Fully Homomorphic Encryption without Squashing Using Depth-3 Arithmetic Circuits

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

April 14, 2011

Abstract

All currently known fully homomorphic encryption (FHE) schemes 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.

We describe a *new approach* that 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, obtaining the following results:

- *A "simple" FHE scheme where we replace SSSP with Decision Diffie-Hellman.*
- *The first FHE scheme based entirely on worst-case hardness*. Specifically, we describe a "leveled" FHE scheme whose security can be quantumly reduced to the approximate shortest independent vector problem over ideal lattices (ideal-SIVP).
- *Some efficiency improvements for FHE.* While at present our new method does not improve computational efficiency, we do provide an optimization that reduces the ciphertext length. For example, at one point, the entire FHE ciphertext may consist of a single Elgamal ciphertext!

Our new method does not eliminate the bootstrapping step. Whether this can be done remains an intriguing open problem. As in the previous blueprint, we can get "pure" (non-leveled) FHE by assuming circular security.

Our 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. We then translate the result back to the SWHE scheme by homomorphically evaluating the decryption function of the MHE scheme. (Due to the special form of the circuit, switching to the MHE scheme can be done without having to evaluate anything homomorphically.) 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.

1 Introduction

Fully homomorphic encryption allows a worker to receive encrypted data and perform arbitrarily-complex dynamically-chosen computations on that data while it remains encrypted, despite not having the secret decryption key. Several fully homomorphic encryption (FHE) schemes appeared recently [3, 17, 14, 5], all following the same blueprint as Gentry's original construction [3, 2]:

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.

Here, we describe a new blueprint for FHE: 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 does not use squashing, and does not rely on the assumed hardness of the sparse subset sum problem (SSSP). With our new blueprint, we construct a "simple" leveled FHE scheme where SSSP is replaced with Decision Diffie-Hellman. We also construct the first 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 [4]). Finally, we get shorter ciphertexts: at one point during the bootstrapping process, the entire leveled FHE ciphertext consists of a single MHE (e.g., Elgamal) ciphertext!

We do not eliminate the bootstrapping step. Whether this can be done remains an intriguing open problem. 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 (homomorphically) evaluate 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 [3, 17, 14, 5] evaluated those polynomials in the "obvious way" using *boolean* circuits. Instead, here we use Ben-Or's observation (reported in [10]) 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 $t_j = P(a_j)$, and finally interpolating the coefficient of interest as a linear combination of the t_j's. The resulting circuit has the form $e_k(\vec{x}) = \sum_{j=1}^{n+1} \lambda_{jk} \prod_{i=1}^{n}(a_j + x_i) \pmod{p}$, 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).

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

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 (indeed it *increases* the degree).[2] Our insight is that to evaluate Ben-Or's high-degree $\sum\prod\sum$ circuit, we can evaluate the $\prod$ part by temporarily working with a MHE scheme, such as Elgamal [1]. 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 Example: An Elgamal-Based Instantiation

Perhaps the simplest illustration of our idea is using Elgamal encryption to do the multiplication. So, we sketch a non-optimized Elgamal-based instantiation here in the Introduction.

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 without squashing [3, 17, 14, 5] can be adapted to handle this large plaintext space).

We require the SWHE scheme to have a decryption function that can be expressed as a "restricted" depth-3 arithmetic circuit. We define "restricted" in Section 2. For now, we just mention that Ben-Or's observation implies that all multilinear symmetric polynomials can be evaluated by depth-3 circuits that are "restricted" according to our definition. Also, we will show that many lattice-based decryption schemes, including all known SWHE schemes [3, 17, 14, 5], have decryption functions that can expressed in this way.

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 can be expressed as a depth-3 circuit that is "restricted" in the sense that the bottom level of sums are of the form $a_j + x_i$ – i.e., linear equations with only *one* variable x_i. Assume the circuit 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 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 $a_j, a_j + 1 \in \mathrm{QR}(p)$, hence 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

[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 $k \leq m$.

[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 $p = 1 \bmod 4$.

$a_j + (s_i \cdot c_i) = a_j + s_i$, 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 easy to obtain Elgamal ciphertexts that encrypt 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 now need to convert these Elgamal encryptions into SWHE encryptions of the same values in $\mathbb{Z}_p$. To do that we use the SWHE ciphertexts that encrypt the bits of the Elgamal secret exponent. Denote by e_i the i'th bit of the secret exponent e, 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 $i \in [1, \lfloor \log q \rfloor]$. Using the SWHE encryptions of the bits e_i, we 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, in our new blueprint, 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.

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 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-referential property that requires squashing. We note, however, that lattice-based additively-homomorphic schemes are not completely error free, so there is a new subtlety here. The noise in these schemes grows with the number of summands (and the number of summands is roughly equal to the size of the SWHE ciphertext), so increasing the SWHE parameters will result in more summands and therefore a more noisy AHE ciphertext after addition. This in turn may require larger parameters of the AHE scheme to handle this more noisy ciphertext, which again means that we need larger SWHE parameters to evaluate the AHE decryption. This may seem like it would yield self-referential dependence again, but the dependence of the AHE noise on the number of summands is very weak (only logarithmic), so the self-referential property can be addressed without the need for squashing. See Appendix A.2 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 [8] have already shown that the decryption functions of Regev's cryptosystems [12, 13] 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 our generic construction (Section 3), the L_j's are not required to be linear. However, they are linear in our actual instantiations, in the depth-3 decryption circuits for known SWHE schemes.*

We will 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 [10]). *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 $\mathcal{L}_A \overset{\text{def}}{=} \{(a + x_i) : a \in A,\ 1 \leq i \leq n\}$. 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 appropriate 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 allows us to 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 univariate 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.

