r}, \ldots \underbrace{s_N,\ldots,s_N}_{[-2^\kappa \cdot u'_N - u''_N]_r}, \underbrace{0,\ldots,0}_{r-[-2^\kappa \cdot u'_N - u''_N]_r}) \bmod p$$

Proof. This follows easily from Lemmas 4 and 2. □

Thus, decryption can be turned into a purely multilinear symmetric polynomial M whose product gates output $\lambda_j \cdot P(a_j)$ (for known ciphertext-independent λ_j's), where $P(z)$ is similar to the polynomial described in Section 4.1. Using the optimization of Section 4.1, we can compress the entire leveled FHE ciphertext down to a single MHE ciphertext that encrypts $P(a_1)$.

Acknowledgments This material is based on research sponsored by DARPA under agreement number FA8750-11-C-0096. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of DARPA or the U.S. Government. Distribution Statement "A" (Approved for Public Release, Distribution Unlimited)

References

[BV11] Zvika Brakerski and Vinod Vaikuntanathan. Fully homomorphic encryption for ring-lwe and security for key dependent messages. In *Advances in Cryptology - CRYPTO 2011*, Lecture Notes in Computer Science. Springer, 2011.

[ElG85] T. ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *Advances in Cryptology – CRYPTO '84*, volume 196 of *Lecture Notes in Computer Science*, pages 10–18. Springer-Verlag, 1985.

[Gen09a] Craig Gentry. *A fully homomorphic encryption scheme*. PhD thesis, Stanford University, 2009. `http://crypto.stanford.edu/craig`.

[Gen09b] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of the 41st ACM Symposium on Theory of Computing – STOC 2009*, pages 169–178. ACM, 2009.

[Gen10] Craig Gentry. Toward basing fully homomorphic encryption on worst-case hardness. In Tal Rabin, editor, *Advances in Cryptology - CRYPTO 2010*, volume 6223 of *Lecture Notes in Computer Science*, pages 116–137. Springer, 2010.

[GGH97] Oded Goldreich, Shafi Goldwasser, and Shai Halevi. Public-key cryptosystems from lattice reduction problems. In Burton S. Kaliski Jr., editor, *Advances in Cryptology - CRYPTO 1997*, volume 1294 of *Lecture Notes in Computer Science*, pages 112–131. Springer, 1997.

[GH11] Craig Gentry and Shai Halevi. Implementing gentry's fully-homomorphic encryption scheme. In *Advances in Cryptology - EUROCRYPT'11*, volume 6632 of *Lecture Notes in Computer Science*, pages 129–148. Springer, 2011. Full version available on-line from `http://eprint.iacr.org/2010/520`.

[GPV08] Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In *STOC'08*, pages 197–206. ACM, 2008.

[KR] Richard M. Karp and Vijaya Ramachandran. A survey of parallel algorithms for shared-memory machines. Chapter 17 of *Handbook of Theoretical Computer Science, Volume A: Algorithms and Complexity*, MIT Press, 1990, pages 869-941.

[KS06] Adam R. Klivans and Alexander A. Sherstov. Cryptographic hardness for learning intersections of halfspaces. In *FOCS*, pages 553–562. IEEE Computer Society, 2006.

[NW97] Noam Nisan and Avi Wigderson. Lower bounds on arithmetic circuits via partial derivatives. *Computational Complexity*, 6(3):217–234, 1997. Cites "M. Ben-Or, Private communication".

[Pei11] Chris Peikert, 2011. Private communication.

[Reg04] Oded Regev. New lattice-based cryptographic constructions. *JACM*, 51(6):899–942, 2004.

[Reg09] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. *JACM*, 56(6), 2009.

[SS10] Damien Stehlé and Ron Steinfeld. Faster fully homomorphic encryption. In Masayuki Abe, editor, *Advances in Cryptology - ASIACRYPT 2010*, volume 6477 of *Lecture Notes in Computer Science*, pages 377–394. Springer, 2010.

[SV10] Nigel P. Smart and Frederik Vercauteren. Fully homomorphic encryption with relatively small key and ciphertext sizes. In Phong Q. Nguyen and David Pointcheval, editors, *Public Key Cryptography - PKC 2010*, volume 6056 of *Lecture Notes in Computer Science*, pages 420–443. Springer, 2010.

[vDGHV10] Marten van Dijk, Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. Fully homomorphic encryption over the integers. In *Advances in Cryptology - EUROCRYPT'10*, volume 6110 of *Lecture Notes in Computer Science*, pages 24–43. Springer, 2010. Full version available on-line from `http://eprint.iacr.org/2009/616`.

[Wik11] Wikipedia. Chimera. `http://en.wikipedia.org/wiki/Chimera`, Accessed on August 2011.

A Instantiations of Chimeric FHE

A.1 The Homomorphic Capacity of SWHE Schemes

Our instantiations are mildly sensitive to the tradeoff between the parameters of the SWHE scheme and its homomorphic capacity. Recall that when used with plaintext space $\mathbb{Z}_p$, the SWHE schemes that we consider have secret key $\vec{s} \in \{0,1\}^N$ and decryption formula[5] for post-processed ciphertexts $(c, \{u'_i\}, \{u''_i\})$:

$$m = c + \sum_{i=1}^{N} s_i \cdot u'_i + \left\lceil 2^{-\kappa} \cdot \sum_{i=1}^{N} s_i \cdot u''_i \right\rfloor \bmod p, \tag{4}$$

with $\kappa = \lceil \log(N+1) \rceil$, $u'_i \in \mathbb{Z}_p$ and $u''_i \in \{0, 1, \ldots, 2^\kappa\}$. Below we say that a scheme has *threshold-type decryption* if it has this decryption formula.

We are interested in the tradeoff between the number N of secret-key bits and the degree of the polynomials that the scheme can evaluate homomorphically. For our instantiations, we only need the number of key-bits to depend polynomially on the degree. Specifically, we need a polynomial bound K, such that the scheme with plaintext space $\mathbb{Z}_p$ with security parameter λ can be made to support α-degree polynomials with up to 2^β terms using secret keys of no more than $N = K(\lambda, \log p, \alpha, \beta)$ bits.

Below we say that a SWHE scheme is *"homomorphic for low-degree polynomials"* if it has a polynomial bound on the key-size as above. It can be verified that all the known lattice-based SWHE schemes meet this condition.

A.2 Elgamal-based Instantiation

In the Introduction, we specified (in a fair amount of detail) an instantiation of chimeric leveled FHE that uses Elgamal as the MHE scheme. Here, we provide a supporting lemmas and theorems to show that Elgamal is chimerically compatible with known SWHE schemes, as needed for the chimeric combination to actually work.

Theorem 4. *Let $p = 2q + 1$ be a safe prime such that DDH holds in* $\mathrm{QR}(p)$*, and let* SWHE *be an encryption scheme with message space $\mathbb{Z}_p$, which is homomorphic for low-degree polynomials and has threshold-type decryption. Then* SWHE *is chimerically compatible with Elgamal encryption modulo p over plaintext space* $\mathrm{QR}(p)$*.*

Proof. Denote the security parameter by λ and let $\alpha = \mathrm{poly}(\lambda)$ be another parameter (to be set later) governing the degree of polynomials that can be homomorphically evaluated by the scheme.

[5]This formula differs from Equation (2) in that we add rather than subtract the sums. This change was done to simplify notations in some of the arguments below, and it entails only a slight modification of the scheme.

The scheme SWHE can then be set to support polynomials of degree up to α having at most 2^α terms, using a secret key $\vec{s} \in \{0,1\}^N$ of size $N = K(\lambda, \log p, \alpha)$ for a polynomial K, with decryption formula Equation (4). Since p must be super polynomial in λ (for DDH to hold), then in particular $2N^2 < p$ and we can use Theorem 1.