Lemma 2. *Let T be a positive integer. Associated to inputs $\vec{x} \in \{0,1\}^{Tn}$, there is a "universal" multilinear symmetric polynomial $M(\vec{x})$ such that for all univariate polynomials $f(x)$ and all $t_1, \ldots, t_n \in \{0, \ldots, T\}$,*

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

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

Proof. For $\vec{x} \in \{0,1\}^{Tn}$, let $g(\vec{x}) = f(\sum x_i)$. Then, we have

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

The polynomial $g(\vec{x})$ is symmetric. There is a multilinear symmetric polynomial $M(\vec{x})$ that agrees with $g(\vec{x})$ on the boolean cube (this comes from "flattening" the high powers of $g(\vec{x})$). By Lemma 1, we can compute a $\mathcal{L}_A$-restricted depth-3 circuit representation of $M(\vec{x})$ by evaluating $g(\vec{x})$ over the vectors $\vec{x} = 1^i 0^{Tn-i}$, which we can accomplish using the f-oracle. $\square$

2.2 Decryption in Lattice-Based Cryptosystems

In GGH-type [7] 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 $\vec{e} = \vec{m} \pmod p$. In Appendix C we show 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.

Somewhat similarly to [3], 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 [3], 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.

(We also need to ensure that ciphertexts are close enough to the lattice so that using the decryption formula with full-precision $\vec{u}$'s would have resulted in a vector which is less than $1/2(N+1)$ away from $\mathbb{Z}^n$.)

For simplicity hereafter, we will assume $\vec{m} = \langle 0, \ldots, 0, m\rangle$ – 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 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 secret key is $\vec{s} \in \{0,1\}^N$, and the plaintext is recovered as:

$$m \quad \leftarrow \quad \underbrace{c - \sum s_i \cdot u'_i}_{\text{"simple part"}} - \underbrace{\left\lceil 2^{-\kappa} \cdot \sum s_i \cdot u''_i \right\rfloor}_{\text{"complicated part"}} \bmod p. \tag{1}$$

We now show that we can compute Equation (1) using a $\mathcal{L}_A$-restricted circuit.

Lemma 3. *Let prime $p > 2N^2$. Regarding the "complicated part" of Equation (1), 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 prime $p > 2N^2$. For any $A \subseteq \mathbb{Z}_p$ of cardinality at least $2N^2 + 1$, $\mathcal{E}$'s decryption function (Equation (1)) 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 universal multilinear symmetric polynomial $M(\vec{x})$ taking $2N^2$ inputs such that

$$f(\sum s_i \cdot u''_i) = M(\underbrace{s_1, \ldots, s_1}_{u''_1 \text{ times}}, \underbrace{0, \ldots, 0}_{2N - u''_1 \text{ times}}, \underbrace{s_2, \ldots, s_2}_{u''_2 \text{ times}}, \underbrace{0, \ldots, 0}_{2N - u''_2 \text{ times}}, \ldots, \underbrace{s_N, \ldots, s_N}_{u''_N \text{ times}}, \underbrace{0, \ldots, 0}_{2N - u''_N \text{ times}})$$

for all $\vec{s} \in \{0,1\}^N$. By Lemma 2, we can compute M's representation as a $\mathcal{L}_A$-restricted depth-3 circuit C efficiently. 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 $(c + N \cdot a_1) - \sum (a_1 + s_i) \cdot u'_i \bmod p$ with the constant term $\lambda_0 = (c + N \cdot a_1)$. 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 within the new blueprint, 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!

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

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

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),

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

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 class $\mathcal{F}'$.*

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

- *There is a polynomial bound D such that, 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.*

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

- *For any secret key $sk \in$ SWHE.KeyGen and any polynomial $L_j \in \mathcal{L}$ we have $L_j(sk) \in \mathcal{P}$.*
- *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, there is a polynomial upper bound B on the number of product gates in all the $\mathcal{L}$-restricted circuits from the first bullet above, such that for any two vectors of MHE ciphertexts $\vec{c} = \langle c_1, \dots c_b \rangle$ and $\vec{c}' = \langle c'_1, \dots c'_{b'} \rangle \in \mathcal{C}^{\le B}_{\mathsf{MHE},\mathcal{F}'}$, 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.,* $\mathsf{DAdd}_{\vec{c},\vec{c}'}(sk), \mathsf{DMul}_{\vec{c},\vec{c}'}(sk) \in \mathcal{F}$.

3.3 Chimeric Leveled FHE: The Construction

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

FHE.KeyGen(λ, ℓ): 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

$$\Big(pk^{(i)}_{SW},\ sk^{(i)}_{SW}\Big) \overset{\text{R}}{\leftarrow} \mathsf{SWHE.KeyGen}\ , \quad \Big(pk^{(i)}_{MH},\ sk^{(i)}_{MH}\Big) \overset{\text{R}}{\leftarrow} \mathsf{MHE.KeyGen}\ .$$

Encrypt the i'th MHE secret key under the $(i+1)$'st SWHE public key, $\overline{sk}^{(i)}_{MH} \leftarrow \mathsf{SWHE.Enc}(pk^{(i+1)}_{SW}, sk^{(i)}_{MH})$. 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 $sk^{(i)}_{SW}$ under $pk^{(i)}_{MH}$, compute for all $L_j \in \mathcal{L}$ the value $m_{ij} \leftarrow L_j(sk^{(i)}_{SW})$ and then encrypt it $c_{ij} \leftarrow \mathsf{MHE.Enc}(pk^{(i)}_{MH}, m_{ij})$. Let $\overline{sk}^{(i)}_{SW}$ denote the collection of all the c_{ij}'s. The public key pk_{FH} consists of $(pk^{(i)}_{SW}, pk^{(i)}_{MH})$ and the encrypted secret keys $(\overline{sk}^{(i)}_{SW}, \overline{sk}^{(i)}_{MH})$ for all i. The secret key sk_{FH} consists of $sk^{(i)}_{SW}$ for all i.

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 $\mathsf{SWHE.Enc}(pk^{(1)}_{SW}, m)$.

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

FHE.Recrypt(pk_{FH}, c): Takes as input the public key and a ciphertext c that is a valid "evaluated SWHE ciphertext" under $pk^{(i)}_{SW}$, and outputs a "refreshed" SWHE ciphertext c', encrypting the same plaintext but under $pk^{(i+1)}_{SW}$. 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 $pk_{MH}^{(i+1)}$ 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 [2]. We omit the details.