We thus conclude that the for any $A \subseteq \mathbb{Z}_p$ of cardinality $2N^2 + 1$, given a SWHE ciphertext $(c, \{u'_i\}, \{u''_i\})$ we can compute efficiently a $\mathcal{L}_A$-restricted depth-3 circuit C of $\mathcal{L}_A$-degree at most $2N^2$ and at most $2N^2 + N + 1$ product gates, such that $C(\vec{s}) = \mathsf{SWHE.Dec}_{\vec{s}}(c, \{u'_i\}, \{u''_i\})$. We will thus use $D = 2N^2$ and $B = 2N^2 + N + 1$ as the bounds that are needed for Definition 2.

Next we need to establish that one can choose A so that, for any $sk \in \mathsf{SWHE.KeyGen}$ and any polynomial $L_j \in \mathcal{L}_A$, $L_j(sk)$ is in the plaintext space of our multiplicatively homomorphic scheme. In Lemma 5 below, we show that there are $q - 1 = (p-3)/4$ values a such that $a, a+1 \in \mathrm{QR}(p)$. Since N is polynomial and $2N^2 + 1 \ll q$, we can populate A with $N^2 + 1$ such values efficiently. The value $a_j + x_i$ for $a_j \in A$ and secret key bit x_i is always in $\mathrm{QR}(p)$, which is the Elgamal plaintext space.

In this construction we trivially get the property that the MHE scheme (i.e., Elgamal) can evaluate the D multiplications needed by the circuits C, since the multiplicative homomorphic capacity of Elgamal is infinite.

It remains to show that the homomorphic capacity of the SWHE scheme is sufficient to evaluate Elgamal decryption followed by one operation (i.e., the last bullet in Definition 1). It suffices to show that Elgamal decryption can be computed using a polynomial of degree α with at most 2^α monomials, so our degree parameter α. To prepare for decryption, we post-process each Elgamal ciphertext as follows: Given a ciphertext $(y = g^r, z = m \cdot g^{-er}) \in \mathbb{Z}_p^2$, we compute $y_i = y^{2^i} - 1 \bmod p$ for $i = 0, 1, \ldots, \lceil \log q \rceil - 1$, and the post-processed ciphertext is $\langle z, y_0, \ldots, y_{\tau-1} \rangle$ with $\tau = \lceil \log q \rceil$. Given an Elgamal secret key $e \in \mathbb{Z}_q$ with binary representation $e_{\tau-1} \ldots e_1 e_0$ (where $\tau = \lceil \log q \rceil$, decryption of the post-processed ciphertext becomes

$$\mathsf{MHE.Dec}(e; z, y_0, \ldots, y_{\tau-1}) = z \cdot \prod_{i=0}^{\tau-1} (y^{e \cdot 2^i}) = z \cdot \prod_{i=0}^{\tau-1} (e_i \cdot y_i + 1) \tag{5}$$

Being overly conservative and treating $z, y_0, \ldots, y_{\tau-1}$ as variables; then the degree of the polynomial above is $2\tau + 1$, and it has 2^τ monomials. Hence the degree parameter α as $\alpha = 4 \lceil \log q \rceil + 2$, we get a scheme whose homomorphic capacity is sufficient for Elgamal decryption followed by one operation.

If remains to see that this choice of parameters is consistent. Note that the only constraints that we use in this proof are that $p = \lambda^{\omega(1)}$ (so that DDH is hard), $(p-1)/2 = q > 2N^2 + 1 = \mathrm{poly}(\lambda, \log q, \alpha)$ (in order to be able to use Theorem 1) and $\alpha > 4 \lceil \log q \rceil + 2$ (to get sufficient homomorphic capacity). Clearly, if p is exponential in λ (so α is polynomial in λ) then all of these constraints are satisfied. □

Lemma 5. *Let p be a prime, and let $S = \{(X, Y) : X = Y + 1; X, Y \in \mathrm{QR}(p)\}$. Then, $|S| = (p-3)/4$ if $p = 3 \bmod 4$, and $|S| = (p-5)/4$ if $p = 1 \bmod 4$.*

Proof. Let $T = \{(u, v) : u \neq 0, v \neq 0, u^2 - v^2 = 1 \bmod p\}$. Since X and Y each have exactly two nonzero square roots if they are quadratic residues, we have that $|T| = 4 \cdot |S|$. It remains to establish the cardinality of T.

For each pair $(u, v) \in T$, let $a_{uv} = u + v$. We claim that distinct pairs in T cannot have the same value of a_{uv}. In particular, each a_{uv} completely determines both u and v as follows. We have $u^2 - v^2 = 1 \to (u-v)(u+v) = 1 \to u - v = 1/a_{uv}$, and therefore $u = (a_{uv} + a_{uv}^{-1})/2$, and $v = (a_{uv} - a_{uv}^{-1})/2$. We therefore have $|U| = |T|$, where $U = \{a \neq 0 : a + a^{-1} \neq 0, a - a^{-1} \neq 0\}$.

We have that $a \in U$, unless $a = 0$, $a^2 = -1 \bmod p$, or $a = \pm 1$. If $p = 1 \bmod 4$, then $-1 \in \mathrm{QR}(p)$, and therefore there are 5 prohibited values of a – i.e., $|U| = p - 5$. If $p = 3 \bmod 4$, then $-1 \notin \mathrm{QR}(p)$, and therefore $|U| = p - 3$. □

A.3 Leveled FHE Based on Worst-Case Hardness

We next describe an instantiation where both the SWHE and the MHE schemes are lattice-based encryption schemes with security based (quantumly) on the hardness of worst-case problems over ideal lattices, in particular ideal-SIVP. This scheme could be Gentry's SWHE scheme [Gen09b, Gen10] one of its variants [SS10, SV10, GH11], or one of the more recent proposals based on the ring-LWE problem [BV11, Pei11]. All these schemes are homomorphic for low-degree polynomials and have threshold-type decryption, in the sense of Section A.1.

The main idea of this construction is to use an *additively* homomorphic encryption (AHE) scheme (e.g., one using lattices) as our *MHE* scheme, by working with discrete logarithms. For a multiplicative group G with order q and generator g, we can view an additively homomorphic scheme AHE with plaintext space $\mathbb{Z}_q$ as a multiplicative homomorphic scheme MHE with plaintext space G: In the MHE scheme, a ciphertext c is decrypted as $\mathsf{MHE.Decrypt}(c) \leftarrow g^{\mathsf{AHE.Decrypt}(c)}$. The additive homomorphism mod q thus becomes a multiplicative homomorphism in G. We can therefore use MHE as a component in chimeric leveled FHE, assuming it is compatible with a suitable SWHE scheme. One caveat is that MHE's $\mathsf{Encrypt}$ algorithm is not obvious. Presumably, to encrypt an element $x \in G$, we encrypt its discrete log using AHE's $\mathsf{Encrypt}$ algorithm, but this requires computing discrete logs in G. Fortunately, in our instantiation we can choose a group G of polynomial size, so computing discrete log in G can be done efficiently.

The main difficulty is to set the parameters so that the component schemes each have enough homomorphic capacity to do their jobs.

This sort of compatibility was easy for the Elgamal-based instantiation, since the parameters of the Elgamal scheme do not grow with the multiplicative homomorphic capacity required of the Elgamal scheme; Elgamal's multiplicative homomorphic capacity is *infinite*, regardless of parameters. On the other hand, the additive homomorphic capacity of a lattice-based scheme is *limited*, as system parameters must grow (albeit slowly) to allow more additions. What makes it possible to set the parameters is the fact that such schemes can handle a *super-polynomial* number of additions.

Below let us fix some SWHE construction which is homomorphic for low-degree polynomials and has threshold-type decryption (e.g., Gentry's scheme [Gen09b, Gen10]). For our construction we will use a polynomial-size plaintext space, namely $\mathbb{Z}_p$ for some $p = \mathrm{poly}(\lambda)$. In more detail, we will use two instances of the same scheme, a "large instance", denoted Lrg, as the SWHE of our Chimeric construction and a "small instance", denoted Sml for the MHE of our Chimeric construction. The plaintext space for Lrg is set as $\mathbb{Z}_p$ for a small prime $p = \mathrm{poly}(\lambda)$, and the plaintext space for Sml is set as $\mathbb{Z}_q$ for $q = p - 1$.