4 Optimizations

In the Products step of the 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 Products step. In Section 4.2, we extend this optimization so that the entire leveled FHE ciphertext after the 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 1), 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 universal multilinear symmetric polynomial, the products are all of the form:

$$\lambda_j \cdot P(a_j) \text{ for all } a \in A, \text{ where } P(z) = \prod (z + s_i)^{u_i''} \cdot z^{2N - u_i''}$$

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. In terms of discrete logs, we have $\mathrm{DL}(P(a_j)) = \mathrm{DL}(a_j) \cdot 2N^2 + ((\mathrm{DL}(a_j + 1) - \mathrm{DL}(a_j)) \cdot \sum_{s_i=1} u_i''$. That is, all of the $P(a_j)$'s are determined by the small integer $\sum_{s_i=1} u_i''$, a number implicit in $P(a_1)$.

We choose the a_j's in a very simple way: so that, for all $j > 1$, we know integers (w_j, e_j) such that:

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

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

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 doubling 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 $a, a+1 \in \mathrm{QR}(p)$. 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 1, 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 recover m just from $(h(c), \{h(u'_i)\}, \{h(u''_i)\})$, where $h(x) = x \bmod r \in \{0, \ldots, r-1\}$ is a "hash" function associated to a "large-enough" polynomial-size prime r. Moreover, after hashing the ciphertext components down smaller than 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 $t_r = h(2^\kappa \cdot c) + \sum s_i \cdot h(-2^\kappa \cdot u'_i - u''_i)$ for $h(x) = x \bmod r \in \{0, \ldots, r-1\}$.*

Proof. Let $t = 2^\kappa(c - \sum s_i \cdot u'_i) - \sum s_i \cdot u''_i$. The original decryption formula (Equation 1) 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 h has no collisions over the support of t, t can be recovered from $h(t)$. Note that $h(t) = h(t_r)$. Thus m can be recovered from t_r (via $h(t_r) = h(t)$, 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 $(h(2^\kappa \cdot c), \{h(-2^\kappa \cdot u'_i - u''_i)\})$ that encrypt $m \in \{0,1\}$, we have*

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

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 $P(a_1)$.

References

[1] Taher Elgamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *CRYPTO*, pages 10–18, 1984.

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

[3] Craig Gentry. Fully homomorphic encryption using ideal lattices. In Michael Mitzenmacher, editor, *STOC*, pages 169–178. ACM, 2009.

[4] Craig Gentry. Toward basing fully homomorphic encryption on worst-case hardness. In *Advances in Cryptology - CRYPTO'10*, volume 6223 of *Lecture Notes in Computer Science*, pages 116–137. Springer, 2010.

[5] Craig Gentry and Shai Halevi. Implementing gentry's fully-homomorphic encryption scheme. manuscript, `https://researcher.ibm.com/researcher/view_project.php?id=1548`, 2010. To appear in Eurocrypt 2011.

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

[7] Oded Goldreich, Shafi Goldwasser, and Shai Halevi. Public-key cryptosystems from lattice reduction problems. In *Advances in Cryptology - CRYPTO'97*, volume 1294 of *Lecture Notes in Computer Science*, pages 112–131. Springer, 1997.

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

[9] Daniele Micciancio. Improving lattice based cryptosystems using the hermite normal form. In Joseph H. Silverman, editor, *CaLC*, volume 2146 of *Lecture Notes in Computer Science*, pages 126–145. Springer, 2001.

[10] 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".

[11] Chris Peikert, 2011. Private communication.

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

[13] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In Harold N. Gabow and Ronald Fagin, editors, *STOC*, pages 84–93. ACM, 2005.

[14] Nigel P. Smart and Frederik Vercauteren. Fully homomorphic encryption with relatively small key and ciphertext sizes. In *Public Key Cryptography - PKC'10*, volume 6056 of *Lecture Notes in Computer Science*, pages 420–443. Springer, 2010.

[15] Damien Stehlé and Ron Steinfeld. Faster fully homomorphic encryption. Cryptology ePrint Archive, Report 2010/299, 2010. `http://eprint.iacr.org/`.

[16] Vinod Vaikuntanathan, 2011. Private communication.

[17] 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`.

[18] Wikipedia. Chimera. `http://en.wikipedia.org/wiki/Chimera`, 2011.

A Instantiations of Chimeric FHE

A.1 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.

The first lemma is designed to cover a common case. Known SWHE schemes [3, 14, 17, 4, 5] are capable of evaluating polynomials of degree up to $\alpha(\lambda_{\mathsf{SWHE}})$ having up to $2^{\alpha(\lambda_{\mathsf{SWHE}})}$ monomials, where α is some polynomial and λ_{SWHE} is the security parameter of the SWHE scheme. Also, for our chimeric leveled FHE construction to work, we need the decryption function of the MHE scheme to be not-too-complex: we need that MHE's decryption function can be expressed as a (ciphertext-dependent) polynomial of degree at most $\beta(\lambda_{\mathsf{MHE}})$ having at most $2^{\beta(\lambda_{\mathsf{MHE}})}$ monomials, where β is some polynomial and λ_{MHE} is the security parameter of the MHE scheme. The following lemma answers, in general, what further properties SWHE and MHE need to have to be chimerically compatible.

Lemma 5. *Let* SWHE *and* MHE *be encryption schemes with security parameters* λ_{SWHE} *and* λ_{MHE}*, and plaintext spaces* $\mathbb{Z}_p$ *and* $\mathcal{P} \subset \mathbb{Z}_p$*. Suppose that* SWHE*'s decryption can be expressed as an* $\mathcal{L}$*-restricted circuit over* $\mathbb{Z}_p$ *with* $\mathcal{L}$*-degree* D *and* B *product gates, where* $|\mathcal{L}|$*,* D *and* B *are polynomial in* λ_{SWHE}*. Suppose also that, for some functions* α*,* β*,* γ*:*

- SWHE*'s homomorphic capacity includes all polynomials over* $\mathbb{Z}_p$ *of degree at most* $\alpha(\lambda_{\mathsf{SWHE}})$ *having at most* $2^{\alpha(\lambda_{\mathsf{SWHE}})}$ *monomials.*

- MHE*'s decryption function can be expressed as a (ciphertext-dependent) polynomial of degree at most* $\beta(\lambda_{\mathsf{MHE}})$ *having at most* $2^{\beta(\lambda_{\mathsf{MHE}})}$ *monomials.*

- MHE*'s homomorphic capacity includes products of degree up to* $\gamma(\lambda_{\mathsf{MHE}})$.

Then, the schemes SWHE *and* MHE *are chimerically compatible if the following hold:*

- *Property 1: The plaintext spaces of* SWHE *and* MHE *are compatible (i.e., for any secret key* $sk \in$ $\mathsf{SWHE.KeyGen}$ *and any polynomial* $L_j \in \mathcal{L}$ *we have* $L_j(sk) \in \mathcal{P}$*).*

- *Property 2:* $\gamma(\lambda_{\mathsf{MHE}}) \geq D(\lambda_{\mathsf{SWHE}})$.

- *Property 3:* $\alpha(\lambda_{\mathsf{SWHE}})$ *exceeds both* $2\beta(\lambda_{\mathsf{MHE}})$ *and* $\beta(\lambda_{\mathsf{MHE}}) + 2\log B(\lambda_{\mathsf{SWHE}})$.

Proof. Property 2 implies that the homomorphic capacity MHE includes all products of D or less variables, as required for chimeric compatibility. It remains to show that Property 3 implies that SWHE has the required homomorphic capacity – namely, that for any two vectors of MHE ciphertexts $\vec{c} = \langle c_1, \ldots c_b \rangle$ and $\vec{c}' = \langle c'_1, \ldots c'_{b'} \rangle$ with $b, b' \leq B$ 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. The functions $\mathsf{DAdd}_{\vec{c},\vec{c}'}(sk)$ and $\mathsf{DMul}_{\vec{c},\vec{c}'}(sk)$ can each be expressed as a polynomial of degree at most $2\beta(\lambda_{\mathsf{MHE}})$ with $B^2 \cdot 2^{\beta(\lambda_{\mathsf{MHE}})}$ monomials. These functions are with SWHE's homomorphic capacity as long $\alpha(\lambda_{\mathsf{SWHE}})$ exceeds $2\beta(\lambda_{\mathsf{MHE}})$ and $\log B^2 \cdot 2^{\beta(\lambda_{\mathsf{MHE}})} = \beta(\lambda_{\mathsf{MHE}}) + 2\log B(\lambda_{\mathsf{SWHE}})$. □

Now we address the Elgamal instantiation.

Theorem 4. *Let* $p = 2q + 1$ *be a safe prime. Let* $\tau = \lceil \log q \rceil$. *Let* MHE *be Elgamal with plaintext space* $\mathrm{QR}(p)$. *Suppose* SWHE *is a SWHE scheme with plaintext space* $\mathbb{Z}_p$ *such that*

- SWHE*'s decryption can be expressed (for any* $A \subset \mathbb{Z}_p$ *of cardinality at least* $N_A \leq q - 1$ *and* $\mathcal{L}_A = \{(a + x_i) : a \in A,\ 1 \leq i \leq n\}$*) as an* $\mathcal{L}_A$*-restricted circuit over* $\mathbb{Z}_p$ *with* $\mathcal{L}_A$*-degree* D *and* B *product gates, where the secret key bits* $\{x_i\}$ *are restricted to* $\{0, 1\}$*, and* N_A*,* D *and* B *are polynomial in the security parameter.*

- SWHE*'s homomorphic capacity includes all polynomials over* $\mathbb{Z}_p$ *of degree up to* $\alpha(\lambda_{\mathsf{SWHE}})$ *with up to* $2^{\alpha(\lambda_{\mathsf{SWHE}})}$ *monomials, where* $\alpha(\lambda_{\mathsf{SWHE}})$ *exceeds both* $4\tau + 2$ *and* $2\tau + 1 + 2\log B(\lambda_{\mathsf{SWHE}})$.

Then, SWHE *and and* MHE *are chimerically compatible schemes.*

Proof. Regarding Property 1 of Lemma 5, we need to establish that we can choose A so that, for any $sk \in \mathsf{SWHE.KeyGen}$ and any polynomial $L_j \in \mathcal{L}$ we have $L_j(sk) \in \mathrm{QR}(p)$. In Lemma 6 below, we show that there are $q - 1 = (p - 3)/4$ values a such that $a, a + 1 \in \mathrm{QR}(p)$. Since N_A is polynomial and at most $q - 1$, we can populate A with N_A such values efficiently. The value $a + x_i$ for $a \in A$ and secret key bit x_i is always in $\mathrm{QR}(p)$, MHE's plaintext space. Therefore Property 1 is satisfied.

Property 2 of Lemma 5 is satisfied trivially since the multiplicative homomorphic capacity of Elgamal is infinite.

Finally, we need to show that Property 3 is satisfied. It suffices to show that Elgamal decryption can be computed using a polynomial of degree $\beta(\tau)$ with at most $2^{\beta(\tau)}$ monomials for $\beta(\tau) = 2\tau + 1$.

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 $e_{\tau-1} \ldots e_1 e_0$, decryption of the post-processed ciphertext now becomes

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

We will be overly conservative and treat $z, y_0, \ldots, y_{\tau-1}$ as variables; then the degree of the polynomial above is $2\tau + 1$, and it has 2^τ monomials. Setting $\beta(\tau) = 2\tau + 1$, the theorem follows. □

Lemma 6. *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 $u^2 - v^2 = 1 \rightarrow (u - v)(u + v) = 1 \rightarrow u - v = 1/a_{uv}$, 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 $a \neq 0$, $a^2 \neq -1 \bmod p$, or $a \neq \pm 1$. 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 $|U| = p - 3$. □

From Theorem 4, it should be clear that we can instantiate chimeric leveled FHE with Elgamal and known SWHE schemes. In particular, the homomorphic capacity required of SWHE is independent of its own decryption function, except that the number of monomials SWHE must sum grows quadratically with B (the number of product gates in its restricted circuit). However, the parameters of SWHE only need to grow logarithmically with B, and therefore this weak dependence is not a problem. We formalize this in the following theorem.