We will use the small instance as a multiplicative homomorphic encryption scheme with plaintext space $\mathbb{Z}_p^*$. Below let g be a generator of $\mathbb{Z}_p^*$. Encryption of a plaintext $x \in \mathbb{Z}_p^*$ under this MHE scheme consists of first computing the discrete logarithm of x to the base g, i.e., $e \in \mathbb{Z}_q$ such that $g^e = x \pmod p$, then encrypting e under Sml. Similarly, MHE decryption of a ciphertext c consists of using the "native decryption" of Sml to recover the "native plaintext" $e \in \mathbb{Z}_q$, then exponentiating $x = g^e \bmod p$.

The homomorphic capacity of Lrg must be large enough to evaluate the decryption of Sml followed by exponentiation mod p and then a quadratic polynomial. The parameters of Sml can be chosen much smaller, since it only needs to support addition of polynomially many terms and not even a single multiplication.[6]

A.3.1 Decryption under Sml

The small instance has n bits of secret key, where n is some parameter to be determined later (selected to support large enough homomorphic capacity to evaluate linear polynomials with polynomially many terms.) Since native decryption of Sml is of the form of Equation (4), decryption under the MHE scheme has the following formula

$$\mathsf{MHE.Dec}_{sk}(c) \;=\; g^c \cdot g^{\sum_{i=1}^n u'_i s_i} \cdot g^{\lceil 2^{-\kappa} \sum_{i=1}^n u''_i s_i \rfloor} \mod p \tag{6}$$

where $(c, \{u'_{i\bullet} u''_i\})$ is the post-processed ciphertext (with $u'_i \in \mathbb{Z}_q$ and $u''_i \in \mathbb{Z}_{2^\kappa}$, and $\kappa = \lceil \log(n+1) \rceil$). Below we show how this formula can be evaluated as a rather low-degree arithmetic circuit.

The complicated part. To evaluate the "complicated part", $\lceil 2^{-\kappa} \sum_{i=1}^n u''_i s_i \rfloor$, as an arithmetic circuit mod p (with input the bits s_i), we will construct a mod-p circuit that outputs the binary representation of the sum. We have n binary numbers, each with κ bits, and we need to add them over the integers and then ignore the lower κ bits. Certainly, each bit of the result can be expressed mod-p as a multilinear polynomial of degree only $n \cdot \kappa$ over the $n \cdot \kappa$ bits of the addends. It is challenging, however, to show that these low-degree representations can actually be computed efficiently.

In any case, we can compute the sum using polynomials of degree $n \cdot \kappa^c$ for small c, easily as follows: Consider a single column $\vec{x} \in \{0,1\}^n$ of the sum. Each bit in the binary representation of the Hamming weight of $\vec{x}$ can be expressed as a mod-p multilinear symmetric polynomial of degree n over $\vec{x}$. After using degree n to obtain the binary representation of the Hamming weight of each column, it only remains to add the κ κ-bit Hamming weights together (each Hamming weight shifted appropriately depending on the significance of its associated column) using degree only κ^c. Adding numbers κ κ-bit numbers is in NC^1, and in particular can be accomplished with low degree using the "3-for-2" trick (see [KR]), repeatedly replacing each three addends by two addends that correspond to the XOR and CARRY (and hence have the same sum), each replacement only costing constant degree, and finally summing the final two addends directly. Over $\mathbb{Z}_p$, the 3-for-2 trick is done using the formulas

$$\begin{aligned} XOR(x,y,z) &= 4xyz - 2(xy + xz + yz) + x + y + z \\ CARRY(x,y,z) &= xy + xz + yz - 2xyz \end{aligned}$$

The simple part and exponentiation. Although it is possible to compute the simple part similarly to the complicated part, it is easier to just push this computation into the exponentiation step. Specifically, we now have a κ-bit number v_0 that we obtained as the result of the "complicated part", and we also have the $\lceil \log q \rceil$-bit numbers $v_i = u'_i s_i$ for $i = 1, \ldots, n$ (all represented in

[6]The "small" scheme could also be instantiated from other additively homomorphic lattice-based schemes, e.g., one of Regev's schemes [Reg04, Reg09], or the GPV scheme [GPV08], etc.

binary), and we want to compute $g^c \cdot g^{\sum_{i=0}^n v_i} \cdot \bmod p$. Denote the binary representation of each v_i by $(v_{it} \dots v_{i1} v_{i0})$, namely $v_i = \sum_j v_{ij} 2^j$. Then we compute

$$\begin{aligned} g^{c+(\sum_{i=0}^n v_i)} &= g^{c+(\sum_{i,j} v_{ij} 2^j)} = g^c \cdot \prod_{i,j} (g^{2^j})^{v_{ij}} = g^c \cdot \prod_{i,j} \Big(v_{i,j} \cdot g^{2^j} + (1 - v_{ij}) \cdot 1 \Big) \\ &= \underbrace{\prod_{j=0}^{\kappa} \Big(1 + v_{0,j} \cdot (g^{2^j} - 1) \Big)}_{\text{``complicated part''}} \cdot \underbrace{g^c \cdot \prod_{i=1}^{n} \prod_{j=0}^{\lceil \log q \rceil} \Big( 1 + v_{i,j} \cdot (g^{2^j} - 1) \Big)}_{\text{``simple part''}} \end{aligned}$$

The terms g^c and $(g^{2^j} - 1)$ are known constants in $\mathbb{Z}_p$, hence we have a representation of the decryption formula as an arithmetic circuit mod p.

To bound the degree of the complicated part, notice that v_0 has κ bits, each a polynomial of degree at most $n \cdot \text{poly}(\kappa)$, hence the entire term has degree bounded by $n \cdot \text{poly}(\kappa)$. For the simple part, all the v_i's together have $n \lceil \log q \rceil$ bits (each is just a variable), so the degree of that term is bounded by just $n \lceil \log q \rceil$. Hence the total degree of the decryption formula is $\tilde{O}(n)$, assuming q is quasi-polynomial in n. One can also verify that the number of terms is $2^{\tilde{O}(n)}$. (Known lattice-based SWHE schemes have $n = \tilde{O}(\lambda)$, in which case Sml's decryption has degree $\tilde{O}(\lambda)$.)

A.3.2 The SWHE scheme Lrg.

The large instance has N bits of secret key, where N is some parameter to be determined later, selected to support large enough homomorphic capacity to be compatible with Sml. As explained in Section 2, the decryption of Lrg can be expressed as a restricted depth-3 circuit of degree at most $2N^2$ and with at most $2N^2 + N + 1$ product gates. Note that the number of summands in the top addition is at most $2N^2 + N + 1 < 3N^2$.

A.3.3 Setting the parameters.

Lemma 6. *Let Lrg and Sml be as above. We can choose the parameters of Lrg and Sml so that Lrg is chimerically compatible with the MHE derived from Sml.*

Proof. Denoting the security parameter by λ, below we choose the plaintext spaces and parameters α, β, where Lrg can support polynomials of degree up to α with 2^α terms, Sml can support linear polynomials with up to 2^β terms, so as to get chimerically compatible schemes. Note that making the plaintext spaces of the two schemes compatible is simple, all we need to do is choose a prime p and set $q = p - 1$, and let the plaintext spaces of Lrg, Sml be $\mathbb{Z}_p$ and $\mathbb{Z}_q$, respectively. In terms of size constraints on the parameters, we have the following:

- $p > 2N^2$, so that we can use Theorem 1.
- $p = \text{poly}(\lambda)$, so that we can compute discrete logs modulo p efficiently.
- $\beta \geq \log(2N^2) = 2 \log N + 1$, since the restricted depth-3 circuits for the decryption of Lrg all have degree at most $2N^2$, hence we need an MHE scheme that supports $2N^2$ products, which means that Sml should support linear functions with $2N^2$ terms.