Theorem 5. *Chimeric leveled FHE can be instantiated with Elgamal and known SWHE schemes.*

Proof. Let $p = 2q + 1$ be a safe prime, and $\tau = \lceil \log q \rceil$. Let MHE be Elgamal with plaintext space $\mathrm{QR}(p)$. We only need to show the existence of a SWHE scheme SWHE that is compatible with MHE – that is, a scheme with plaintext space $\mathbb{Z}_p$ such that

- SWHE's decryption can be expressed (for any $A \subset \mathbb{Z}_p$ of cardinality at least $N_A \leq q - 1$ and $\mathcal{L}_A = \{(a + x_i) : a \in A,\ 1 \leq i \leq n\}$) as an $\mathcal{L}_A$-restricted circuit over $\mathbb{Z}_p$ with $\mathcal{L}_A$-degree D and B product gates, where the secret key bits $\{x_i\}$ are restricted to $\{0, 1\}$, and N_A, D and B are polynomial in the security parameter.

- SWHE's homomorphic capacity includes all polynomials over $\mathbb{Z}_p$ of degree up to $\alpha(\lambda_{\mathsf{SWHE}})$ with up to $2^{\alpha(\lambda_{\mathsf{SWHE}})}$ monomials, where $\alpha(\lambda_{\mathsf{SWHE}})$ exceeds both $4\tau + 2$ and $2\tau + 1 + 2 \log B(\lambda_{\mathsf{SWHE}})$.

We saw (Section 2.2, also Appendix C) that known SWHE schemes [3, 14, 17, 4, 5] have decryption functions that can be expressed as suitable $\mathcal{L}_A$-restricted circuits where the associated values $N_A(\lambda_{\mathsf{SWHE}})$, $D(\lambda_{\mathsf{SWHE}})$, and $B(\lambda_{\mathsf{SWHE}})$ are polynomial in SWHE's security parameter λ_{SWHE}. Moreover, for some polynomial α, these schemes can evaluate polynomials of degree $\alpha(\lambda_{\mathsf{SWHE}})$ with $2^{\alpha(\lambda_{\mathsf{SWHE}})}$ monomials. It suffices to set λ_{SWHE} so that $\alpha(\lambda_{\mathsf{SWHE}})$ exceeds $4\tau + 2$ and $2\tau + 1 + 2\log B(\lambda_{\mathsf{SWHE}})$. $\square$

A.2 Leveled FHE Based on Worst-Case Hardness

We now 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 [3, 4] one of its variants [15, 14, 5], or one of the more recent proposals based on the ring-LWE problem [16, 11]. As mentioned above, these schemes have the property that, for some polynomial α, they can evaluate polynomials of degree up to $\alpha(\lambda)$ with up to $2^{\alpha(\lambda)}$ monomials, where λ is the security parameter.

The main idea of this construction is to use an *additively* homomorphic encryption (AHE) scheme (e.g., one using lattices) as our *MHE* scheme. 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. The only 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 showing that the component schemes are *compatible* – in particular, that the component schemes each have enough homomorphic capacity to do their jobs. This sort of compatibility was easy to see for the Elgamal-based instantiation, since the parameters of the Elgamal scheme (and hence the homomorphic capacity required of the SWHE 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*. The additive homomorphic capacity is *almost unlimited*, in that such schemes can handle a *super-polynomial* number of additions with reasonable parameter choices – quite unlike the situation with multiplicative homomorphic capacity, where current SWHE schemes can handle only polynomial degree. But still, some care is necessary to deal with a minor feedback loop in our parameter choices.

We begin with a high-level overview. Let λ_{SWHE} and λ_{AHE} be the security parameters of the our SWHE and AHE schemes. Consider the operations that SWHE must perform homomorphically as part of SWHE's depth-3 decryption circuit. First, it decrypts MHE ciphertexts, where MHE is implemented as an AHE scheme followed by exponentiation. For some polynomial β, AHE decryption – i.e., decrypting an AHE ciphertext to a plaintext in $[0, q)$ represented in binary – can be expressed as a polynomial with degree at most $\beta(\lambda_{\mathsf{AHE}})$ and at most $2^{\beta(\lambda_{\mathsf{AHE}})}$ monomials. To perform *MHE* decryption, we then need to exponentiate by the result, which increases the degree to $\beta(\lambda_{\mathsf{AHE}}) \cdot \log q$ and the number of monomials to about $2^{\beta(\lambda_{\mathsf{AHE}}) \cdot \log q}$. Next, we need to perform some polynomial number $B(\lambda_{\mathsf{SWHE}})$ of additions in the $\mathsf{Summation}$ step of SWHE's decryption circuit, which does not further increase the degree, but increases the number of monomials by a factor of $B(\lambda_{\mathsf{SWHE}})$. Finally, we need to be able to perform at least one Mult inside SWHE, which doubles the degree and squares the number of monomials. Overall, the SWHE scheme needs to be capable of handling polynomials of degree k with 2^k monomials for $k = O(\log B(\lambda_{\mathsf{SWHE}}) + \beta(\lambda_{\mathsf{AHE}}) \cdot \log q)$.

Fortunately, for some polynomial α, SWHE can be made to handle polynomials of degree up to $\alpha(\lambda_{\mathsf{SWHE}})$ with $2^{\alpha(\lambda_{\mathsf{SWHE}})}$ monomials, and it suffices to make $\alpha(\lambda_{\mathsf{SWHE}})$ larger than $\log B(\lambda_{\mathsf{SWHE}}) + \beta(\lambda_{\mathsf{AHE}}) \cdot \log q$. For this, it suffices to take λ_{SWHE} to be a sufficiently large polynomial of λ_{AHE}. We also need to make sure that the MHE scheme has sufficient homomorphic capacity to handle $D(\lambda_{\mathsf{SWHE}})$ multiplications for some polynomial D, but this is easy to do since the capacity of a lattice-based AHE scheme can be made super-polynomial. More details are provided in the following subsections.

A.2.1 Gentry's Somewhat-Homomorphic Scheme

Recall that Gentry's scheme [3, 4] works over the Euclidean space $\mathbb{R}^d$ for some dimension d, polynomial in the security parameter, and uses some ring $\mathcal{R}$ which is defined over $\mathbb{Z}^d$. (For example, one may use the ring of polynomials modulo $x^d + 1$ where d is a power of two.)

The plaintext space can be set to any $\mathbb{Z}_p$ (as long as p can be represented by poly(d) bits), but in our case we will use a small plaintext space, say $p = O(d^{10})$. The public key specifies an (ideal) lattice $J \subseteq \mathbb{Z}^d$, and a ciphertext encrypting a message $m \in \mathbb{Z}_p$ is a point in $\mathbb{Z}^d$ whose distance from J is congruent to the vector $m \cdot \vec{e_1}$ modulo p. Homomorphic addition and multiplication is implemented via additions and multiplications in the ring $\mathcal{R}$, and the decryption formula of the scheme has the form of Equation (1).

Below let p be a small prime number, say $d^{10} < p < 2d^{10}$, denote $q = p - 1$, and let g be a generator for $\mathbb{Z}_p^*$. For the SWHE we use an instance of Gentry's scheme with "large" parameters and input space $\mathbb{Z}_p$, and for the AHE scheme we use another instance with "small" parameters and message space $\mathbb{Z}_q$. Below we denote the "large" instance by Lrg and the "small" instance by Sml.

The parameters are chosen so that the homomorphic capacity of Lrg is 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.[5]

A.2.2 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.) As explained above, the native message space of this instance of Gentry's scheme is $\mathbb{Z}_q$ (where $q = p - 1$), but we use it implicitly to encrypt elements in $\mathbb{Z}_p^*$. To encrypt an element $x \in \mathbb{Z}_p^*$ under Sml, we first find the exponent $e \in \mathbb{Z}_q$ such that $g^e = x \pmod p$ and then use the native encryption to encrypt $-e$. (The negation is used just for convenience, so that the secret-key dependent parts of Equation (1) appear with a positive sign.)

To decrypt a ciphertext under Sml we first use the native decryption to recover the exponent e and then exponentiate. Since the native decryption in Gentry's scheme is of the form of Equation (1), then decryption under Sml has the following formula

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

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.

[5]The "small" scheme could also be instantiated from other additively homomorphic lattice-based schemes, e.g., one of Regev's schemes [12, 13], or the GPV scheme [6], etc.

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 imitate the binary circuit for evaluating it. We have n numbers (represented in binary), each with κ bits, and we need to add them over the integers and then ignore the lower κ bits. Hence the result will be κ-bits long (and for all the intermediate calculations we only need to keep 2κ bits for each number).

We first use the 3-for-2 trick, repeatedly replacing each three of these numbers by two numbers corresponding to the XOR and CARRY bits. Over $\mathbb{Z}_p$ this is done using the formulas

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

Arranging these 3-to-2 reductions in a tree structure, we have a tree of depth $\log_{3/2} n$, at the end of which we are left with only two numbers to add, each with 2κ bits. Since each level multiplies the degree by 3, then the bits of these final numbers are polynomials in the initial bits, of degree $3^{\log_{3/2} n} = n^{\log_{3/2} 3} \approx n^{2.47}$.

Using the XOR and CARRY formulas from above, we can add these two numbers in binary, and each bit of the result will be a multi-linear polynomial of the bits of the two numbers, and hence of degree at most 4κ in those bits. Therefore each bit in the result of the "complicated part" can be computed as a polynomial mod p of degree at most $n^{2.47} \cdot 4\lceil \log(n+1) \rceil$ in the secret key bits. It can also be verified that the number of terms in these polynomials is less than $2^{2\cdot\text{degree}}$.

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

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

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 $n^{2.47} \cdot 4\kappa$), hence the entire term has degree bounded by $n^{2.47} \cdot 4\kappa \cdot \kappa$. 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 bounded below $n^{2.47} \cdot 4\kappa^2 + n \log q < n^3$. Again one can verify that since all the coefficients are from $\mathbb{Z}_p$ then the number of terms is bounded by $p^{O(\text{degree})} < 2^{n^3}$.

A.2.3 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.2.4 Setting the parameters.

Theorem 6. *Let* Lrg *and* Sml *be as above. We can choose the parameters of* Lrg *and* Sml *so that they are chimerically compatible schemes.*

Proof. Denote the security parameters λ_{Lrg} and λ_{Sml}. It is clear that the plaintext spaces of the scheme are compatible and Lrg has a suitable restricted decryption circuit with degree $D(\lambda_{\mathsf{Lrg}})$ and $B(\lambda_{\mathsf{Lrg}})$ products gates for polynomials D and$B$. It remains to establish that the schemes can satisfy Properties 2 and 3 of Lemma 5. Let α, β, γ be some functions such that:

- Lrg's homomorphic capacity includes all polynomials over $\mathbb{Z}_p$ of degree at most $\alpha(\lambda_{\mathsf{Lrg}})$ having at most $2^{\alpha(\lambda_{\mathsf{Lrg}})}$ monomials.
- Sml's decryption function can be expressed as a (ciphertext-dependent) polynomial of degree at most $\beta(\lambda_{\mathsf{Sml}})$ having at most $2^{\beta(\lambda_{\mathsf{Sml}})}$ monomials.
- Sml's homomorphic capacity includes products of degree up to $\gamma(\lambda_{\mathsf{Sml}})$.

We have seen that α and β are polynomial functions, and γ can be super-polynomial For convenience, say $\alpha(x) = \theta(x^{c_\alpha})$ and $\beta(x) = \theta(x^{c_\beta})$. Asymptotically, Properties 2 and 3 are satisfied by setting $\lambda_{\mathsf{Lrg}} = \lambda_{\mathsf{Sml}}^{c}$ for some constant $c > c_\beta / c_\alpha$. $\square$

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

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

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

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. $\square$

C Background on Lattice-Based Encryption

C.1 Background on Lattices

Before sketching a generic lattice-based encryption scheme, we recall some basic facts about lattices.

A full-rank (all lattices here all full-rank) n-dimensional *lattice* is a discrete subgroup of $\mathbb{R}^n$, concretely represented as the set of all integer linear combinations of some *basis* $B = (\vec{b}_1, \ldots, \vec{b}_n) \in \mathbb{R}^n$ of linearly independent vectors. Viewing the vectors $\vec{b}_i$ as the rows of a matrix $B \in \mathbb{R}^{n \times n}$, we have:

$$L = \mathcal{L}(B) = \{\vec{y} \cdot B \ : \ \vec{y} \in \mathbb{Z}^n\}$$

An *independent set* of L is a set of n linearly independent vectors from L.