- α is at least twice the degree of Sml's decryption, so that we can compute a multiplication within Lrg after evaluating Sml's decryption function.

Up front, we are promised polynomial bounds $K_{\mathsf{Sml}}, K_{\mathsf{Lrg}}$ such that the key-size of Sml is bounded by $n \leq K_{\mathsf{Sml}}(\lambda, \log q, \beta)$ and the key-size of Lrg is bounded by $N \leq K_{\mathsf{Lrg}}(\lambda, \log p, \alpha)$.

Assuming $N = \mathrm{poly}(\lambda)$ (we establish this later), we can meet the first three constraints by choosing a prime $p \in [2N^2+1, 4N^2]$ (such a prime must exist and can be found efficiently) and $\beta = \log p$. Then $K_{\mathsf{Sml}}(\lambda, \log q, \beta) = o(\lambda^{c_{\mathsf{Sml}}+\epsilon})$ for any $\epsilon > 0$ and some constant c_{Sml}. We argued that before that when Sml has n-bit keys, decryption can be computed with degree $\tilde{O}(n \cdot (\log^{c_2} n + \log q))$ for some constant c_2. Therefore, still assuming that $N = \mathrm{poly}(\lambda)$, all of the constraints can be satisfied with $\alpha = \theta(\lambda^{c_{\mathsf{Sml}}+\epsilon})$ for any $\epsilon > 0$. But then of course N can be $\mathrm{poly}(\lambda)$ since it is bounded by $K_{\mathsf{Lrg}}(\lambda, \log p, \alpha)$. □

Using Gentry's scheme and proof [Gen09b, Gen10] we get:
Corollary 1. *There exists a leveled FHE, whose security is reducible via quantum reduction to the worst-case hardness of S(I)VP in ideal lattices, ideal-SIVP.* ■

B Proof of Lemma 1

Proof. (Lemma 1) Every multilinear symmetric polynomial $M(\vec{x})$ is a linear combination of the elementary symmetric polynomials: $M(\vec{x}) = \sum_{i=0}^{n} \ell_i \cdot e_i(\vec{x})$. Given the evaluation $M(\vec{x})$ over binary vectors $\vec{x} = 1^i 0^{n-i}$, we can compute the ℓ_i's as follows. We obtain the constant term $\ell_0 \cdot e_0(\vec{x}) = \ell_0$ by evaluating M at 0^n. We obtain ℓ_k recursively via

$$M(1^k0^{n-k}) = \sum_{i=0}^{n} \ell_i \cdot e_i(1^k0^{n-k}) = \ell_k + \sum_{i=0}^{k-1} \ell_i \cdot e_i(1^k0^{n-k})$$

$$\Rightarrow \quad \ell_k = M(1^k0^{n-k}) - \sum_{i=0}^{k-1} \ell_i \cdot e_i(1^k0^{n-k}) = M(1^k0^{n-k}) - \sum_{i=0}^{k-1} \ell_i \cdot \binom{k}{i}$$

At this point, it suffices to prove the lemma just for the elementary symmetric polynomials. This is because we have shown that we can efficiently obtain a representation of $M(\vec{x})$ as a linear combination of the elementary symmetric polynomials, and we can clearly use the known ℓ_j values to "merge" together the depth-3 representations of the elementary symmetric polynomials that satisfy the constraints of Lemma 1 into a depth-3 representation of M that satisfies the constraints.

For each i, the value $e_i(\vec{x})$ is the coefficient of z^{n-i} in the polynomial $P(z)$. We can compute the coefficients of $P(z)$ via interpolation from the values $P(a)$, $a \in A$. Therefore, each value $e_i(\vec{x})$ can be computed by a $\mathcal{L}_A$-restricted depth-3 arithmetic circuit as follows: using $n+1$ product gates, compute the values $P(a)$, $a \in A$, and then (as the final sum gate), interpolate the coefficient of z^{n-i} from the $P(a)$ values. □