To B we associate the half-open parallelepiped $\mathcal{P}(B) \leftarrow \{\sum_{i=1}^{n} x_i \vec{b}_i : x_i \in [-1/2, 1/2)\}$. The volume of $\mathcal{P}(B)$ is $|\det(B)|$ and is invariant among bases of L; thus, we denote it by $\det(L)$. For integer lattices, $\det(L)$ is the size of the quotient group $\mathbb{Z}^n/L$.

For $\vec{c} \in \mathbb{R}^n$ and basis B of L, we use $\vec{c} \bmod B$ to denote the unique vector $\vec{c}' \in \mathcal{P}(B)$ such that $\vec{c} - \vec{c}' \in L$. Given $\vec{c}$ and B, $\vec{c} \bmod B$ can be computed efficiently as $\vec{c} - \lfloor \vec{c} \cdot B^{-1} \rceil \cdot B = [\vec{c} \cdot B^{-1}] \cdot B$. (We will use the notation $\lfloor \cdot \rceil$ for rounding to the nearest integer or integer vector and $[\cdot]$ for the fractional part, mapped to $[-1/2, 1/2)$.)

Every lattice has a unique Hermite normal form (HNF) basis where $b_{i,j} = 0$ for all $i < j$ (lower-triangular), $b_{j,j} > 0$ for all j, and for all $i > j$ $b_{i,j} \in [-b_{j,j}/2, +b_{j,j}/2)$. Given any basis B of L, one can compute HNF(L) efficiently via Gaussian elimination. For this reason, the HNF is in some sense the "least revealing" basis of L, and is therefore often used as a public key [9].

Every lattice L has a dual lattice L^*, consisting of all vectors in $\mathbb{R}^n$ that have integer dot product with all vectors in L. For example, if $L = p\mathbb{Z}^n$, then $L^* = p^{-1}\mathbb{Z}^n$. In general, if B is a basis of L, then the inverse transpose of B is a basis of L^*. If L is an integer lattice with basis B, L^* is its dual with basis B^*, then $\det(B) \cdot \vec{e}_i \in L$ for all i, and $\det(B) \cdot B^*$ is an integer matrix.

C.2 A Generic Lattice-Based Encryption Scheme

Here, we sketch a generic lattice-based encryption scheme E_{lat}. There is nothing new here – quite the opposite. Our goal here is generality, to make it obvious that our techniques apply to many natural lattice-based encryption schemes – in particular, to SWHE schemes. Thus, E_{lat} is designed to encompass many natural lattice-based encryption schemes, in the sense that these schemes can be seen as an instance of E_{lat}. Since it is generic, E_{lat} does not include many natural optimizations, such as compactly representing the secret key as a vector, rather than as a matrix.

E_{lat}: A Generic Lattice-Based Encryption Scheme.

KeyGen: Generate a n-dimensional lattice L together with a "good" basis (or independent set) R of the dual lattice L^*. Generate $p \leftarrow \mathbb{Z}^+$ relatively prime to $\det(L)$, such that $\mathbb{Z}_p^n$ is a suitable plaintext space. Output $pk \leftarrow (\text{HNF}(L), p)$ as the public key and keep $sk \leftarrow R$ secret.

Encrypt(pk, $\vec{m} \in \mathbb{Z}_p^n$): Generate a random short "error" or "noise" vector $\vec{e}$, subject to the constraint that $\vec{m} = \vec{e} \bmod p$. Output the ciphertext $\vec{c} \leftarrow \vec{e} \bmod \text{HNF}(L)$.

Decrypt($sk, \vec{c}$): Output $\vec{m} \leftarrow [\vec{c} \cdot R] \cdot R^{-1} \bmod p$. ($R^{-1}$ is computed over $\mathbb{Q}^{n \times n}$.)

Correctness: We are being informal about the scheme's parameters, using terms like "good" and "short". What we need for correctness is the following: all of the coefficients of $\vec{e} \cdot R$ are less than 1/2 in magnitude when $\vec{e}$ is a properly short noise vector of a valid ciphertext. Then, assuming $\vec{c} = \vec{v} + \vec{e}$ is a valid ciphertext with $\vec{v} \in L$ and properly bounded noise $\vec{e}$, we have $[\vec{c} \cdot R] \cdot R^{-1} \bmod p = [\vec{e} \cdot R] \cdot R^{-1} \bmod p = \vec{e} \cdot R \cdot R^{-1} \bmod p = \vec{e} \bmod p = \vec{m}$.

C.3 Tweak to the Generic Scheme

To express decryption as a depth-3 circuit, we tweak the scheme slightly (similar to [3]) to obtain a simpler decryption formula. Currently, decryption computes $\vec{m} \leftarrow [\vec{c} \cdot R] \cdot R^{-1} \bmod p = \vec{c} - \lfloor \vec{c} \cdot R \rceil \cdot R^{-1} \bmod p$. After the tweak, decryption will compute $\vec{m} \leftarrow \vec{c} - \lfloor \vec{c} \cdot S \rceil \bmod p$, where S is the new secret key.

$\underline{E^*_{lat}\text{: Tweaked Version of } E_{lat}.}$

KeyGen: Run E_{lat}'s KeyGen algorithm to obtain $pk = (\text{HNF}(L), p)$ and R. Let q be an integer such that $q \cdot R$ is an integer matrix (this is true if q is a multiple of $\det(L)$) and $q = 1 \bmod p$ (this is possible by the Chinese Remainder Theorem). Let $A = (q \cdot R)^{-1} \bmod p$, with entries in $[-p/2, p/2)$. Set $sk \leftarrow S = A \cdot R$.

Encrypt(pk, $\vec{m} \in \mathbb{Z}_p^n$): As in E_{lat}.

Decrypt$(sk, \vec{c})$: Output $\vec{m} \leftarrow \vec{c} - \lfloor \vec{c} \cdot S \rceil \bmod p$.

Correctness: First, note that (assuming the parameters are chosen properly), S works as a decryption key in the original E_{lat} scheme – that is, $m \leftarrow \vec{c} - \lfloor \vec{c} \cdot S \rceil \cdot S^{-1} \bmod p$. This is because, as required for an E_{lat} secret key, S is an independent set of L^*. Moreover, though we will again be informal about the parameters here, it is clear that the parameters can be chosen so that, despite the fact that S has entries that are about $p \cdot \sqrt{n}$ times as large as R, its entries are still small enough to ensure correctness of decryption.

To show that our new decryption formula works, we need to establish that $\lfloor \vec{c} \cdot S \rceil \cdot S^{-1} = \lfloor \vec{c} \cdot S \rceil \bmod p$. Clearly $\lfloor \vec{c} \cdot S \rceil \cdot S^{-1}$ and $\lfloor \vec{c} \cdot S \rceil$ are both integer matrices if $\vec{c}$ is a valid ciphertext. Set $C = q \cdot S$. C is an integer matrix that equals I modulo p. We have:

$$\begin{aligned}
\lfloor \vec{c} \cdot S \rceil \cdot S^{-1} \quad &= \lfloor \vec{c} \cdot S \rceil \cdot S^{-1} \cdot C \bmod p \\
&= \lfloor \vec{c} \cdot S \rceil \cdot q \bmod p \\
&= \lfloor \vec{c} \cdot S \rceil \bmod p
\end{aligned}$$

C.4 Post-Processing a Generic Ciphertext

In the *old* blueprint, as part of the squashing step, the public key is augmented to include a "hint" about the original secret key, and a ciphertext is *post-processed* by combining it with that hint [3]. Specifically, suppose that S is the initial secret key, as in E^*_{lat}. The "hint" is a polynomial-size set of matrices $\{T_1, \ldots, T_N\}$ such that there is a secret *sparse* subset that sums to S. The post-processed ciphertext is:

$$\text{Post-processed ciphertext:} \quad \vec{c} \quad \text{and} \quad \vec{u}_i \leftarrow \vec{c} \cdot T_i \text{ for all } i.$$

The new secret key is the incidence vector $\vec{s} \in \{0,1\}^N$ such that $\sum s_i \cdot T_i = S$. Decryption computes $\vec{c} - \lfloor \sum_{i=1}^N s_i \cdot \vec{u}_i \rceil = \vec{c} - \lfloor \vec{c} \cdot S \rceil = \vec{m} \bmod p$.

In the *new* blueprint, we also use this post-processing step, but with an important difference: the subset summing to S is *not necessarily sparse*. The matrices T_i in the new blueprint contain no secret information, and therefore can be provided publicly as parameters without necessitating any new assumptions – e.g., an assumption about the hardness of the sparse subset sum problem (SSSP) [3]. In particular, the T_i's could simply be the set $\{E^{(ijkb)}\}$, where $E^{(ijkb)}$ is an $n \times n$ matrix of all zeros, except that the entry (i, j) equals

$(-1)^b \cdot 2^k/q$, where k ranges from 0 to $\lfloor \log q \rfloor$. All we need from the T_i's is the guarantee that, for any possible initial secret key S, there is some subset of the T_i's that sums to S.

$E^{\dagger}_{lat}$: E^{*}_{lat} with Post-Processing.

ParamSetup: Output a suitable set $\{T_1, \ldots, T_N\}$ of matrices, as described above.

KeyGen: Run E^*_{lat}'s KeyGen algorithm to obtain $pk = (\mathrm{HNF}(L), p)$ and S. Compute $\vec{s} \in \{0,1\}^N$ such that $S = \sum_{i=1}^{N} s_i \cdot T_i$. Set $sk \leftarrow \vec{s}$.

Encrypt(pk, $\vec{m} \in \mathbb{Z}_p^n$): As in E_{lat}.

PostProcess(pk, $\vec{c}$): Output $\vec{c}$ and the vectors $\vec{u}_i \leftarrow \vec{c} \cdot T_i$ for all i, where the $\vec{u}_i$'s are formatted as follows. For each coefficient u_{ij} of $\vec{u}_i$, split it into an integer part u'_{ij} and a fractional part u''_{ij}. Furthermore, for the fractional part, preserve only $\kappa \leftarrow \lceil \log(N+1) \rceil$ bits of precision: $u_{ij} = u'_{ij\,\bullet} u''_{ij}$, where u''_{ij} is a κ-bit integer.

Decrypt($sk, \vec{c}, \{\vec{u}_i\}$): For the j-th coordinate,

$$m_j \quad \leftarrow \quad c_j - \sum s_i \cdot u'_{ij} - \left\lceil 2^{-\kappa} \cdot \sum s_i \cdot u''_{ij} \right\rfloor \bmod p. \tag{3}$$

Correctness: If the u''_{ij} values had infinite precision, correctness would follow from the correctness of E^*_{lat}. We would have

$$\begin{aligned} & m_j \leftarrow c_j - \sum s_i \cdot u'_{ij} - \left\lceil 2^{-\kappa} \cdot \sum s_i \cdot u''_{ij} \right\rfloor \bmod p \\ & \Leftrightarrow m_j \leftarrow c_j - \lfloor (\sum_i s_i \cdot u_{ij}) \rceil \bmod p \\ & \Leftrightarrow m_j \leftarrow c_j - \lfloor \vec{c} \cdot S_j \rceil \bmod p \end{aligned}$$

where S_j is the j-th column of S.

It remains to show that the lost precision does not affect the result – in particular, the rounding part. For this, we need one more tweak to the scheme. We require that the parameters in E^*_{lat} are such that, if $\vec{e}$ is the properly bounded noise of a valid ciphertext, then $\vec{e} \cdot S$ has entries all less than $1/2(N+1)$ in magnitude (versus merely less than 1/2 in magnitude). Then, correctness holds, because the maximum magnitude of the lost precision is at most $N/2^{\kappa+1} \leq N/2(N+1) = 1/2 - 1/2(N+1)$. Combined with the fact that $\lfloor \vec{c} \cdot S_j \rceil$ is within $1/2(N+1)$ of an integer (this was the tweak), it is clear that the lost precision does not change the rounding.